CHAPTER 1

INTRODUCTION

1.1 AIM

The main aim of this project is to perform adaptive computational offloading from mobile to cloud in order to perform high level computationally intensive application execution in mobile and evaluate the efficiency of this approach in terms of the execution time saved. In this project we attempt to create a middleware for offloading which will partition and offload computation heavy parts of an application onto the cloud. The middleware is the main decision making and communication entity of the project.

1.2 SYNOPSIS

The proposed approach for adaptive offloading has great significance in executing computationally expensive applications in a mobile device running android platform. The approach can be used for offloading parts of an application to high performance cloud computing environment. To implement the offloading approach, we develop an Engine or middleware over which the application to be offloaded runs. The framework implements a decision algorithm and execution algorithm. The execution algorithm uses the decision algorithm to decide whether to offload a particular code or execute locally. The communication between different activities and the discovery of network is done using broadcast receivers and intents. A computation speed relation (ratio) is formulated to increase the accuracy of the cost estimation during each turn.

CHAPTER 2

LITERATURE SURVEY

2.1 GENERAL

This chapter gives the overall description of the reference papers, the techniques and the methodology used through which we can identify the problems of existing methodology. The study helps us to identify the solutions to the problems prevailing in the existing methodology and to increase the overall performance of the system.

2.2 LITERATURE REVIEW

Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications

Mobile phones are set to become the universal interface to online services and cloud computing applications. However, using them for this purpose today is limited to two configurations: applications either run on the phone or run on the server and are remotely accessed by the phone. These two options do not allow for a customized and flexible service interaction, limiting the possibilities for performance optimization as well. In this paper we present a middleware platform that can automatically distribute different layers of an application between the phone and the server, and optimize a variety of objective functions (latency, data transferred, cost, etc.). Our approach builds on existing technology for distributed module management and does not require new infrastructures. In the paper we discuss how to model applications as a consumption graph, and how to process it with a number of novel algorithms to find the optimal distribution of the application modules. The application is then dynamically deployed on the phone in an efficient and transparent manner. We have tested and validated our approach with extensive experiments and with two different applications. The results indicate that the techniques we propose can significantly optimize the performance of cloud applications when used from mobile phones.

Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones

This paper describes PowerBooter, an automated power model construction technique that uses built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption while explicitly controlling the power management and activity states of individual components. It requires no external measurement equipment. We also describe Power Tutor, a component power management and activity state introspection based tool that uses the model generated by PowerBooter for online power estimation. PowerBooter is intended to make it quick and easy for application developers and end users to generate power models for new smartphone variants, which each have different power consumption properties and therefore require different power models. Power Tutor is intended to ease the design and selection of power efficient software for embedded systems. Combined, PowerBooter and Power Tutor have the goal of opening power modeling and analysis for more smartphone variants and their users. Cuckoo: a Computation offloading Framework for Smartphones.

Offloading computation from smartphones to remote cloud resources has recently been rediscovered as a technique to enhance the performance of smartphone applications, while reducing the energy usage.

In this paper we present the first practical implementation of this idea for Android: the Cuckoo framework, which simplifies the development of smartphone applications that benefit from computation offloading and provides a dynamic runtime system that, can, at runtime, decide whether a part of an application will be executed locally or remotely. We evaluate the framework using two real life applications.

MAUI: Making Smartphones Last Longer with Code Offload

This paper presents MAUI, a system that enables fine-grained Energy-aware offload of mobile code to the infrastructure. Previous approaches to these problems either relied heavily on programmer support to partition an application, or they were coarse-grained requiring full process (or full VM) migration. MAUI uses the benefits of a managed code environment to offer the best of both worlds: it supports fine-grained code offload to maximize energy savings with minimal burden on the programmer. MAUI decides at runtime
Which methods should be remotely executed, driven by an optimization engine that achieves the best energy savings possible under the mobile device's current connectivity constrains. In our evaluation, we show that MAUI enables: 1) a resource-intensive face recognition application that consumes an order of magnitude less energy, 2) a latency-sensitive arcade game application that doubles its refresh rate, and 3) a voice-based language translation application
That bypasses the limitations of the smartphone environment
by executing unsupported components remotely.

2.3 SUMMARY

S.no	References	Improvements	Year
1	I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling theCloud: Enabling Mobile Phones as Interfaces to Cloud Applications,"	A GUI framework is to help the user to utilize the resources.	2009
2	E. Cuervo, A. Balasubramanian, D. Cho, "MAUI: Making Smartphones Last Longer with Code Offload,"	MAUI system for offload coding using partitions.	2011
3	R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a Computation Offloading Framework for Smartphones,"	The first practical implementation of offload computing :the Cuckoo framework	2010
4	L. Zhang, B. Tiwana, Z. Qian, Z. Wang, Z. Morley Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behaviour Based Power Model Generation for Smartphones,"	Accurate measurement of efficiency of the system.	2010

Table 2.1

The summary of the Literature Review is given in Table 2.1. The research papers are analysed based on their objectives, algorithm proposed, merits and demerits.

CHAPTER 3

SYSTEM ANALYSIS

3.1 GENERAL

Smartphones have evolved over the years, but they can never match the performance of computers.In the proposed system, the computations to be performed by mobiles are offloaded to PCs through cloud computing by Adaptive offloading .Applications are developed by using the standard Android development pattern. The middleware does the heavy lifting of adaptive application partitioning, resource monitoring and computation offloading. The evaluation shows that applications, which involve costly computations, can benefit from offloading with around 95% energy savings and significant performance gains compared to local execution only.

3.2 EXISTING SYSTEM

High computational application which cannot be effectively and solemnly run on the mobile within reasonable time and power are offloaded to the cloud for performing the computation and the result is pushed back into the mobile device which is displayed on the mobile device.

Latest updates in Android operating system provide offloading capabilities which are limited to certain applications and processing domains.

3.3 ISSUES IN EXISTING SYSTEM

The offloading of the computation is not adaptive, the whole of the computation if offloaded for that specific application, which induces unnecessary overhead for the computation which are much more efficiently run on the device .This type of offloading would lead to the requirement of persistent internet connection for the application to work, thus may lead to draining of battery rather than saving it.

These systems either offload the application as a whole or none. Consequently these systems only work for a specific domain of applications and reduce the effectiveness of cloud computing systems.

This approach best works with limited communication involved which is contrasting to the existing systems approach where the whole of the application instruction is offloaded.

3.4 PROPOSED SYSTEM

In the proposed system a framework or offloading engine is developed which will adaptively offload the parts of an application which are computationally intensive to the cloud.

It is adaptive in terms of deciding whether to offload the particular code fragment to the cloud or it is effective if executed locally. The framework performs the decision and then carries out the steps necessary to execute the particular function.

The proposed system consists of an engine which takes in the application context of an android application and then determines the functions that need to be offloaded and executes them.

3.5 ADVANTAGES OF PROPOSED SYSTEM

The proposed system considerably reduces the execution times of complex algorithms by offloading them to the cloud thus saving computing cycles of the mobile platform.

It also reduces the battery power usage of the mobile device by making the processor idle during offloaded execution.

This system enables high computation applications to be developed for mobile devices which are usually restricted by the batter power of the mobile devices.

Since this system is adaptive, it reduces the redundant instruction execution and uses only whatever are needed to execute.

3.6 SYSTEM REQUIREMENTS

The requirements specification is a technical specification of requirements for the software products. It is the first step in the requirements analysis process it lists the requirements of a particular software system including functional, performance and security requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective. The purpose of software requirements specification is to provide a detailed overview of the software project, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionality.

3.6.1 HARDWARE REQUIREMENTS

Hard Disk	:	40GB and Above
RAM	:	512MB and Above
Processor	:	Pentium III and Above
Mobile device	:	Dual core preferred

3.6.2 SOFTWARE REQUIREMENTS

Java 1.6 or higher is required.

Eclipse Indigo (Version 3.7.2) or higher is required.

ADT plugin for eclipse.

Amazon ec2 instance.

3.7 SUMMARY

The system analysis chapter describes the existing methodology with its issues and the proposed methodology with its advantages along with the hardware and software specifications.

CHAPTER 4

SYSTEM DESIGN

4.1 GENERAL

System design is the process or art of defining the architecture, components, modules, interfaces and data for a system to satisfy specified requirements. The design phase of a project is to plan out a system that meets the requirements in the analysis phase. Based on the analysis, the modules are identified and designed for identifying the project. The chapter provides the module description along with the techniques used in each module.

4.2 MODULES

4.2.1 USER MODULE

The user module is a generic application developed using the standard android development procedure; it is run independent of the framework and other applications. The user module has no visibility of its lower layers which includes the framework and the cloud and runs as it would run in a normal environment. The user module can be any standalone application which is computationally intensive.In our project we use sample algorithms which are computationally intensive and which are used for evaluating the results produced from adaptive offloading of the framework.

