IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

ANDROID OFFLOADING COMPUTING OVER CLOUD

[1] P R Sonawane, [2] Pragati Tomar , [3] Priya Pandey , [4]Basukinath Tiwari, [5]Rahul singh
Dept. of Computer Engineering
Army Institute of Technology, Pune, India

Abstract:In this fastest growing world of technology , usage of mobile phones has doubled to that of PC's . But the mobile phones are restricted because of the limited resources . Limited resources include CPU , battery and processing power(memory) . In this paper we are going to present a framework for offloading of those heavy tasks of mobile applications like Face Detection application on the cloud . By heavy tasks , tasks whose computation costs more in processing and time . It allows automatic offloading of heavy computation tasks to a standalone android virtual machine (cloud) . Different parameters are considered here to declare whether a task has to be offloaded or not . These parameters are how much energy the task is consuming , remaining battery life of the mobile device on which application is running .

Index Terms – android , cloud , offloader

I. INTRODUCTION

Across the globe mobile phones are widely used . And with the growing usage several computation intensive applications are blooming in the market . As mobiles phones are becoming smarter there is a boom in these applications but the device is still limited as we cannot increase the capacity of mobile phones beyond certain limits . So as discussed in paper [1] we are switching to Mobile Cloud Computing . In the backend we have a complete support system to overcome these difficulties of efficiently running all these applications involving heavy tasks.Google photos and Apple iOS's Siri are examples of these code offloading techniques as discussed []. Many frameworks have been proposed since then . But most of them are not so convenient for developers working out there.In this paper we are proposing a framework for offloading of computation intensive tasks of applications(face detection application) with the use of an already existing framework as mentioned in paper [1] . We are offloading it to a remote server . The framework does not require any changes to be made in the android device side.The static analysis is done to make the decision making more fast and light than the previous techniques. This framework will empower the application to offload its compute intensive part to the cloud via the internet after analyzing the cost of offloading over the cost of running the application on the phone itself. The analysis will be done using parameters like input size and internet connectivity.
The remainder of the paper is organized as follows . In section II the existing technologies related to offloading and in section III we summarize the work done related to the idea we are working on . Section IV describes the design of the framework which we are proposing and all the architecture of our framework . The conclusion is presented in section VIII.

II. LITERATURE SURVEY

It is very vital to recognize the right technique with which we can go forward for our work .
So in this section we will be discussing some of already existing techniques about which we read in depth . There are several research works proposing different code offloading techniques for improving the application's performance and minimizing the energy usage by using resources in the cloud as done in paper[2]. They are discussed below:

1. MAUI provides method level code offloading for .NET applications. This is a realtime framework as it makes its decisions at runtime as mentioned in paper[2]. It offers energy-aware offloading from the mobile device to the remote infrastructure.The MIUI is more dependent on the hardware structure of its hosts and this leads to its drawback .Mobile devices typically have different CPU instruction architecture than desktops and servers. Processes cannot easily run on devices with different architectures when using MAUI.

2. CloneCloudis system that automatically transforms mobile applications to benefit from a cloud environment.The weakness of CloneCloud is that security is assumed basing on the trusted virtual machines, all the device data (media files and user information) is copied in the cloud to ensure a synchronized virtualization between the device and its clone .
3. ThinkAir provides on demand resource allocation and parallel execution on the cloud operating at method level. It requires the developers to indicate methods that could be executed in the cloud, and it focuses on minimizing energy consumption of the mobile devices by paralleling method execution using multiple VM images and cloud scalability by dynamic allocating VMs on demand.The weakness with ThinkAir is that, it assumes a trustworthy cloud server execution environment: there is hope that whenever data is offloaded to the cloud, the code and state of the data are not maliciously modified or stolen.
   Our technique provides method level code offloading for native methods, not for a specific application method, differently from the others. Through this approach, all applications that use those native methods may benefit from it. Besides that, no modification in the application or customization in the compilation process is required, saving programmer effort. The static investigation is done to settle on the basic leadership more quickly and light than the past methods. This structure will enable the application to offload its register serious part to the cloud through the web in the wake of breaking down the cost of offloading over the cost of running the application on the telephone itself. The investigation will be finished utilizing parameters like info size and web network. By utilizing this structure, the engineers will enable the applications to offload themselves without the need of some other application to examine and offload parts of the application.

III. FRAMEWORK DESIGN

The Framework has Five main elements:

(i) Face Detection program : The Face Detection program is executed in the main smartphone and is implemented as a normal application. It consists of code to offload the data and it uses methods from the Android Framework.

