A lightweight Framework for Improving the Mobile Applications Performance

Ibrahim A. Elgendy, Mohamed El-kawkagy, Arabi Keshk
Computer Science dept., Faculty of Computers and Information
Menoufia University, Egypt
Ibrahim.elgendy@ci.menofia.edu.eg, M_nabil_shams@yahoo.com, arabikeshk@yahoo.com

***Abstract*—In Recent years, the smartphones have exploded in popularity and set to become the fastest spreading technology. These devices have wide range of capabilities like more processing, more storage, GPS, Wi-Fi, camera, and others. As a result, the developer trends to build a complex application such as augmented reality, face detection, image processing and speech recognition. Although the capabilities that smartphones support, these applications require more processing and consume more battery. Therefore, the researchers trend to solve this problem by using rich resources such as cloud computing to support the mobile devices for execution these application. Where the mobile device offload these applications or part of it to be executed remotely on the cloud. This paper will introduce a lightweight framework to improve the performance of mobile applications and save the battery consumption. This framework have a core module which so-called dynamic offloader. This module determines dynamically at run time the decision for the offloading process based on the current environment. The experimental results are applied on Gaussian blur filter application and this results proved that this lightweight framework improves the performance and saves energy of mobile applications.**

Keywords—Smartphones; Android; Computation Offloading; Mobile Cloud Computing; Battery Consumption;Lightwight Framework;

I. INTRODUCTION

In recent year, smartphone devices have a wide range of capabilities such as GPS (Global Positioning System), Wi-Fi, cameras, huge storage and fast processors [11]. Therefore, developers trend to build complex applications such as augmented reality, face detection, image processing and speech recognition; also Smartphones' users can play games, upload videos, manage their personal health care and check bank account. All these applications require more processing power, memory and energy of battery [10].Although all the mentioned capabilities of Smartphones, battery life still the primary bottleneck for Smartphone devices to run these applications. There are three basic approaches [7] which are used to solve this problem:

1) *Upgrade Battery technology using transistors*: Although transistors are smaller and consume less power, battery needs more transistors for supporting better performance. Therefore, power consumption actually increases.

2) *Avoid wasting energy*: In this approach, system's components go sleep to save energy like dim the display while not using.

3) *Computation offloading*: It offloads intensive methods of mobile applications to run remotely on rich resource such as cloud.

Note that the proposed framework depends on the third approach that uses cloud computing as rich resource.

Cloud computing [14] refers to applications and services that run on a distributed network using virtualized resources and accessed by common Internet protocols and networking standards. It has been widely recognized as the next generation computing infrastructure. It also offers some advantages by allowing users to use computing, storage, services, and applications over the Internet at low cost. Cloud computing enables users to elastically utilize resources in an on-demand fashion.

In this paper, a novel framework which improves the performance of mobile applications and saves battery consumption of mobile is presented. This framework has dynamic offloader module which decides dynamically at runtime whether the application's methods run locally on mobile device or will be offloaded to the cloud. The framework applied only on Android applications. (Android applications are written in the Java language and can be written using any Development tool) [3, 8].

In recent years, a lot of studies have been appeared to support remote execution for mobile applications on the cloud to increase the performance and reduce energy consumption [5, 12].

Generally, there are two main approaches which are introduced to perform remote execution. The first approach is to use full process or full VM (Virtual Machine) migration [1, 6]. The full process or VM can be migrated to the rich infrastructure to execute remotely. While the second approach is only offload intensive methods or services of applications to execute remotely [2, 4, 9, and 13]. This approach leads to large energy saving because it is fine grained applications. This means that it can remote only the sub-parts that benefit from remote execution.

According to the first approach, CloneCloud [1] offloads parts of the unmodified application execution from the mobile device to the smartphone clone on the cloud. It used a combination of static analysis and dynamic profiling to partition applications to optimize execution time and energy. Static analysis identifies legal partitions of the application executable, while dynamic

profiler profiles the input executable on different platforms and returns a set of profiled executions. To offload process to clone, it requires VM to start which takes more seconds; therefore it is not a good solution for real-time applications. It also requires basic synchronization between smartphone and clone which are resource intensive. So offloading only intensive methods is better.