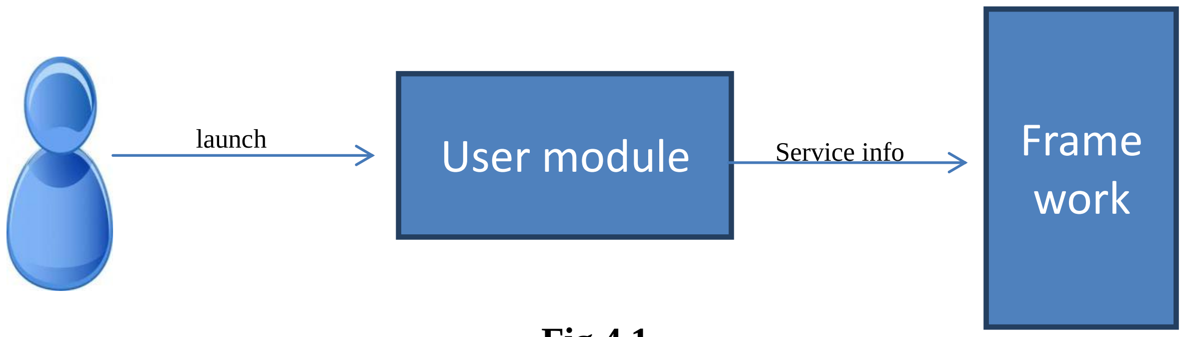


Fig 4.1

The user module acts as the interface between the user and the engine. It provides functionalities to set input parameters and offloading parameters and select the appropriate algorithm with which to evaluate the offloading approach.

Once an algorithm is executed, it transfers control to the middleware which is responsible for decision making and execution.

After the execution and finding of the result the control is given back to the user module which displays the results and metrics of the execution.

The user module consists of a single activity with 3 different layouts , each layout to display different stages of the execution of the application.

Initially a GUI with editable text fields for input parameters and force offloading flags along with multiple clickable buttons for each of the algorithms used for evaluating the approach.

Apart from these, another button to check the engine parameters , which on clicked will show the different computation speed relations of the algorithms, then the RTT(Round Trip Time), the bandwidth of the mobile device , if the connection is available , if yes then what type of connection it is and then finally if an offloading server is available.

On executing a particular algorithm the details of the algorithms are shown by loading a separate layout. It first shows the estimated android runtime, server runtime and offloading time. The second part displays the real values on completion of the project which include the overhead imposed due to the decision making, offloading time and the server or android execution time and whether the code was offloaded or not. It also shows the result of the operation if any.

4.2.2 MIDDLEWARE

The middleware module acts as the interface between the android platform and the user module. The middleware is a framework which functions as a service on top of the android platform. The framework is divided into three sub modules:-

- Service info module.
- Decision module.
- Communication module.

The service info module gathers the information about the user module (application) which includes code size, memory, execution parameters, algorithm costs, input parameters and offloading parameters. The service info about an application is obtained by receiving the application context from the corresponding application. The application context contains all the necessary information regarding the application. The MACS library contains the cost functions of the different algorithms used. The service info module retrieves the values from the library for the estimation. The input parameters and the algorithm name are sent to the module by sending the application context. The application context can also be additionally used for determining other execution factors on a on-demand basis.

The decision module decides whether to offload the execution to the cloud or execute it locally based on parameters using formulae. The decision module is the heart of the engine which decides which code to offload. It makes an estimate of the android execution times, server execution times and offloading times for the particular algorithm and decides to offload only when a certain criterion is metWhen an algorithm is executed by the user, the Engine first makes a call to the decision module. The decision module estimates the android runtime, the server runtime and the offloading time and decides to offload only when the estimated android runtime is greater than the sum of the offloading time and server runtime. The estimation is carried out only if a connection exists and a server is available. If no server exists, estimation is redundant since it will be executed in the android runtime whatever the cost.

Based on the result from the decision module, the execution may take place within the android locally or offloaded to the cloud. To improve the accuracy of the estimation of the runtimes a special mapping is developed which contains the computation speed relations for the algorithms which is updated every time that particular algorithm is offloaded. It involves finding the average of the real runtimes of the present and previous averages. Initially a sample algorithm is run and the CSR is found and is assumed for all the algorithms. This CSR is the ratio of the android runtime to the server runtime. This essentially improves the accuracy of the estimation every time the code is offloaded. The CSR is stored in a shared preferences file for the algorithms and is kept updated consequently.

Suppose the computation requires C instructions. Let S and M be the speeds, in instructions per second, of the cloud server and the mobile system, respectively. The same task thus takes C/S seconds on the server and C/M seconds on the mobile system. If the server and mobile system exchange D byte of data and B is the network bandwidth, it takes D/B seconds to transmit and receive data. The mobile system consumes, in watts, Pc for computing, Pi while being idle, and Ptr for sending and receiving data. (Transmission power is generally higher than reception power, but for the purpose of this analysis, they are identical.)

If the mobile system performs the computation, the energy consumption is $Pc \times (C/M)$. If the server performs the computation, the energy consumption is $[Pi \times (C/S)] + [Ptr \times (D/B)]$. The amount of energy saved is

$Pc \times C/M – Pi \times C/S – Ptr \times D/B$. (1)

Suppose the server is F times faster that is, $S = F \times M$. We can rewrite the formula as $C/M \times (Pc - Pi/F) - Ptr \times D/B$. (2)

Energy is saved when this formula produces a positive number. The formula is positive if D/B is sufficiently small compared with C/M and F is sufficiently large. The values of M, Pi, Pc, and $Pth$are parameters specific to the mobile system. For example, an HP iPAQ PDA with a 400-MHz ($M = 400$) Intel XScale processor has the following values: $Pc \approx 0.9$ W, $Pi \approx 0.3$ W, and $Ptr \approx 1.3$ W.

If we use a four-core server, with a clock speed of 3.2 GHz, the server speedup F may be given by $(S/M) \approx [(3.2 \times 1{,}024 \times 4 \times X)/400]$, where X is the speedup due to additional memory, more aggressive pipelining, and so forth. If we assume $X = 5$, we obtain the value of $F \approx 160$.

The value of F can increase even more with cloud computing if the application is parallelizable, since we can offload computation to multiple servers. If we assume that $F = 160$, Equation 2 becomes

$C/400 \times 0.9 - 0.316o - 1.3 \times D/B = (0.00225 \times C) - 1.3 \times D/B$. (3)

For offloading to break even, we equate Equation 3 to zero and obtain

$Bo/577.77 \times DC$, (4)

Where Bo is the minimum bandwidth required for offloading to save energy, determined by the ratio of (D/C). If (D/C) is low, then offloading can save energy. Thus, as Figure 1 shows, offloading are beneficial when large amounts of computation C are needed with relatively small amounts of communication D.

As service-based implementation is adopted, for each service we can profile following metadata:

Type: whether can be offloaded or not

Memory cost: the memory consumption of the service on the mobile device

Code size: size of compiled code of the service

Dependency information on other services, for each
Related module, we collect following:
O transfer size: amount of data to be transferred
O send size: amount of data to be sent
O receive size: amount of data to be received
Metadata is obtained by monitoring the application execution and environment.

The communication module is responsible for transferring the data back and forth the cloud and the framework. It is responsible for the interaction between android platforms. The communication module uses the post method of http requests to send forth input parameters to the cloud server and then get back the result produced by the cloud server. The communication module is responsible for giving back the results to the main layout of the user module.

The communication module communicates with the cloud server with the use of Http Post request where the algorithms and parameters are sent as List objects. The communication module keeps listening to the network channel for a connection and when a connection becomes available it updates the connection type and server available fields. This monitoring is done with the help of broadcast receivers. Broadcast receivers are registered for a particular event. In this case a network connection and when and if a connection is available the details are updated. A separate thread is maintained for monitoring the connection. The bandwidth of the connection is found out by sending a file of a fixed size (200Kb) multiple times and finding the average of the time taken to send the file to the server. The communication module also keeps the ping measure of the network updated. It is an essential component of the engine because the offloading time plays a crucial part in the decision making process.

After the parameters are sent to the cloud the communication module keeps listening for the result from the cloud in the background as an Async Task.

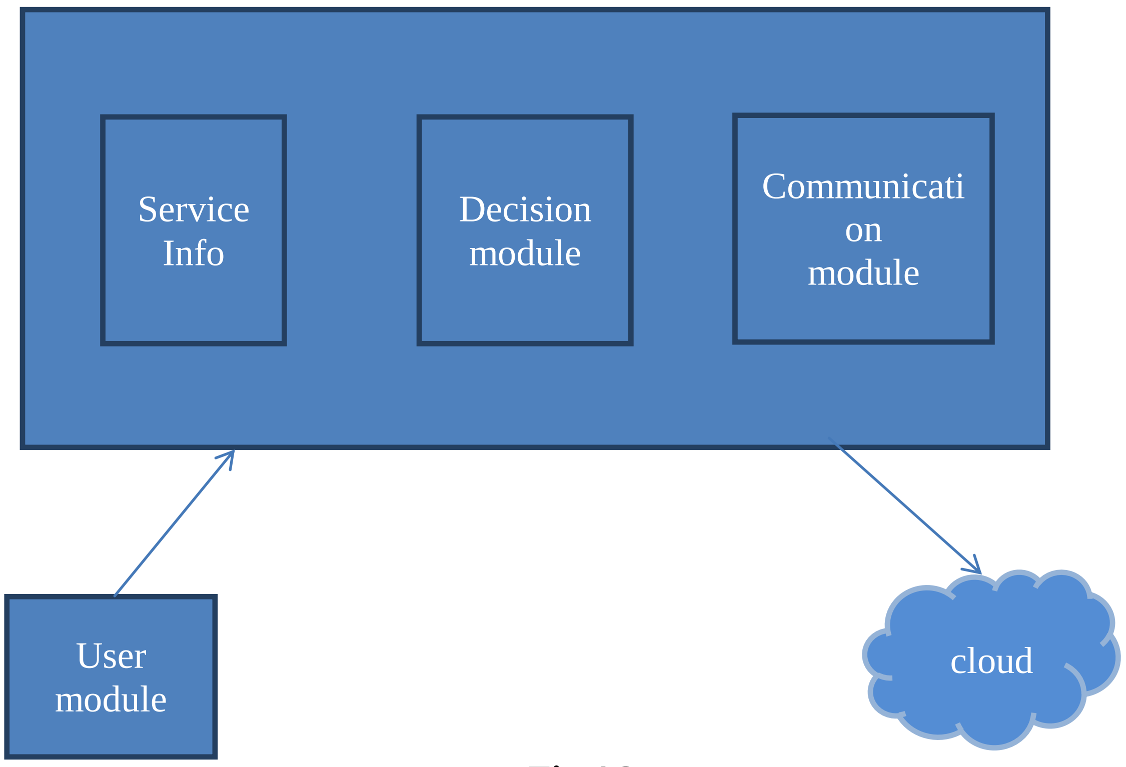


Fig 4.2

4.2.3 SERVER

The server is an Amazon EC2 Cloud instance which is a c3.large instance providing 2.8 GHz 10 core computing power. The server gets the input parameters and then executes the particular algorithm which has been offloaded.

