 ISBN: 978-93-86005-02-1

OCRA: Offloading of Code of Recursive Algorithm from Smartphone to Cloud to Enhance Performance

Deep Mandal*, Koushik Dey, Debdip Ghosh and Diptabrata Saha

Department of Computer Science and Engineering, RCCIIT, Kolkata – 700015, West Bengal, India; deep.mandal106@gmail.com, deykoushik89@gmail.com, debdip20@gmail.com, diptabrata1994@gmail.com

Abstract

Developments in mobile computing have changed user preference for computing in modern era. However, Smartphones have limited resources - smaller memory and cache, less powerful processors, limited screen size, etc. Therefore, it take too much time for computational intensive jobs. One way of improving performance of Smartphones is to offload its computation intensive tasks to more powerful servers, keeping the GUI to the phones[9,10]. As the Smartphone and the backend server have different Software and Hardware Architectures and different Memory address Space, to migrate an executing process we need to clone the Stack, Heap and Process Control Block. In case of Recursive algorithms, additionally, we need to migrate the Data and the Control Stacks. A network monitor should check the network condition and take a decision dynamically for offloading. This paper introduces a system prototype named OCRA that shows such an offloading scheme using the Tower of Hanoi (TOH)[8] as an example. The Smartphone is connected to the server through Wi-Fi. The system can perform multiple back and forth offloading between the server and the Smartphone. The system uses URLClass Loader[3] to load the TOH class once into the JVM from Network at the beginning of the offload and uses Java Reflection API[4,5] to instantiate and dynamically invoke the Tower of Hanoi methods by examining Stack Object. The system was tested and achieved more than 50% improvement in performance for higher values of N (no. of disks) as compared to execution on Smartphone alone.

Keywords: Cloud Computing, JVM, Network Monitoring, Smart Phone

1. Introduction

In modern era, Smartphone become one of the most important parts of human life. However, Smartphone is more advanced than previous but it is still slower than any computers or servers. To improve performance of execution of a Recursive Algorithm in Mobile we have implemented code-offloading mechanism. Code offloading is a process through which we can send computation from Smartphone to cloud dynami-

*Author for correspondence

cally, continue rest of execution or some partial execution in cloud, and get the result to the Smartphone. This is used to enhance the performance of Smartphone. For Iterative Algorithms it is much easier to offload a code because we need to send only the data to the server to resume the execution. But for Recursive Algorithms the complexity is higher. Because it is not possible to resume the execution on the server if we only send the data. For recursion, we need to send the current state and the control stack of computation also. Here we send the control stack of execution in mobile. Then we reconstruct the tree to the server according to the control stack and resuming the execution from the state.

2. Methodology

2.1 Dynamic Class Migration

In our offloading System two versions of the application is designed, one is Android version, meant to be run on the phone and another is JAVA version that runs on the server. Android uses Java as its Application Programming Interface. But it does not use Java Class files. Its compile and build process is different from simple Java compilation and build process.

The process we are following to offload class file is given below:

- There is a Directory in Android named Assets. Here we can insert anything external to the application. So, here we are inserting our simple Java version Tower of Hanoi code.
- Then we read the file at runtime in Android and send it as a Byte Stream to the Server.
- Server then compiles it by using Java Compiler Class and creates the Class file at runtime. And then by passing the URI of the Class file to the URL Class Loader Class we are loading the class inside the Server JVM.
- Then using the Java Reflection API, we are successfully creating the Object of that loaded Class. And we can also invoke its Methods or modify its Fields at Runtime using Java Reflection.

Besides the Tower of Hanoi code offload we are also sending the Stack, a custom created Stack Class created by us to resume the Recursive algorithm in Server. The process to send the Stack Class is given below. For serialization and deserialization at Runtime we are using Google Gson API. Because it can create a JSON string to an equivalent Java Object. And it can also create a Java Object from a JSON String.

- At first we are getting the equivalent JSON string of the runtime Stack Object using Gson in Android.
- Then we are sending the string through network to the Server.
- Then creating the Stack Object at Runtime with the JSON string using Gson which is equivalent to the Object in Android.

2.2 Execution Offloading

Our the system performs execution offloading by suspending the thread executing in the mobile devices and captures the state of the thread at that moment to send the state over the network to the server. At the server end, a new object is build up at the transfer state to start a new thread there, to continue the execution at that point where it was suspended at the client.