This followed by another approach so-called ThinkAir [6]. It exploits the concept of smartphone virtualization in the Cloud and provides method-level offloading to a smartphone clone executing in the cloud. It creates virtual machines (VMs) of a complete smartphone system on the Cloud. The Thinkair model uses profiler to monitor remoteable methods and execution controller which take the decision about these methods for running remotely on the cloud or locally on mobile. But this profiling process causes an overhead on smartphone.

According to the second approach, MAUI [2] supports fine-grained code offload to maximize energy saving. It uses dynamic application profiling and partition approach for partition offloading. But, similar to thinkair, this profiling process causes an overhead on smartphone.

After that, a framework that is only offloads intensive services of mobile applications to run remotely on the cloud is implemented. This framework is so-called Cuckoo [4]. It is designed for Android platform. Note that the offloading decision of this framework is static and the context unaware of parameters that effect on taking the decision. Therefore the dynamic offloading is better.

This followed by another model which is To cloud or Not to cloud [9]. It uses an algorithm to decide whether application will run locally on mobile or offloaded to the cloud. However, it considers the application as a single unit that cannot be decomposed into multiple methods. Although in some cases, it's better to execute some methods locally and offload others to the cloud.

Recently, Shiraz, M. & Gani, A. [13] proposed a lightweight framework for computational offloading. This framework is implemented for offloading the mobile application at service-level granularity to remote server nodes. But it requires synchronization between mobile device and virtual device instance on server node which are resource intensive.

The rest of this paper is organized as follows. Section II describes the proposed framework architecture and design goals. In section III, the results of the experimental studies are discussed. The paper ends with some concluding remarks in Section V.

II. FRAMEWORK ARCHITECTURE AND DESIGN GOALS

In Android applications, there are some intensive computation methods which consume mobile resources and battery. Therefore, this paper will propose a novel framework that uses the cloud resources to improve energy consumption. At runtime, this framework will decide which methods will run locally on the mobile device and which one will offload and run on the cloud. In this section, the proposed framework architecture is presented and also, how to develop the applications using it.

A. Architecture Overview

As shown in the Fig. 1, the proposed framework architecture consists of five main modules which include Network and Bandwidth monitor, Execution Time Expect, Dynamic Offloader, Mobile Manager and Cloud Manager that exist on cloud.

First, at runtime the Network and Bandwidth monitor module detects the network status of smartphone and current bandwidth value. At the same time the Execution Time Expect module predict the execution time for the method on mobile and on the cloud. Then, based on the result from Network and Bandwidth monitor and Execution Time Expect, the Dynamic Offloader module decides whether the methods of the application will be offloaded or not. If the decision is not to offload, then the Mobile Manager calls the local implementation of the methods and run on the mobile device. Otherwise, the Mobile Manager will communicate to the Cloud Manager and send the required data to execute the methods on the cloud. Then Cloud Manager executes the methods on the cloud and returns the result to Mobile Manager. Finally, Mobile Manager receives the results and then delivers it to the application on mobile device.

B. Architecture Modules

As mentioned before, the proposed framework architecture consists of five main modules. These modules will be described in details and also, how they communicate with each other.

1) Network and Bandwidth Monitor

The Network and Bandwidth Monitor module monitors the network and gathers information about it. This information helps the framework to determine if the mobile device has connection to the internet or not. This information also contains data about the current bandwidth of the network to show the quality of it. After that, this information is sent to Dynamic Offloader module which uses it later to take the decision.

2) Execution Time Expect

The Execution Time Expect module predicts the total time required to execute the method. Let C is the number of instructions involved in a method invocation, and the processor speed (instructions per second) of the mobile and the cloud are SM and SC respectively. If the amount of data that needed to send over network between mobile and cloud is D and the network throughput is T, then the time required to send this data is D/T.