It is several times faster than the mobile device and hence reduces the computation time and consequently saving power.

The server side code contains a servlet which receives post requests from the communication module of the engine. It evaluates the parameters sent and executes the algorithm by retrieving the needed code from the MACS library.

After execution, it sends back the result and execution time as an xml response message back to the engine. The xml message is parsed using document factory at the client side.

The EC2 instances can also be changed as per the processing demands of the vendor. Several instance types like on-demand instances, spot instances and reserved instances are available for purchase.

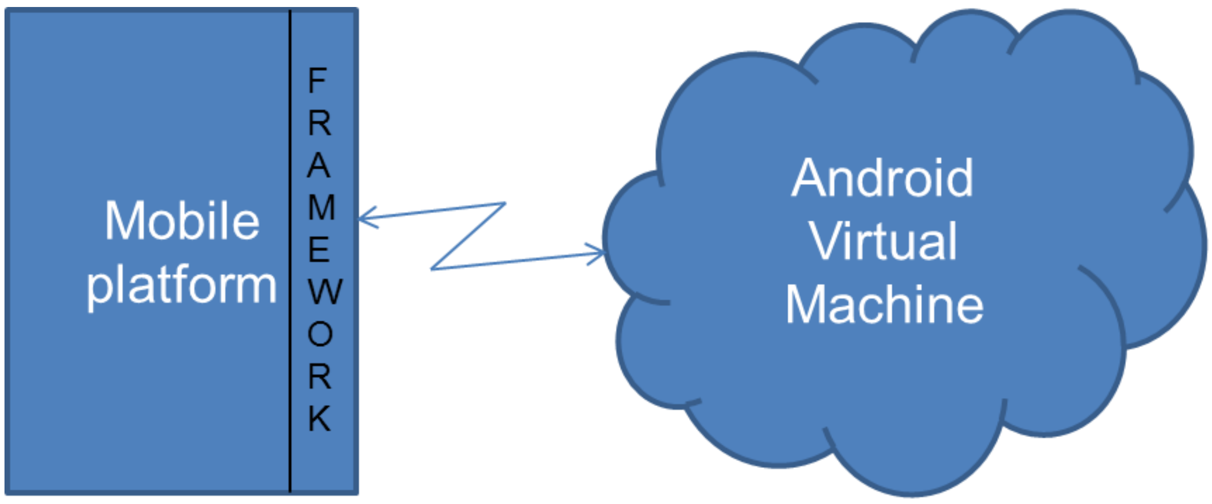


Fig 4.3

4.3 ARCHITECTURE DIAGRAM

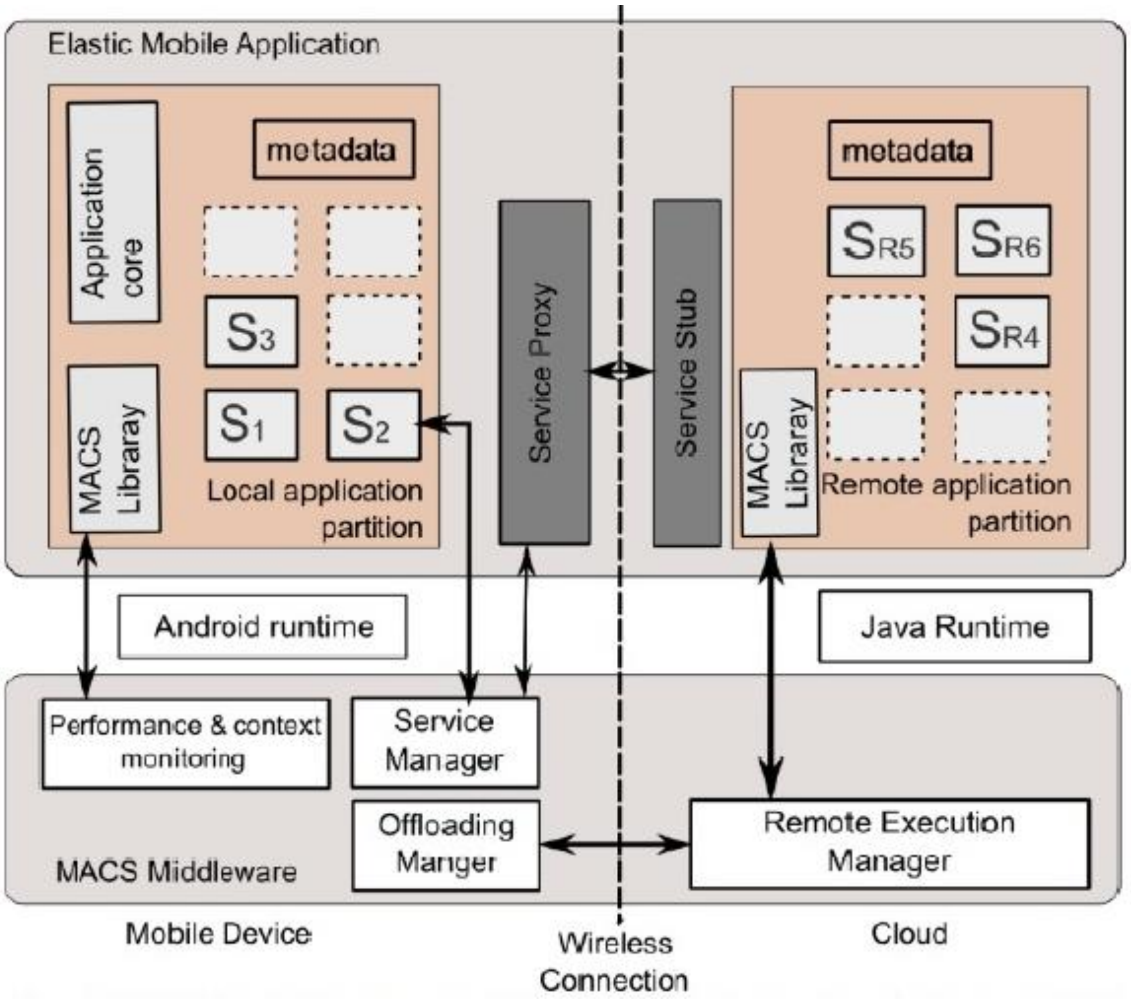


Fig 4.4

The architecture diagram in fig 4.4 shows the complete layout of the offloading system. In the mobile application side, the application is developed according to the standard android development framework with the different computation parts divided into different methods. Each of these methods constitute an offload able component. The front end of the application is designed on a single activity with three layouts. The metadata contains the information about the application which may include code size, parameter lengths, the application context etc., S1, S2, S3 are the individual offload able components which are coded as different methods. The MACS library contains the different codes of the algorithms used and cloud communication modules.

The performance and context monitoring part of the middleware does the job of estimating the android runtime, offloading time and server runtime which will be used by the offloading manager for decision making. The application context is passed to the middleware which is used for offloading and gathering information about the application.

The Offloading manager is responsible for decision making based on the estimated android runtime, estimated offloading time and estimated server runtime. It is also responsible for the communication between the application and the cloud. It sends the input parameters and the algorithm to be offloaded to the cloud.

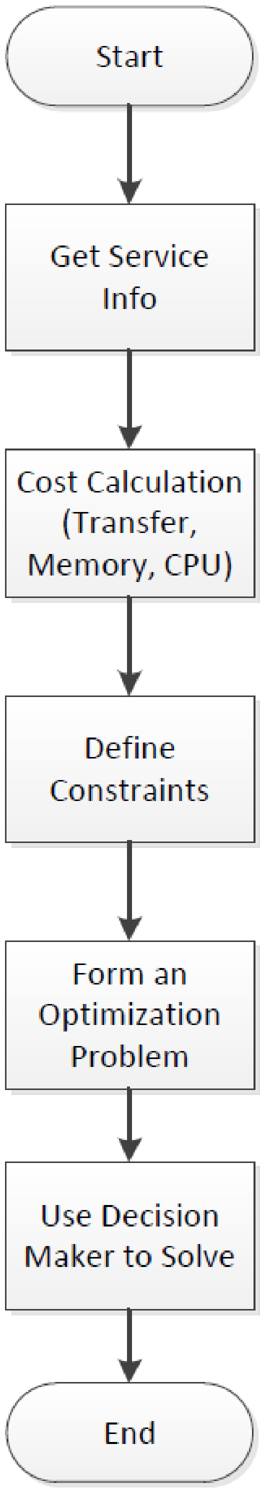
The communication between the mobile application and the cloud execution manager is done over the Internet which maybe a 3G/2G network.