(ii) Helper tool : A normal program is created using resource files in xml that afterwards generates a java file by aapt tool. The interfaces are written in aidl for the service components and are converted to java files by a tool called aidl. Java files then generated along with other files are compiled by Javac and finally packed in apk file to be installed on an android device. The developer needs to make change in the source code to offload task.The aidl tool that comes with Android SDK generates interface defined in a .aidl file into a Java interface class . An inner abstract class named is present in this interface class stub which implements methods from the AIDL interface.The developer need to modify this stub inner class so that it can be used to redirect method invocations from an activity. The developer also modifies AndroidManifest.xml and service source code manually in order to create service wrappers to assist offloading.

(iii) Service offloader : The framework doesn't modifies the main Android platform or it doesn't need root privilege because it is present in the application layer. The users download and install the client application which helps to offload tasks which have the offloading code present in them and those who follow our Determination Standards.Whenever offloading needs to be done calls are made to methods which are matching in the proxy and the activity invokes some methods in the service.The decomposed objects are then taken care of by the proxy by decomposing the original source code into understandable parts by the OS.

(iv) Virtual Smartphone : The Server is run as a service in Android Virtual Machine so that the offloading requests are handled from the main smartphone.The server has pre-configured image of the android along with the address i.e. IP.Whenever task offloading is finalized the Service Offloader in the physical smartphone tells Service Manager in the Virtual Smartphone. Server Manager after listening to it downloads the corresponding service component and with Remote Service Wrapper from the common application distribution server,it then installs it in Virtual Smartphone.If the Determination Standards on the physical smartphone device decides to offload a service task to the Virtual Machine, the external stub in Local Service Wrapper redirects the invocation from the activity to Service Manager. It then finds the matching remote service and then redirects the invocation to the dummy activity in Remote Service Wrapper. The dummy activity is then used to make an internal call to the stub inside the service component in Remote Service Wrapper. The output is then returned back to the outer stub of Local Service Wrapper using the proxy in the original calling activity along the same address.

(v) Xposed Framework module : Specifically for the face detection application Xposed framework is utilized. It is used to connect both components i.e. the face detection application and the Virtual Machine. What it does is that it modifies the detection method of the class vision.face.FaceDetector. So whenever the offloading is done instead of locally executing the code for detection it sends the message to the Server application and then waits for its response which then returns the appropriate result. A class must be implemented in the interface Serializable so as to enable its objects to be sent by the network to the designated address. Since the two classes Frame and SparseArray, which are used by face detection methods, does not implement this interface, this needs to be handled by manually serializing or inserting into those classes the offloading code using the GSON library. We have chosen

GSON, as among the few open-source projects which can convert Java objects to JSON, among most of them they require the developer place Java annotations in the classes. Since we do not have access to the source code this is tough. Even most of them also have not fully supported the uses of Java Generics.

IV. DETERMINATION STANDARDS

There are three major considerations while deciding on whether to offload the task or not. These include Consumption of Energy, Battery remaining, and the time taken to execute on both the devices. Combining all these facts the final results decides whether to offload the application on the virtual machine or not. Here physical smartphones are called clients and Virtual Smartphones are called servers.

The first condition involves energy consumption for the client.

Pc = Power used for computation (watt)

Pi = Power used for remaining ideal (watt)

Pn = Power used for sending and receiving information (watt)

Energy Consumed by the client is in watt-second (joule).

Assuming the server is even involved then the total energy consumed will be even including

Communication overhead(joule) + Energy used while server was idle(joule)

Mathematical equation for this is:

The second condition deals with time taken to complete the task. Assuming,

C = total instructions in method invocation

Sc = Client processor speed (instructions per second)

Ss = Server processor speed (instructions per second)

D = Data transferred between client and server

T = Network Throughput

L = Network Latency

u (subscript) = upward (client to server)

d (subscript) = downward (server to client) transmissions

So, the time it takes to transfer data is D/T.

The relationship between execution speed and communication overhead is thus mathematically represented as: The task is transferred to Virtual Smartphone only if the server's process speed is extremely greater than the cost involved in offloading. As we can see from the above two standards that compute-intensive tasks like generating large prime numbers or solving problems in chess involve offloading when perform in a weak device for greater performance.The thinvolves seeing the remaining battery life before making the decision.The task should be completed locally if it consumes more energy or battery than is left in the device to upload the data and get back the results. But if it suffices then it can go on and perform the offloading on the server. The server can provide the result when the client is back online. Mathematically it is expressed as:ird standard In real-world there are a large number of standards that can be involved in deciding but these three are the major ones that include the decision-making process of offloading.