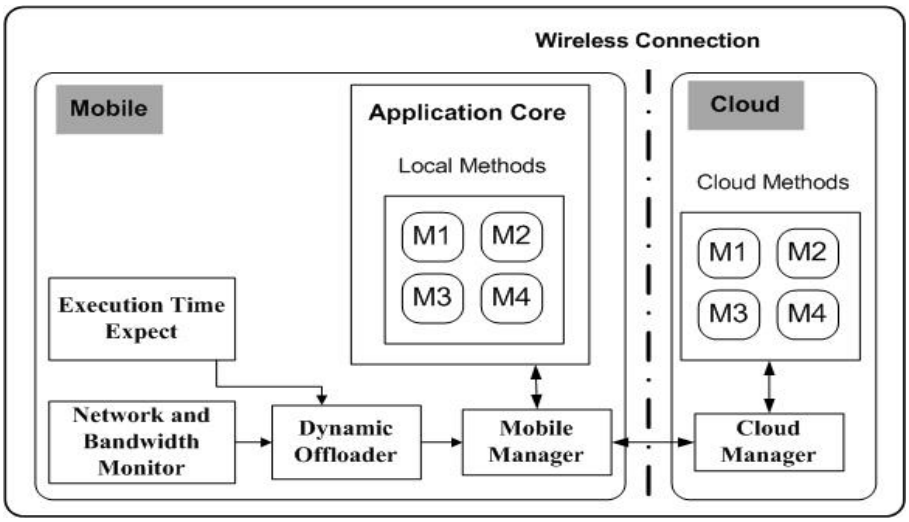

Fig.1. Proposed Framework Architecture

Then Execution Time Expect module can derive the relationship between execution speed and communication overhead for mobile and for cloud. Offloading of a method is beneficial if the following relation is true.

$$\frac{C}{S_M} - \frac{C}{S_C} > \frac{D}{T} \quad (1)$$

$$\frac{C}{S_M} > \frac{D}{T} \quad (2)$$

If the processing time on the cloud is ignored as cloud execute at very high speed (assume no workload on cloud), the relation becomes:

3) Dynamic Offloader

The Dynamic Offloader is considered as the backbone of the proposed framework. It decides dynamically at runtime whether the methods will be offloaded from mobile to cloud resource and executed there or will be executed locally on mobile device. First, it gets the information from the Network and Bandwidth Monitor and Execution Time Expect modules. According to the collected information, the Dynamic Offloader module will decide which methods of the android application will run locally on mobile device and which methods will be offloaded to the cloud.

4) Mobile Manger

The Mobile Manager module is responsible for running methods locally on mobile or offloads it to the cloud. After the Dynamic Offloader takes the decision; In case of the decision is to run methods locally, Mobile Manager will trigger application to run method locally. Otherwise, it will communicate with Cloud Manager Module and send the needed data for methods execution to it on the cloud. And finally receives the result from Cloud Manager Module.

5) Cloud Manager

The Cloud Manager module is written in pure java code. So any application can benefit from the proposed framework to offload its computation to any resource that run Java Virtual Machine (JVM). If the Dynamic Offloader decides that method will be offloaded and run on the cloud then, Mobile Manger communicates with Cloud Manger using the Ibis communication middleware [15]. (Ibis is an open source scientific software package developed for high performance distributed computing in Java. It offers a simple and effective interface that abstracts from the actual used network, being Wi-Fi, Cellular or Bluetooth. Since it is written in Java, it can also run on Android devices.) At first connection to cloud, the Mobile Manager sends jar file which created by Class and Jar Generator as shown latter. This jar file contains the needed methods that will be executed on the cloud and also the required libraries. Then, Cloud Manger can use this jar file to execute this method remotely on the cloud. And then transfer the result to the Mobile Manger module which delivers it to application.

C. Integration of Building Process

In this paper, only the android applications is applied. During the building process of any android applications, they are compiled and packaged into an APK file. This APK file is considered as a container for your application binary. If you are using the Eclipse as IDE for developing, the building process will be done automatically as you develop and save your code changes. If you are using other IDEs, this build process is done every time you run the generated Ant build script for your project. The build process of an android application will invoke the following builders in order [4]:

Android Resource Manager: In this step, Java file is generated to ease the access of resources, such as images, sounds and layout definitions in code.

Android Pre Compiler: In this step, Java file is generated from AIDL files.

Java Builder: In this step, Java source code is compiled and generated Java code.

Package Builder: This is the last step which bundles the resources, the compiled code and the application manifest file into a single file (apk) which can be installed on any Android device.