The Remote execution manager is responsible for the communication between the android Engine (middleware) and the cloud execution. It receives the input parameters and the algorithm to be offloaded and transfers it to the java runtime for execution.

The cloud side also contains the code for the different potentially offload able algorithms as different methods stored in a MACS library. It also maintains a cost estimate of all the algorithms.

4.4 FLOW CHART

4.4.1 ALLOCATION DETERMINATION



Start
Get Service Info
Cost Calculation (Transfer, Memory, CPU)
Define Constraints
Form an Optimization Problem
Use Decision Maker to Solve
End

Fig 4.5

This flowchart depicts the overall working of all the three modules. Initially the service info of the application is got through the application context. The cost estimation is then done by the engine. The constraints involve the bandwidth and the connection and the server availability. Considering these factors an optimization problem is formulated which is used by the decision module to make the decision whether to offload or not.

4.4.2 PROCESS REGISTRATION

The following flowchart depicts the functioning of the Engine. The service information is transferred to the engine. The decision module of the engine first decides whether the algorithm can be run locally, if not it decides whether the server is available for the client for offloading. If offloading is decided, the CSR is calculated and registered in the database. If it is the first time it is being registered the CSR is calculated anew and stored. If it has already been registered then the CSR is just retrieved from the shared preferences file stored in the mobile device.

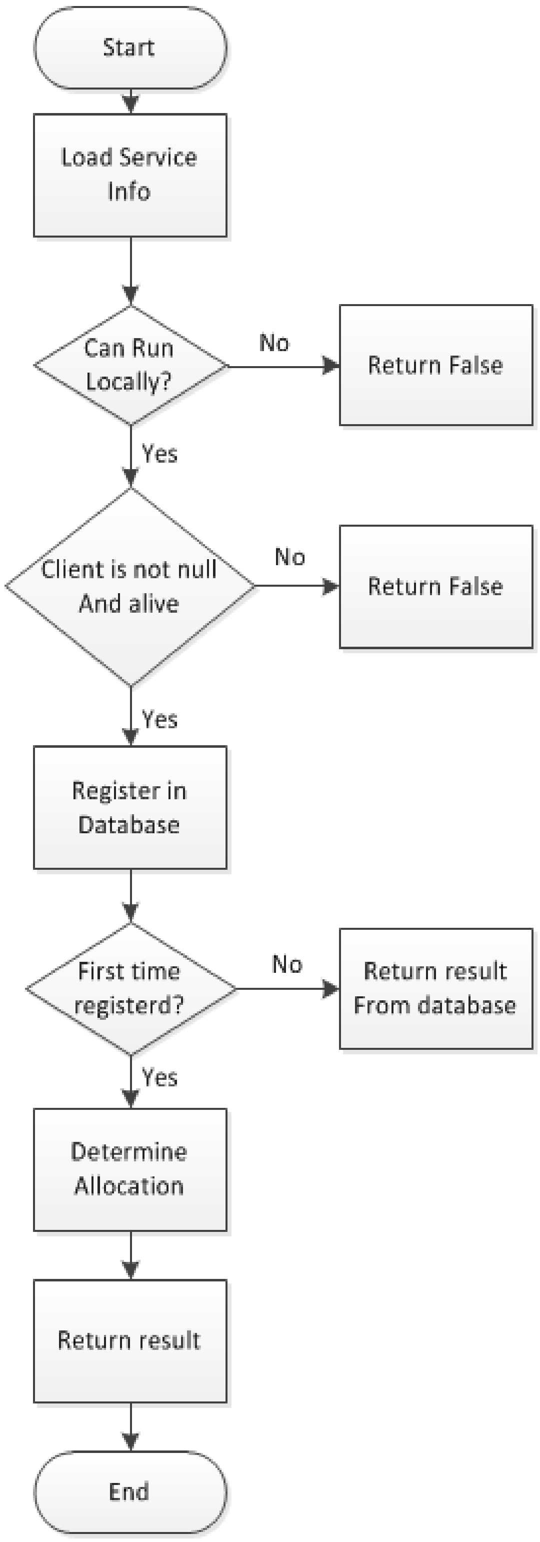
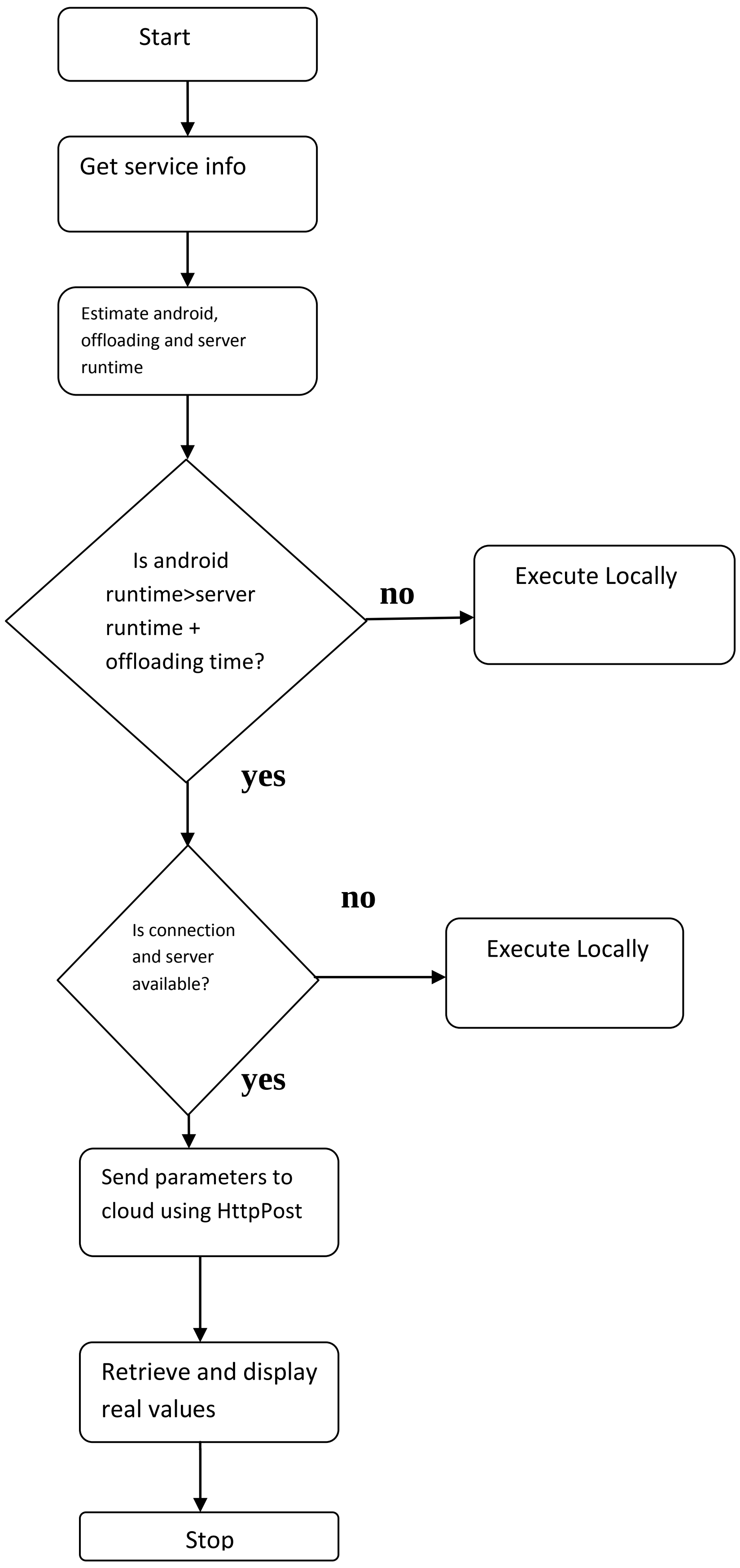

Start
Load Service Info
Can Run Locally?
No
Return False
Yes
Client is not null And alive
No
Return False
Yes
Register in Database
First time registerd?
No
Return result From database
Yes
Determine Allocation
Return result
End