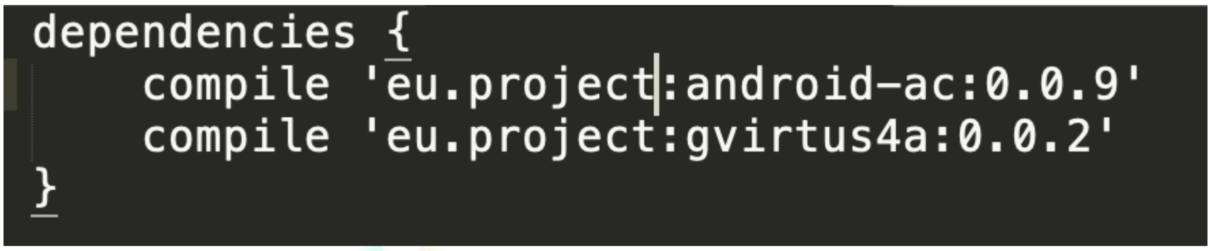
V. OFFLOADING ANDROID APPLICATION CODE

- **Java Method Offloading in Android**

We will start with an Android/Java class that implements a compute intensive method.

To make this class offloadable, we simply do this:

Include the libraries in the *build.gradle* file :



```
dependencies {
    compile 'eu.project:android-ac:0.0.9'
    compile 'eu.project:gvirtus4a:0.0.2'
}
```

- Next, change the source code in the following way:
    - Make the main class *extend* Remoteable, which is an abstract class.
        - The Remoteable class implements the Serializable Java interface, which means that the main class should also be serializable.
        - Moreover, the abstract method copyState() should be completed.
    - Add a @Remote Java annotation to the main method so that that this method is considered for offloading.

Declare an instance variable transient PRT controller; and initialize it on the constructor with a PRT object that will be passed to the class from the main class.

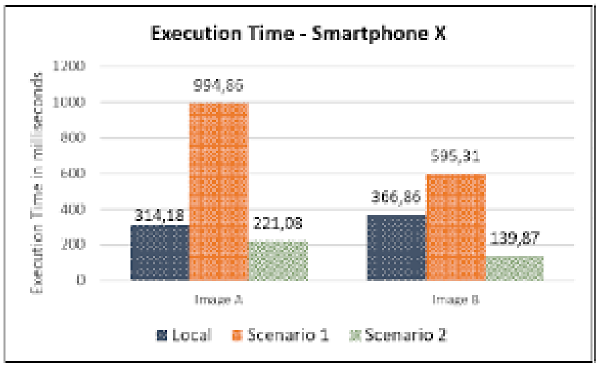
First a PRT PRT object should be created in the main activity of your program.

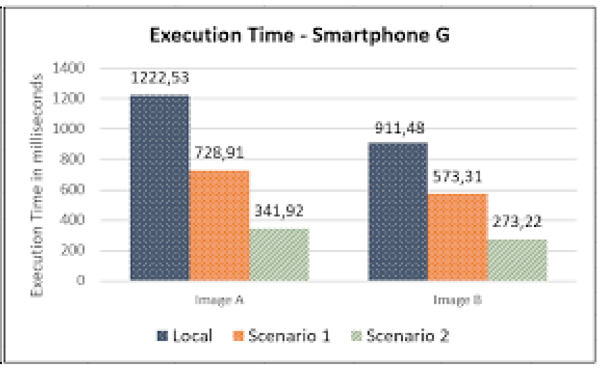
- The PRT class is implemented as Java Singleton and exposes two static public methods for getting an instance:
    - IP of the Vm needs to be specified first.
    - The second performs a registration with the architecture and allocates a VM on the cloud (currently works only for internal testing).
- Pass the PRT object to the constructor of the main when you create the new object.
- Call the PRT.onDestroy() method in the onDestroy() method of your main activity.

- Now when running the method main() will be executed via the PRT locally on the device or remotely on the VM.

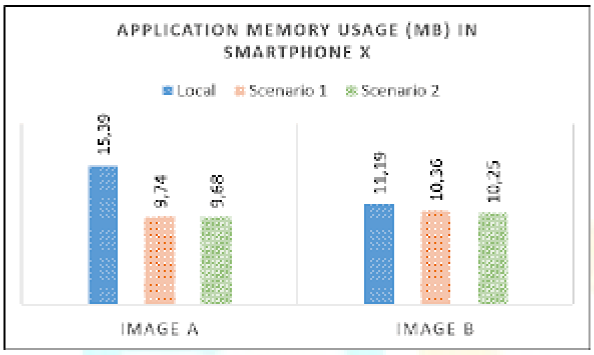
VI . RESULT

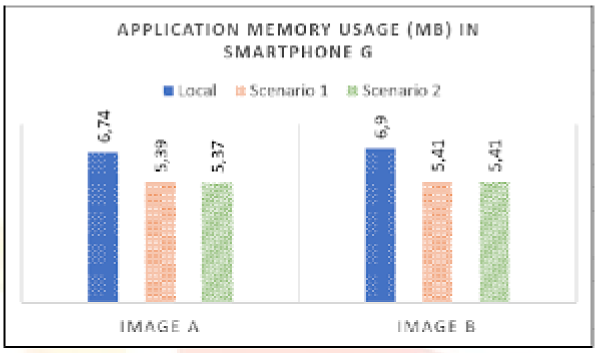
We will be analyzing the execution time first. The following figures will be obtained after that which shows the comparison of execution time for local and remote executions in both scenarios .