In the proposed framework, a new step is added to these builders that is inserted into an android project's build configuration. This step is called the Class and Jar Generator. It will create class that contains methods which may run remotely. After that it will generate jar file from created class that will send to the cloud. This step will be done after creating the java file using Java Builder, and before creating the java package using package Builder, so that the package file (apk) that generated from Package Builder will contain the jar file as show in Fig. 2.

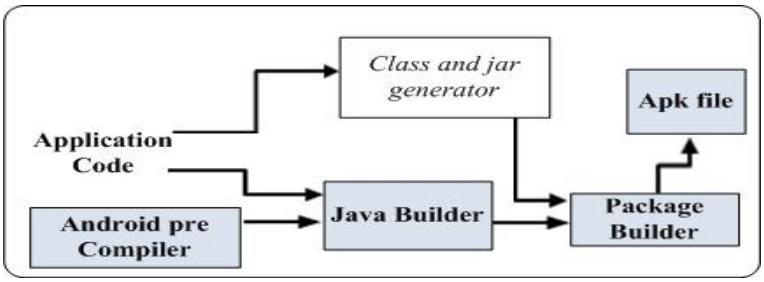

Fig.2. A schematic overview of Integration into the Build Process

D. Developing Application

To use the proposed framework in developing the application; the developer first create project and then write the source code. Then Class and Jar Generator will create class during the build process that contains the methods that will be executed on the cloud. Then, this class and required libraries that will be needed to run methods on the cloud is collected and jar file is generated from them. This followed by compiling the application code using Java Builder to generate the compiled code. Finally, the compiled code and the generated jar file will be packaged by Package Builder to generate the APK file. This Final APK can be uploaded to android market for using by mobile user.

III. EVALUATION AND ANALYSIS

The proposed framework is evaluated using Gaussian blur filter application. This application provides blur filter to make images blur. It applies Gaussian Blur algorithm where it utilizes Gaussian distribution to process images. The experiment evaluates the Processing Time, CPU Utilization, Battery consumption and Memory Usage when executing the application methods locally on the mobile and also when offloading them to the cloud using the proposed framework. The evaluation shows how this framework improves the performance of application.

A. Experimental Setup

In the evaluation, Samsung Galaxy S Plus GT-I9001 Smartphone is used in the experiments. It has Android platform with version 4.4.2, integrate with Wi-Fi interface which will be used in the experiments. It also, has Qualcomm MSM8255T CPU, 512 MB memory and battery capacity of 1650mAh at 3.7 volts. A laptop which acts as a cloud provider that host offloaded computation includes Windows-7 operating system with 64 bit, Intel Core I5 processor with 2.4GHZ frequency, 4 GB RAM, 600GB hard disk and 100Mbs network interface. In the evaluation, little eye V2.41 software is used to measure Processing Time, CPU Utilization, Battery Consumption and Memory Used.

B. Result of the Experimental

In the evaluation, two scenarios are used to measure Processing Time, Battery consumption, CPU Utilization, and Memory Usage. The first scenarios represents the execution of the application method locally on the mobile device, while the second one represents the offloading of the method for execution on the cloud using the proposed framework.

[1] Little eye: http://www.littleeye.co/

In the experimental result, five images are used with different resolutions. The experiments are executed nearly fifteen times and final the average is taken.

Fig. 3 shows the results of the processing time for Gaussian blur application when executing the method that applies Gaussian blur locally on mobile device and also when offloading it for execution on the cloud. The image resolution is represented in x -axis and the processing time in ms (Milli-Seconds) is represented in y-axis. As shown in the figure, the processing time when offloading is smaller than the processing time of running it locally on the mobile device. For example, the image with resolution 300*300 takes nearly 1363 ms when offloaded to the cloud while it takes nearly 4245 ms when executed on mobile. Moreover, the difference in processing time increases as the resolution of image increases.