Fig 4.6

4.4.3 OFFLOADER WORKING

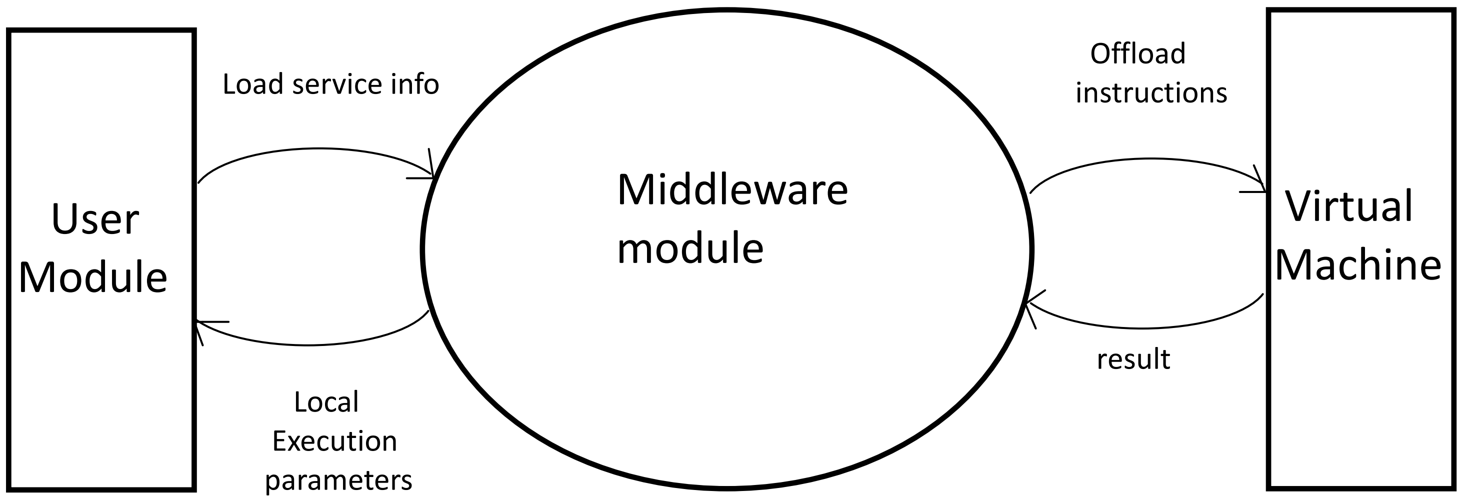


Start
Get service info
Estimate android, offloading and server runtime
Is android runtime>server runtime + offloading time?
no
Execute Locally
yes
Is connection and server available?
no
Execute Locally
yes
Send parameters to cloud using HttpPost
Retrieve and display real values
Stop