In our system to offload the execution we need to suspend the Tower of Hanoi execution in Android Mobile and then serialize the Tower of Hanoi Object and the Stack object. Then send it through the network and deserialization is done at Server end. Then using the Stack, we are invoking the Tower of Hanoi object methods. In our Android application there are three Edit Texts. One Edit Text is used for Server IP Address, one is for Port Number and the last one is for Tower of Hanoi N. And there are two Buttons. One is for start computation in Mobile. Other one is for clear the output. When Start Compute in Mobile

button is clicked then a popup will appear showing the background execution progress with an Offload button. When Offload button is triggered then following things happens.

- Application receives an interruption request.
- Then the application will complete its execution of current Tower of Hanoi Node.
- After completing the execution of the current Node it then interrupts the execution.
- At last it will offload the Tower of Hanoi and the Stack to the Server.

In the server then the following things happens.

- Server receives the Tower of Hanoi and Stack and instantiate its objects.
- It will examine the Stack which is holding Control information and as well as Data related to the Control Information.
- Then it will resume the Tower of Hanoi execution from the exact point in which it was stopped in the Mobile by popping out the Stack.
- We are generating Random Numbers then we are checking the no. of Tower of Hanoi moves with the Random Number and if it exceeds the Random Number then we are Offloading it. This process takes place both in Mobile and Server. Thus we are performing Back and Forth Offload.

2.3 Thread Serialization

Java doesn't support thread serialization, so instead of serializing the thread, we serialize the Objects. We are using Google Gson API for Object Serialization and Deserialization. Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also convert a JSON string to an equivalent Java Object. Gson provides simple to JSON() and from JSON() methods to convert Java objects to JSON and vice-versa. It supports Java Generics and allow custom representation for objects and also supports arbitrarily complex objects. JSON (JavaScript Object Notation) is an open standard format that uses human readable text to transmit data objects consisting of attribute-value pairs. JSON is a language independent data format and it derives from JavaScript. In our system prototype before Serialization of Objects in Android Mobile we are suspending the Running Threads that are using those Objects because Google Gson is not Thread Safe. Then the Gson API gives us a JSON string representation on those Objects. After that we are sending those Strings through network to the Server. Then Server also uses Gson API to transform those JSON String to Java Objects which is a perfect clone of the Client Objects. And then we are creating separate Threads for computations using those Objects.

2.4 How Java Reflection is used

Java Reflection is used in our system prototype. When the server is setup and started, it opens a socket and goes on an infinite loop and waits for client's request. As soon as the server receives the request from client then it reads the required Tower of Hanoi class and Stack from the input stream and then by using Java Reflection API, we first instantiate the Object of the receiving Tower of Hanoi Class by using new Instance() method of the class Class. Then finding a method which matches a particular list of arguments by using get Declared Method(). Then we are setting the accessibility of the method to true. After that we are getting the field variables by using get Declared Field() and again setting the accessibility of the fields to true to set the field variables as it was in the client machine. Then we are invoking the methods using invoke(). So, using Java Reflection we are instantiating the objects of the classes which is received through network and dynamically changing their field variables and invoking the methods of the class.

3. System Design

In the client server architecture, the term client refers to Smartphone and server can be available through the local network (desktop or laptop connected with Wi-Fi) or internet (cloud server). Transmission Control Protocol (TCP) is used to communicate between client and server.

Detail division of the client and server process of the system is shown in below figure. The components of the clients are as followsz;

APP: User application named OCRA that need to be run in Smartphone.

Local Executor: Executes the user application entirely on the mobile (if cloud offload button is turned off) or partially for the back and forth approach.

Remote Executor: Executes the user application on server if cloud offload button is turned on.

Service Requestor: Open a connection with server and requests for a service.

Class Sender: Offload the class from the client to the server process.

State Sender: Responsible for sending the current state of user app from the client mobile to the server.

State Receiver: Responsible for receiving the state of execution thread running on the server.

Result Receiver: Receives the result from the server.

The server process are as follows:

Request Acceptor: Responsible for opening a connection and waiting on those connections to accept service request from the client.

Class Receiver: Responsible for receiving the application classes from the client's class sender component.

State Receiver: Responsible for receives the execution state of the clients thread on the fly from the state sender components of client.