Now we will be analyzing both images in scenario 1, Smartphone X takes longer to run the method remotely than to run it locally. In scenario 2, the execution time will be 29,63% faster for image A and 31,88% faster for image B. However, when using Smartphone G and running the method remotely, the execution time will be performing better in both scenarios and images. In scenario 1, the execution time will be 40,37% faster for image A and 37,10% faster for image B compared with the local execution time. In scenario 2, the execution time will be 72,03% faster for image A and 70,02% for image B. Since Smartphone G has a less powerful hardware than smartphone X,it can benefit more from code offloading. Scenario 2 will be showing the best performance in this context.We will be profiling the application to get the network usage. The data transferred should be the same for both smartphones, however, in some cases, the number of faces can be different due to ACCURATE MODE set in the FaceDetector in the service application. Due to the higher capacity of the VM, we will enable the ACCURATE MODE flag to increase in accuracy when executing the method remotely. Smartphone X will find 23 faces in image A running the method locally, while running the method remotely the application service will find 24 faces in scenario 1 and 25 in scenario 2. Smartphone G will find 22 faces in image A by running the method locally, against 25 by running it remotely. We will also be profiling the application to get the memory usage. Following figures represent the application memory allocated after the execution of the method detect.





In all cases, less memory will be allocated when executing the method remotely. Smartphone X will allocate in average approximately 40% less memory for detecting the faces from image A and in average approximately 7,95% for image B. Smartphone G will allocate in average approximately 20,18% less memory for detecting the faces from the image A and in 21,59% for the image B.

VII. CONCLUSIONS AND FUTURE WORK

Current technology stats show that mobile devices are becoming an integral part of daily routine, but with limitations on processing and storage.In this paper, we have presented a method to code offloading, where some computation-intensive tasks can be transferred from an android server to a remote server so that the execution time of the task and power consumption can be reduced significantly.As a future work, the proposed approach to address the issue will be tested and refined on multiple platforms such as the iOS mobile platform and Windows Mobile other than the Android platform.Also migrate part of the code to the remote server may undermine user privacy. In addition, information transmission between server and phone is very vulnerable to interception.

REFERENCES

[1] Smartphone Market Share, http://www.idc.com/getdoc.jsp?containerId=prUS23503312

[2] Apps drain battery power,www.droidforums.net/forum/droidrazr -support/216454-battery-drain.html

[3] Android Application Requirements, www.netmite.com/ android/mydroid/development/pdk/docs/systemrequirements

[4] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, Bharat Bhar-gava. “A Survey of Computation Offloading for Mobile Sys-tems”. In the Journal of Mobile Networks and Applications, Springer, April, 2012.

[5] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, “A survey of computation offloading for mobile systems,” Mobile Networks and Applications, vol. 18, no. 1, pp. 129–140, 2013

[6] L. Jiao, R. Friedman, X. Fu, S. Secci, Z. Smoreda, and H. Tschofenig, "Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities," in Proceedings of Future Network and Mobile Summit. IEEE, 2013, pp.1–11

[7] 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. ACM, 2010, pp. 49–62.

[8] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. Wireless communications and mobile computing, 13(18):1587–1611, 2013.

[9] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. Future generation computer systems, 29(1):84–106, 2013.

[10] J. Flinn. Cyber foraging: Bridging mobile and cloud computing. Synthesis Lectures on Mobile and Pervasive Computing, 7(2):1–103,2012.

[11] M. Othman, S. A. Madani, S. U. Khan, etal. A survey of mobile cloud computing application models. IEEE Communications Surveys &Tutorials, 16(1):393–413, 2014.[15] Pixabay.Image

a.https://pixabay.com/en/smile-laugh-portrait-close-joy-1491429/.[16] Pixabay.Image

b.https://pixabay.com/en/family-kids-happy-people-mother-521551/.

[12] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck,K. Joshi, and K. Sabnani. An open ecosystem for mobile-cloud convergence. IEEE Communications Magazine, 53(3):63–70, 2015.

[13] A. Bergmayr, et al., "UML-based Cloud Application Modeling with Libraries, Profiles, and Templates," 2nd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE), Sep.2014.

[14] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb,"Simplifying cyber foraging for mobile devices," in Proceedings of the 5th international conference on Mobile systems, applications and services, New York, NY, USA, 2007, pp. 272–285.

[15] Md Whaiduzzaman, Abdullah Gani, and Anjum Naveed. An empirical analysis of finite resource impact on cloudlet performance in mobile cloud computing. In CEET-2014 ,CEET,2014.