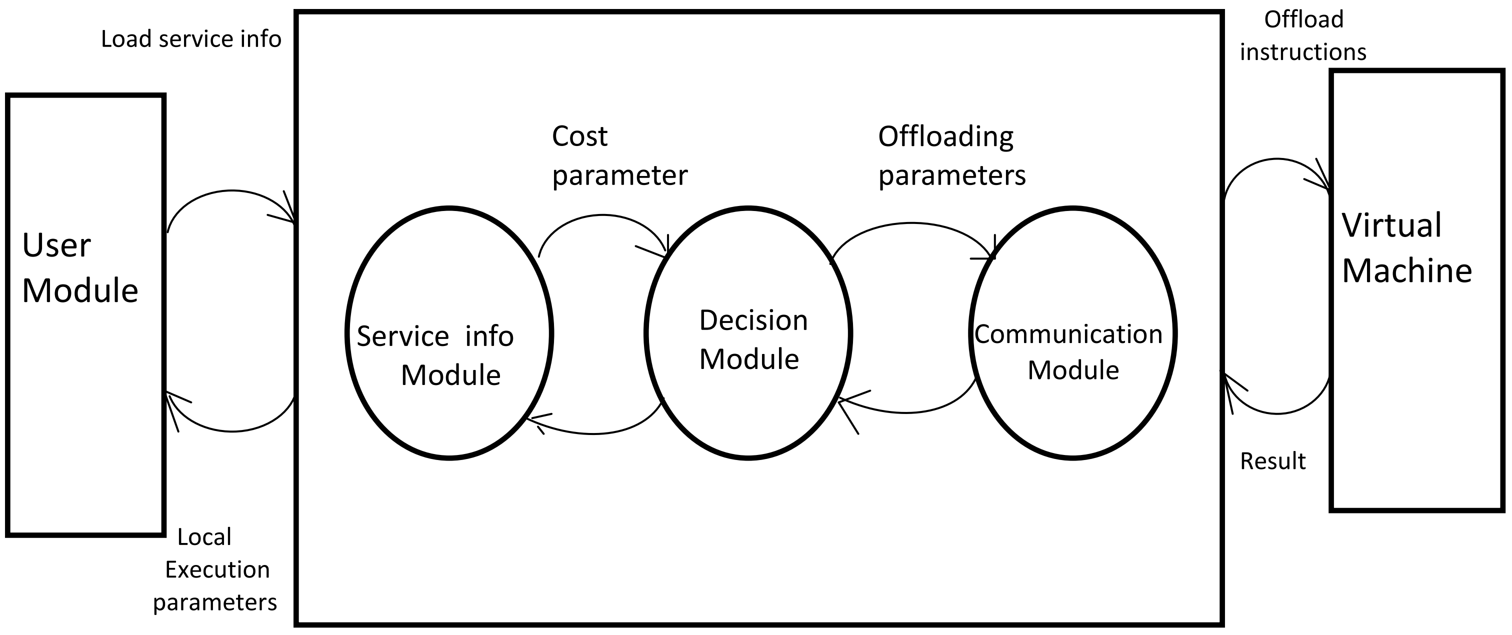
Fig 4.7

4.5 DATA FLOW DIAGRAM

LEVEL 0 DFD

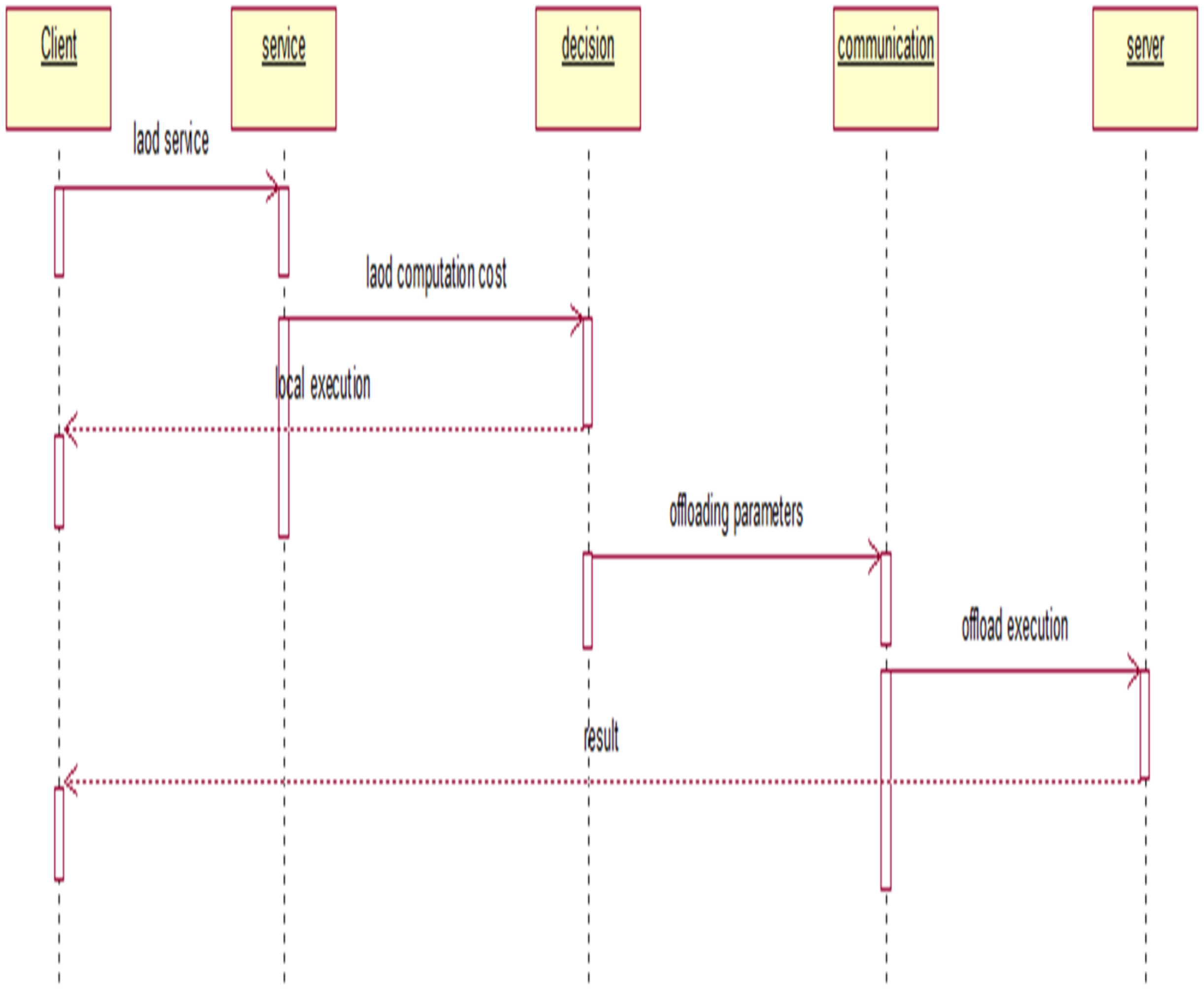


LEVEL 1 DFD

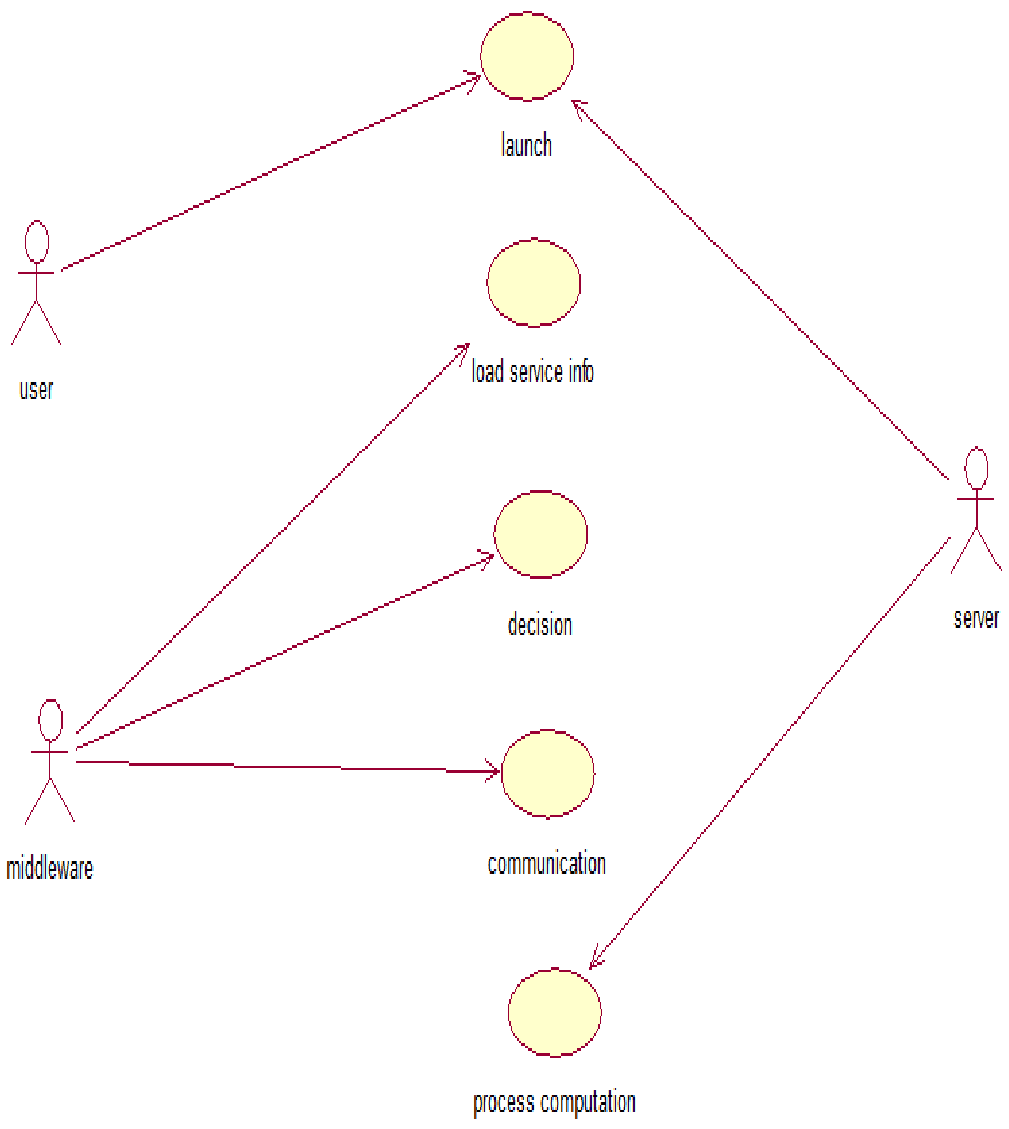


4.6 UML DIAGRAMS

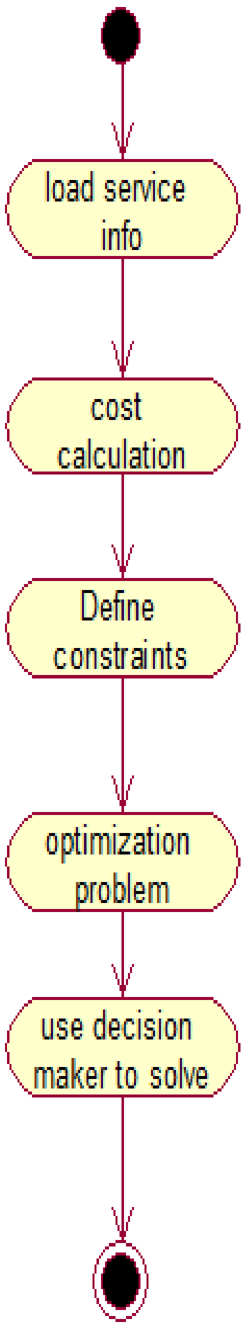
4.6.1 SEQUENCE DIAGRAM



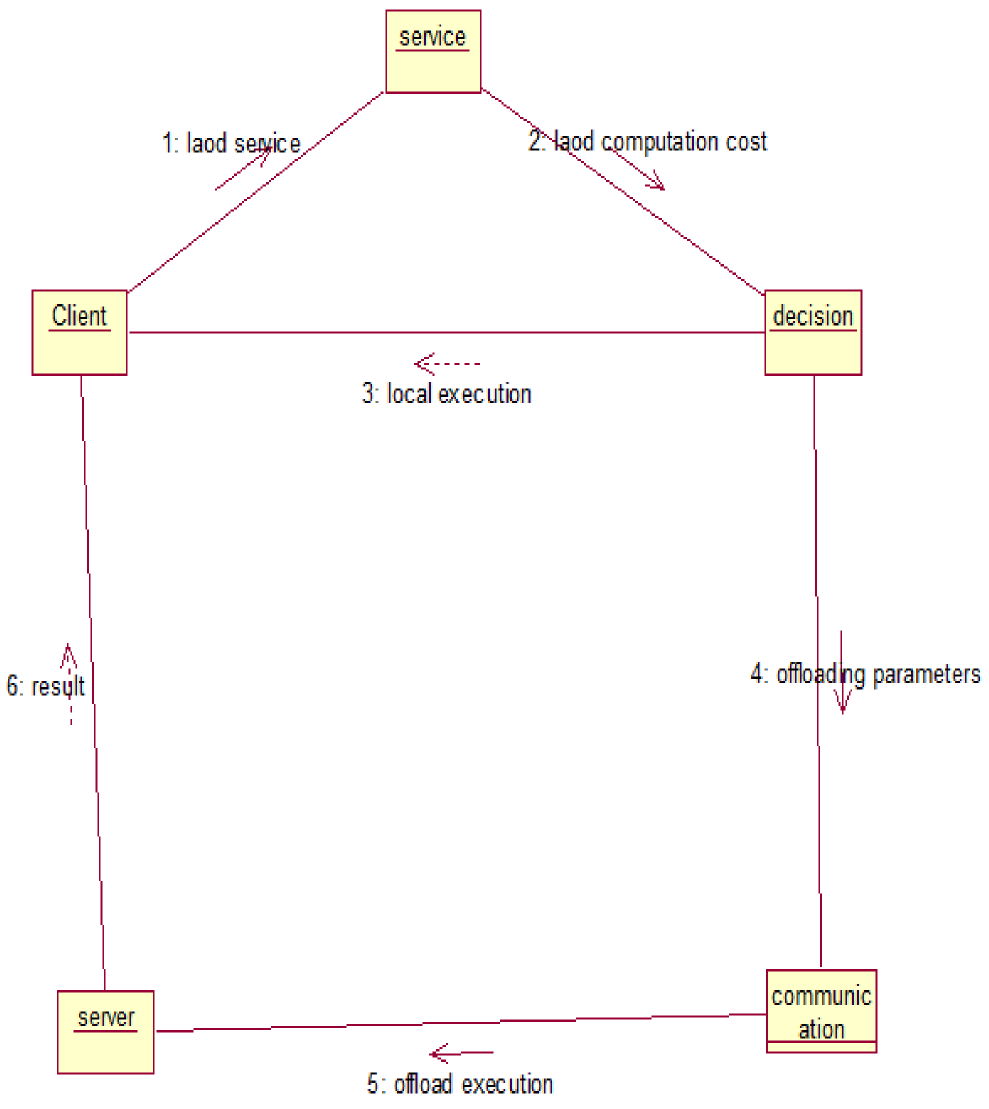
4.6.2 USE CASE DIAGRAM



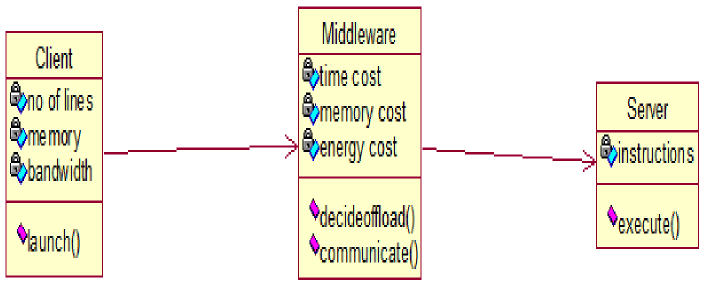
4.6.3 ACTIVITY DIAGRAM



4.6.4 COLLABORATION DIAGRAM



4.6.5 CLASS DIAGRAM



4.8 SUMMARY

This chapter deals with the design of the proposed system and its architecture diagram. The techniques used are explained and the UML diagrams are constructed.

CHAPTER 5
IMPLEMENTATION AND RESULT

5.1 GENERAL

Three main concepts of the android development technique is used in the development of the engine and the application.

Activity

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with windowIsFloating set) or embedded inside of another activity (using ActivityGroup). There are two methods almost all subclasses of Activity will implement:

onCreate()

onCreate(Bundle) is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using findViewById(int) to retrieve the widgets in that UI that you need to interact with programmatically.

onPause()

onPause() is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the ContentProvider holding the data).