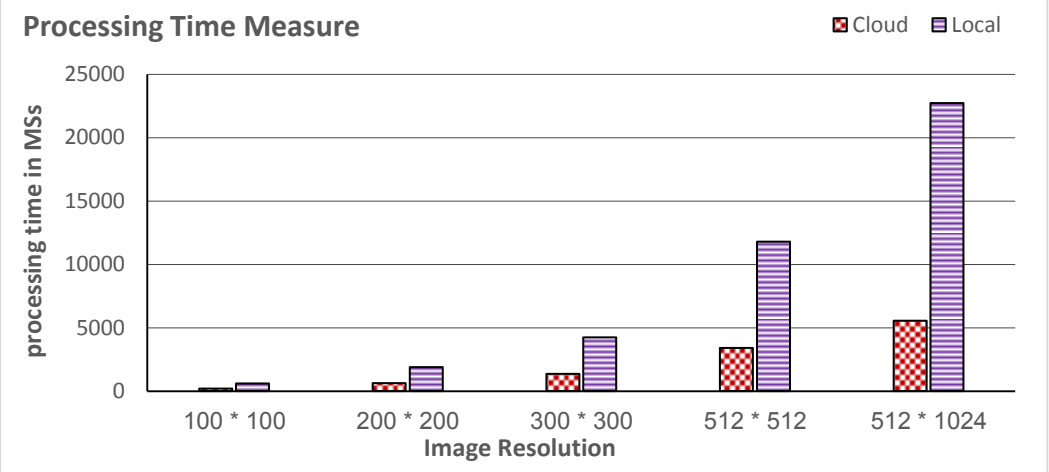

Fig.3. Processing time for Gaussian Blur Application on mobile and on the cloud

Fig. 4 shows the results of the Battery Consumption for Gaussian blur application when executing the method that applies Gaussian blur locally on mobile device and also when offloading it for execution on the cloud. The image resolution is represented in x -axis and the total battery consumed in mAh (Milliampere-hour) is represented in y-axis. The result proves that the consumption of the mobile battery in case of executing Gaussian blur method locally, while it is more efficient to offload it to the cloud. The total average battery consumption of the experimental for different resolutions of images is 3.6 mAh for running Gaussian blur method locally. This value is minimized to 2.9 mAh when offloading method to the cloud.

Fig. 5 shows the results of the Average CPU Utilization percentage for Gaussian blur application when executing the method that applies Gaussian blur locally on mobile device and also when offloading it for execution on the cloud. The image resolution is represented in x -axis and the average CPU Utilization percentage is represented in y-axis.

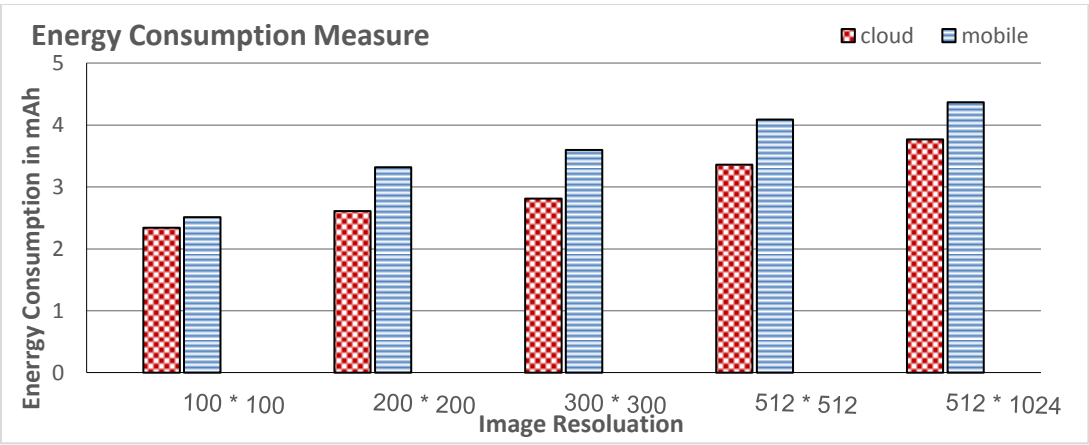

Fig.4. Battery Consumption for Gaussian Blur Application on mobile and on the cloud