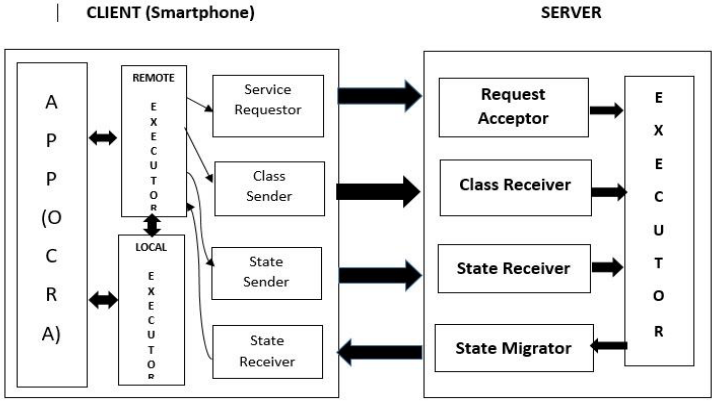


Figure 1. Client Server Architecture

State Migrator: Sends the execution state of server thread when required by the client's state receiver component on the time of back and forth execution.

Executor: Executes the task and send the result to the client's result receiver component.

4. Experimental Results

Specification:

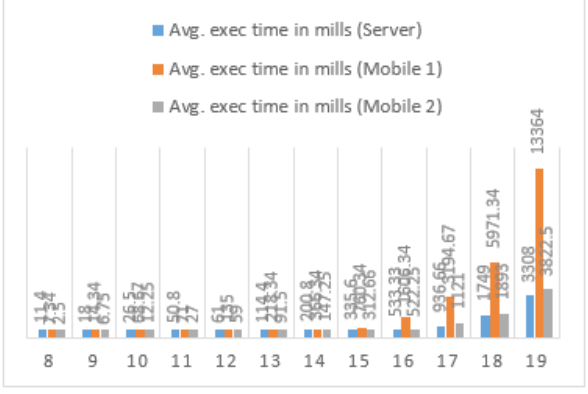
Server: **Processor:** Intel Core i5-3210M CPU @ 2.50GHz (2 Cores and 4 Threads)
RAM: 4.00GB
System Type: 64-bit Operating System, x64-based processor
OS: Windows 10

Mobile 1: **Processor:**Qualcomm Snapdragon 410, 1.21G
Architecture: 4x ARM Cortex-A53 @ 1.21 GHz
RAM: 1.00GB
OS: Android 4.4.4

Mobile 2: **Processor:** Qualcomm Snapdragon 801, 2.27GHz
Architecture: Krait 400
RAM: 3.00GB
OS: Android 5.1.1

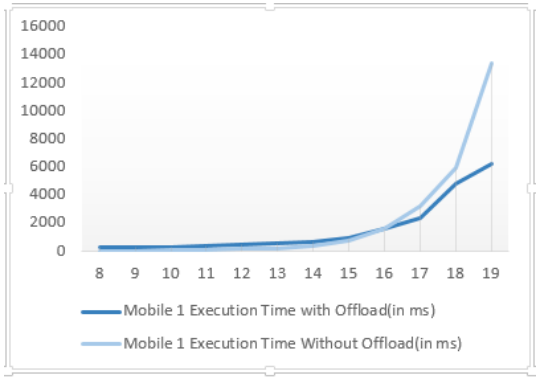
4.1 Without Interruption (Server vs Mobile)

N	Avg. exec time in mills (Mobile 1)	Avg. exec time in mills (Mobile 2)	Avg. exec time in mills (Server)
8	7.34	2.5	11.4
9	14.34	6.75	18
10	68.67	12.25	26.5
11	71	27	50.8
12	135	59	61
13	218.34	91.5	114.4
14	366.34	147.25	200.8
15	760.34	312.66	335.6
16	1606.34	522.25	533.33
17	3194.67	1121	936.66
18	5971.34	1893	1749
19	13364	3822.5	3308



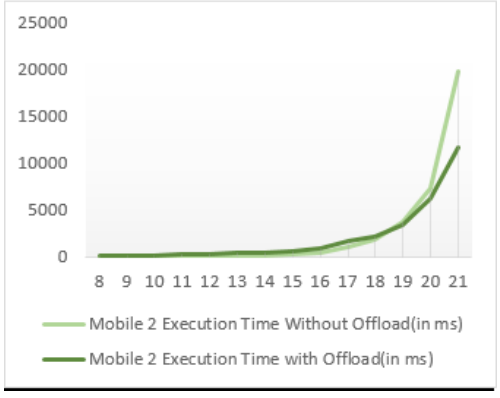
4.2 With Offload (Back and Forth) Vs Without Offload in Mobile 1