Intents

An intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background

Service. An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

Broadcast Receivers

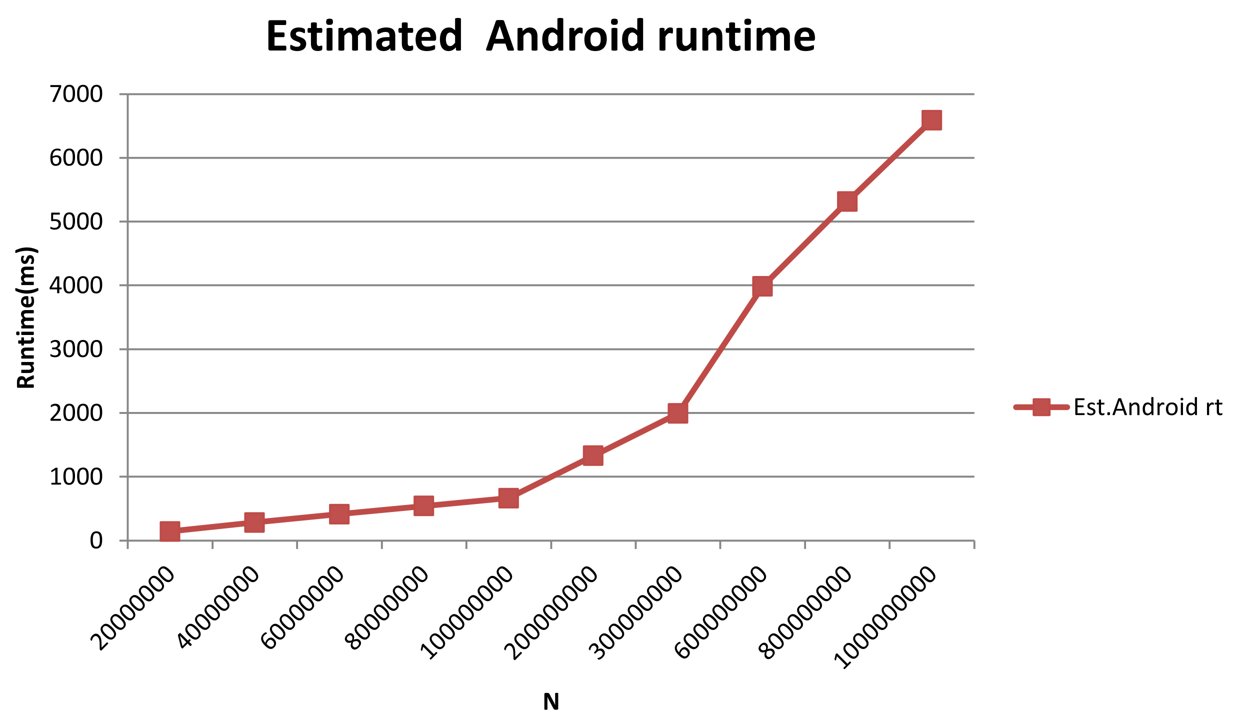
A broadcast receiver (short receiver) is an Android component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens.
For example, applications can register for the ACTION_BOOT_COMPLETED system event which is fired once the Android system has completed the boot process. A receiver can be registered via the AndroidManifest.xml file.
Alternatively to this static registration, you can also register a receiver dynamically via the Context.registerReceiver() method. The implementing class for a receiver extends the BroadcastReceiver class. If the event for which the broadcast receiver has registered happens, the onReceive() method of the receiver is called by the Android system.

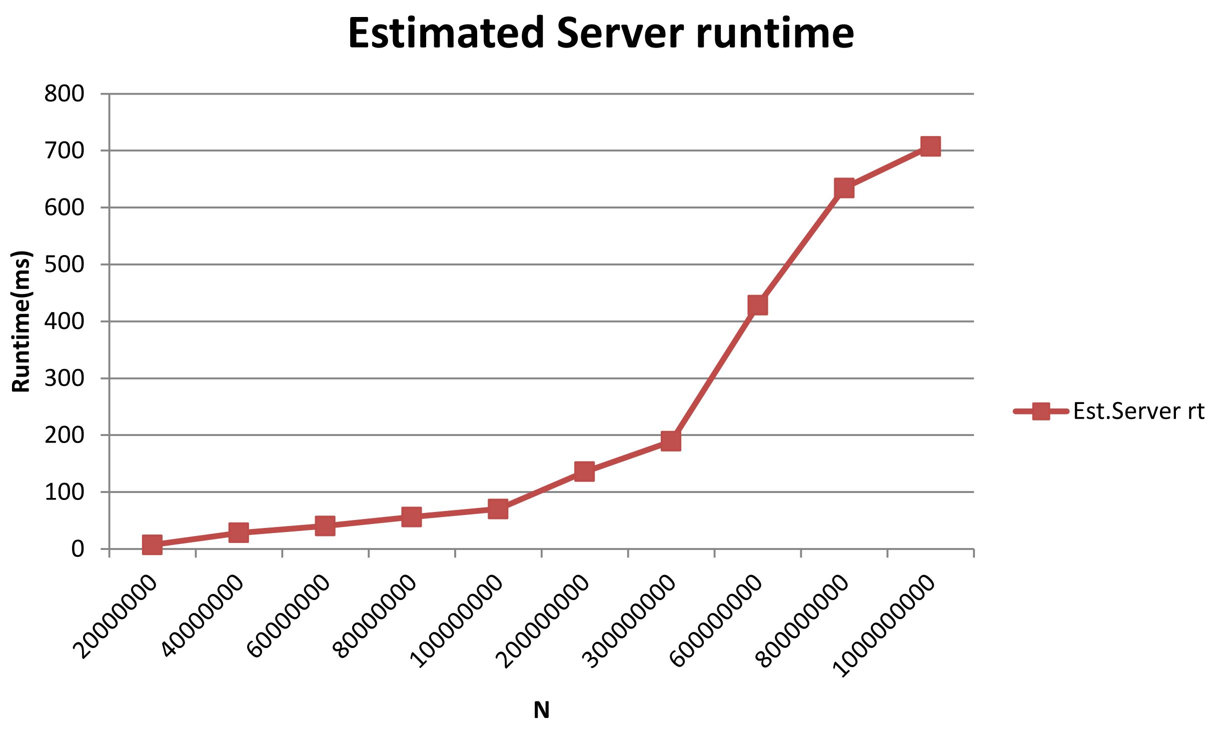
The Android API 17 (Android 4.2 Jellybean) is used as the target system for the development of the application. It offers several new features like the use of HttpClient package which enables communication with the cloud server.