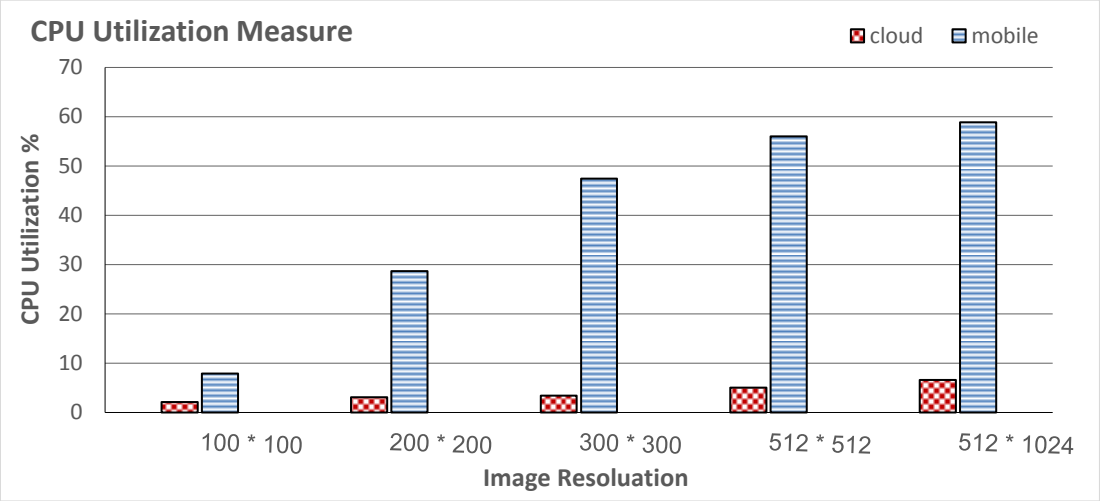

Fig.5. Average CPU Utilization percentage for Gaussian Blur Application on mobile and on the cloud

The result demonstrates the aggressive consumption of CPU resource when executing the Gaussian blur method locally on mobile device. While this CPU resource can be saved by offloading Gaussian blur method to the cloud using the proposed framework. The total average CPU Utilization percentage of the experiment for executing Gaussian blur method on the cloud nearly 4%, while this percentage is raised to become nearly 39.8% when running locally on mobile. So it is a good decision to offload the Gaussian blur method of the application for execution on the cloud.

Fig. 6 shows the results of the Average Memory Usage for Gaussian blur application when executing the method that applies Gaussian blur locally on mobile device and also when offloading it for execution on the cloud. The image resolution is represented in x -axis and the average Memory Used in MB is represented in y-axis. The result elucidates that executing Gaussian blur method locally on the mobile uses more memory space than offloading it to the cloud. The total average memory usage of the experiment for executing the Gaussian blur method on the cloud nearly 23MB, while this value is raised to become 24MB when running locally on mobile.

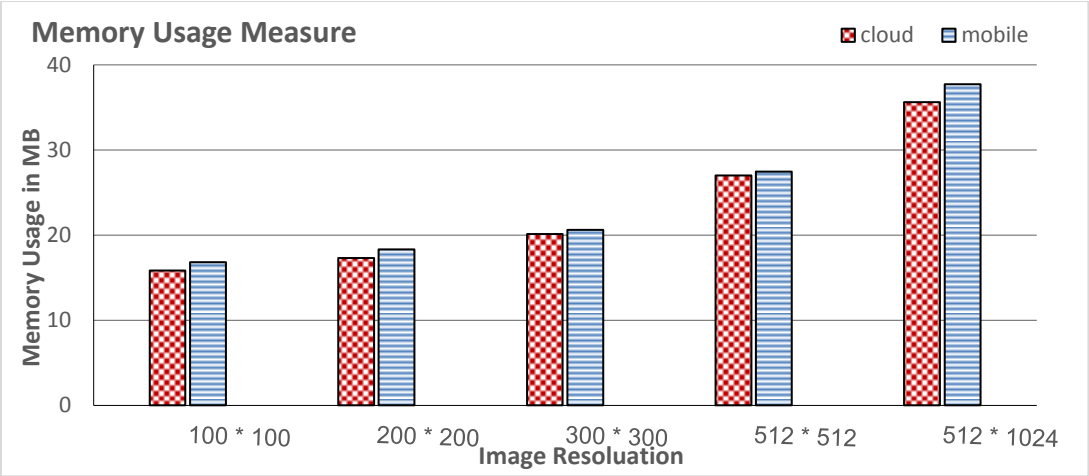

Fig.6. Memory Usage for Gaussian Blur Application on mobile and on the cloud

In general, the experimental results elucidates that the processing time is reduced from 11.8 seconds to 3.4 seconds for the 512 * 512 image resolution in Gaussian blur application when using the proposed framework application. It also reduced the battery consumption from 3.6 mAh to 2.9 mAh, CPU Utilization percentage from 39.8 % to 4% and memory usage from 24 MB to 23MB. Moreover, the difference increases as the image resolution increases. Finally, these results show the ability of the

proposed framework to improve the performance of mobile applications and save CPU utilization, memory usage and battery consumption. Note that in the future the proposed technique can be tested using different Benchmark [16]