N	No. of Offload (Back and forth)	Part Execution Time with Offload In Server (in ms)	Part Execution Time with Offload In Mobile 1 (in ms)	Overall Execution Time with Offload (in ms)	Mobile 1 Execution Time Without Offload (in ms)
8	2	5	31	249	7.34
9	2	3	19	261.34	14.34
10	4	4	23	286.34	68.67
11	4	8	25	408.67	71
12	4	9	61	479	135
13	6	24	60	520.67	218.34
14	6	21	215	616.34	366.34
15	8	84	260	890.67	760.34
16	8	78	960	1546.67	1606.34
17	10	285	1160	2308.34	3194.67
18	10	286	3718	4776	5971.34
19	12	1088	3695	6254.34	13364



4.3 With Offload (Back and Forth) Vs Without Offload in Mobile 2:

N	No. of Offload (Back and forth)	Part Execution Time with Offload In Server (in ms)	Part Execution Time with Offload In Mobile 2 (in ms)	Overall Execution Time with Offload (in ms)	Mobile 2 Execution Time Without Offload (in ms)
8	2	4	1	223	2.5
9	2	4	2	198.75	6.75
10	4	5	5	232.5	12.25
11	4	10	20	318	27
12	4	11	30	335	59
13	6	24	65	465.33	91.5
14	6	27	111	519.66	147.25
15	8	105	128	654.25	312.66
16	8	107	420	891	522.25
17	10	311	585	1673.25	1121
18	10	600	883	2186.5	1893
19	12	1205	1857	3532.25	3822.5
20	12	1240	5607	6257.5	7376
21	14	4006	7084	11772.5	19905.3



5. Future Scope

Currently our system is a prototype but we would like to continue our work to make it a complete system for code offloading. We would introduce the below features in our system-

- A network decision maker that check the network condition before offloading and should make a decision that whether to offload or not.
- Server should handle multiple clients at a time.
- System should perform offloading for any recursive algorithm.

6. Conclusion

This paper presents a system prototype that supports dynamic class migration as well as offloading of Tower of Hanoi (TOH) computation for different number of disk from Smartphone to cloud or local server. To use the system, user need to install our android app as well as run java code in an IDE. Benefits of using the system prototype are shown by the result while running the TOH. Result is dependent on Round Trip Delay (RTD) that depend on network health. Result changed according to the specification of smartphone. Result show that by using our system we can achieve more than 50% improvement in performance for higher values of N (no. of disks) while offloading as compared to execution on Smartphone alone. For lower values of N (no. of disks) it is better to not offload the code to the server and execute it on the Smartphone.

7. References

1. Das PK, Shome S, Sarkar AK. APPS: Accelerating Performance and Power Saving in Smartphones using code offload. 6th IEEE International Advance Computing Conference; 2016 Feb.

2. Android.widget.EditText class. Available from: http://developer.android.com/reference/android/widget/EditText.html
3. java.net.URLClassLoader class. https://docs.oracle.com/javase/7/docs/api/java/net/URLClassLoader.html
4. java.lang.reflect.Method class. Available from: https://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Method.html
5. java.lang.reflect.Field class. Available from: https://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Field.html
6. java serialization/deserialization library: Gson. Available from: https://github.com/google/gson
7. Android Studio Apk build process. Available from: http://developer.android.com/sdk/installing/studio-build.html
8. Tower of Hanoi. Available from: https://en.wikipedia.org/wiki/Tower_of_Hanoi
9. Cuervo E, Balasubramanian A, Cho D, Wolman A, Saroju S, Chandra R, Bahl P. Maui: Making Smartphone last linger with code offload. Mobisys '10 Procedings of 8th International Conference on Mobile System, Application and Services; 2010 Jun.
10. Chun BG, Ihm S, Maniatis P, Naik M, Patti A.Clonecloud: Elastic execution between mobile devices and cloud. Eurosys '11 Proceedings of Sixth Conference on Computer System, 2011. p. 301–14.
11. Kemp R, Palmer N, Kielmann T, Bal H. Cuckoo: A computation offloading framework for smartphone. MOBICASE '10 Mobile computing, Applications, and Services 2010.
12. Kosta S, Aucinas A, Hui P, Mortier R, Zhang X. ThinkAir: Dyanamic Resource allocation and parallel execution in the cloud for mobile code offloading. INFOCOM IEEE Proceedings; Orlando, FL. 2012 Mar 25-30. p. 945–53.
13. Khan A, Othman M, Madani S, Khan S. A survey of mobile cloud computing application models. Communication Surveys Tutorials, IEEE. 2013; 16(1):393–413.
14. Zhang Y, Huang G, Liu X, Zhang W, Mei H, Yang S. Refactoring Android Java code for on demand computation offloading. OOPSLA '12 Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications; 2012. p. 233–48.