IV.CONCLUSION

This paper proposed a novel framework to improve the performance of mobile applications and battery consumption. This framework offloads only the intensive methods of the mobile applications to be executed remotely on the cloud. This can be done based on the core module of the proposed framework which is dynamic offloader. The dynamic offloader module take its decision based on the network and bandwidth monitor module that monitor the current network connection and also the execution time expect module which predicts the execution time of these intensive methods.

The experimental studies and results proved that the proposed framework can reduce the processing time, battery consumption CPU Utilization and the Memory Usage for Gaussian blur application. Finally, we can generalize that the mobile applications which have more computation than the needed data for processing can benefit from the proposed framework. Where this framework will save the mobile's resources. The future work will focus on trying to build complex model that will take another metrics for making the offloading decision and also, enabling parallelization for executing the methods remotely on the cloud.

REFERENCES

[1] B.G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, "CloneCloud: Elastic Execution Between Mobile Device and Cloud," in Proceedings of the Sixth Conference on Computer Systems, New York, NY, USA, 2011.
[2] E. Cuervo, A. Balasubramanian, D.k. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, New York, NY, USA, 2010.
[3] M. Burton and D. Felker, Android Application Development For Dummies, 2nd ed., For Dummies, 2012.
[4] R. Kemp, N. Palmer, T. Kielmann and H. Bal, "Cuckoo: A Computation Offloading Framework for Smartphones," in Mobile Computing, Applications, and Services, vol. 76, M. Gris and G. Yang, Eds., Springer Berlin Heidelberg, 2012, pp. 59-79.
[5] A. Khan, M. Othman, S. Madani and S. Khan, "A Survey of Mobile Cloud Computing Application Models," Communications Surveys Tutorials, IEEE, vol. 16, no. 1, pp. 393-413, First 2014.
[6] 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 INFOCOM, 2012 Proceedings IEEE, 2012.
[7] K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," Computer, vol. 43, no. 4, pp. 51-56, April 2010.
[8] M. Murphy, Beginning Android, Berkely, CA, USA: Apress, 2009.
[9] V. Namboodiri and T. Ghose, "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, 2012.
[10] M. Sharifi, S. Kafaie and O. Kashefi, "A Survey and Taxonomy of Cyber Foraging of Mobile Devices," Communications Surveys Tutorials, IEEE, vol. 14, no. 4, pp. 1232-1243, Fourth 2012.
[11] M. Shiraz, M. Whaiduzzaman and A. Gani, "A study on anatomy of smartphone," Computer Communication & Collaboration, vol. 1, pp. 24-31, 2013.
[12] M. Shiraz, A. Gani, R. H. Khokhar and R. Buyya, "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, 2013.
[13] M. Shiraz and A. Gani, "A lightweight active service migration framework for computational offloading in mobile cloud computing," The Journal of Supercomputing, vol. 68, no. 2, pp. 978-995, 2014.
[14] B. Sosinsky, Cloud Computing Bible, 1st ed., Wiley Publishing, 2011.
[15] R. V. van Nieuwpoort, J. Maassen, G. Wrzesi ska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann and H. E. Bal, "Ibis: A Flexible and Efficient Java-based Grid Programming Environment: Research Articles," Concurr. Comput. : Pract. Exper., vol. 17, no. 7-8, pp. 1079-1107, jun 2005.
[16] E. Sarhan, A. Ghalwash, and M. Khafagy,"Specification and Implementation of Dynamic Website Benchmark in Telecommunication Area",In Proceedings of the 12th WSEAS International Conference onComputers, Heraklion, Greece ,2008.