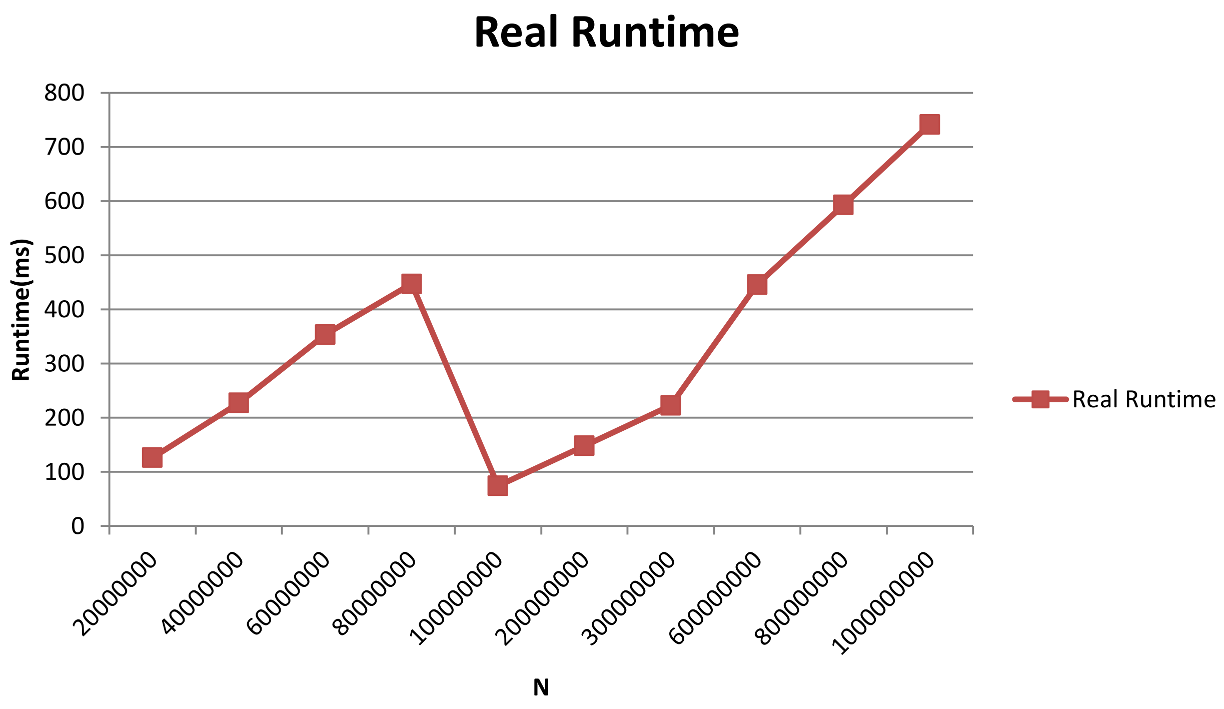
5.2 RESULT

5.2.1 DO SOME LOOPS

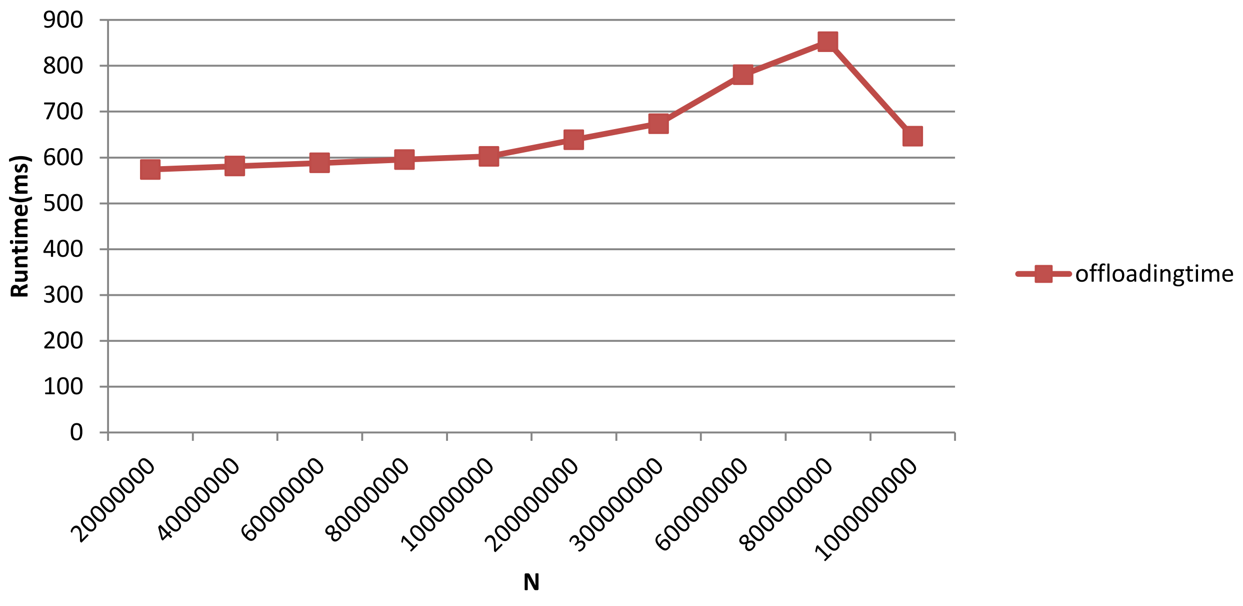
n	Est.Android rt(ms)	Est.Server rt(ms)	Real Runtime(ms)	Offloadingtime (ms)	% saved(time) (ms)
20000000	143.13	7.14	126.4	573.75	11.68867463
40000000	281.48	28.3	227.63	580.89	19.13102174
60000000	412	40.42	353.54	588.03	14.18932039
80000000	540	56.07	447.143	595.18	17.19574074
100000000	664	70.24	74.23	602.32	88.82078313
200000000	1328	135.64	148.27	638.66	88.83509036
300000000	1992	189.14	222.75	673.76	88.81777108
600000000	3985	428.38	445.89	780.45	88.81079046
800000000	5314	634.33	592.84	852.33	88.84380881
1000000000	6590	707.45	741.56	645.92	88.74719272

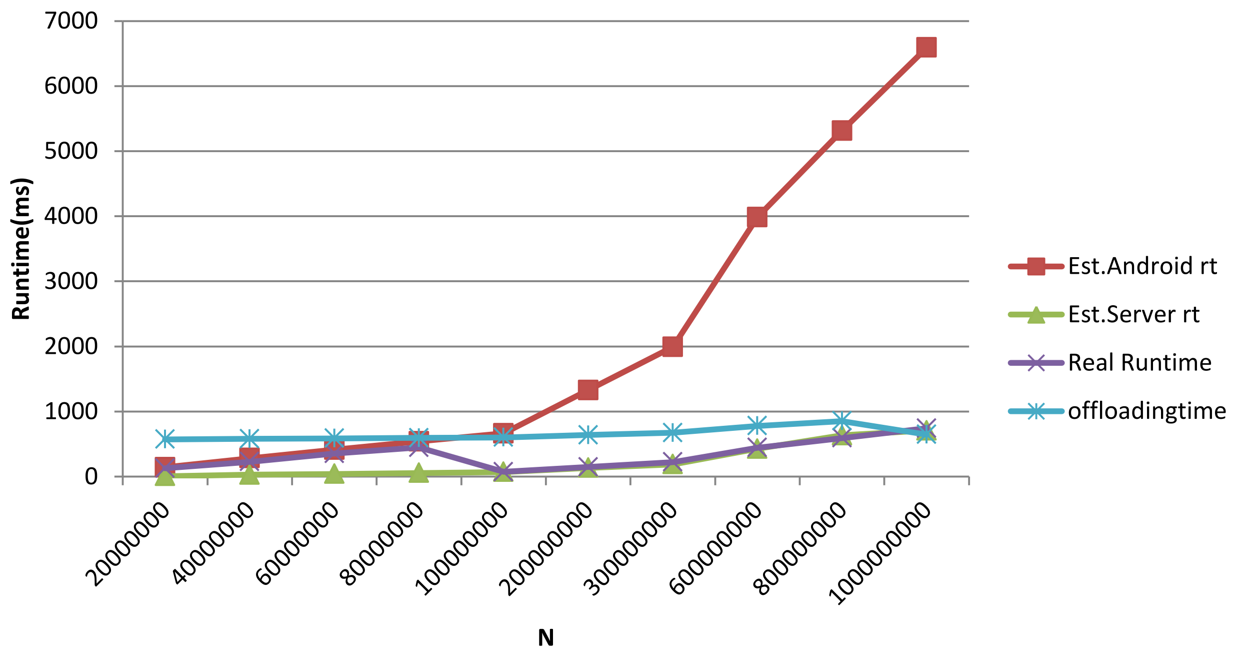



Estimated Server runtime
800
700
600
500
400
300
200
100
0
Runtime(ms)
20000000
40000000
60000000
80000000
100000000
200000000
300000000
600000000
800000000
1000000000
N
Est.Server rt


Real Runtime
800
700
600
500
400
300
200
100
0
Runtime(ms)
20000000
40000000
60000000
80000000
100000000
200000000
300000000
600000000
800000000
1000000000
N
Real Runtime

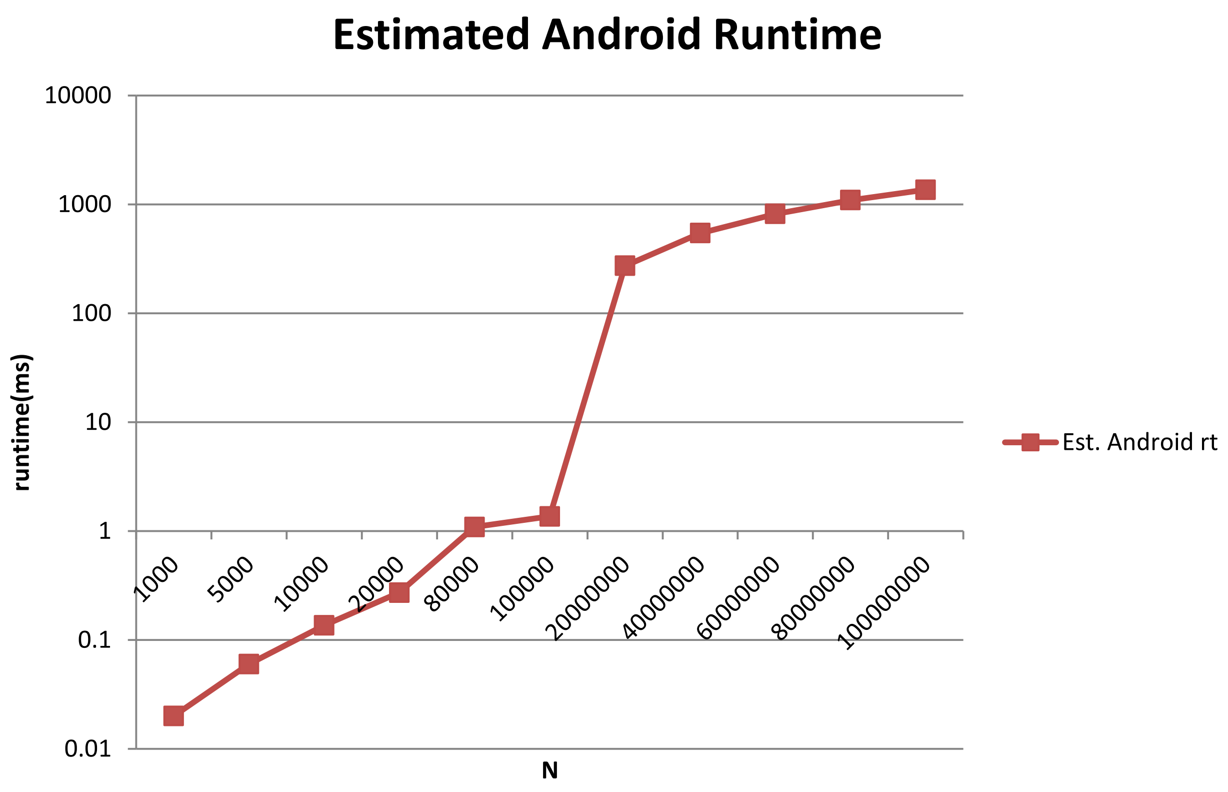
Offloading Time

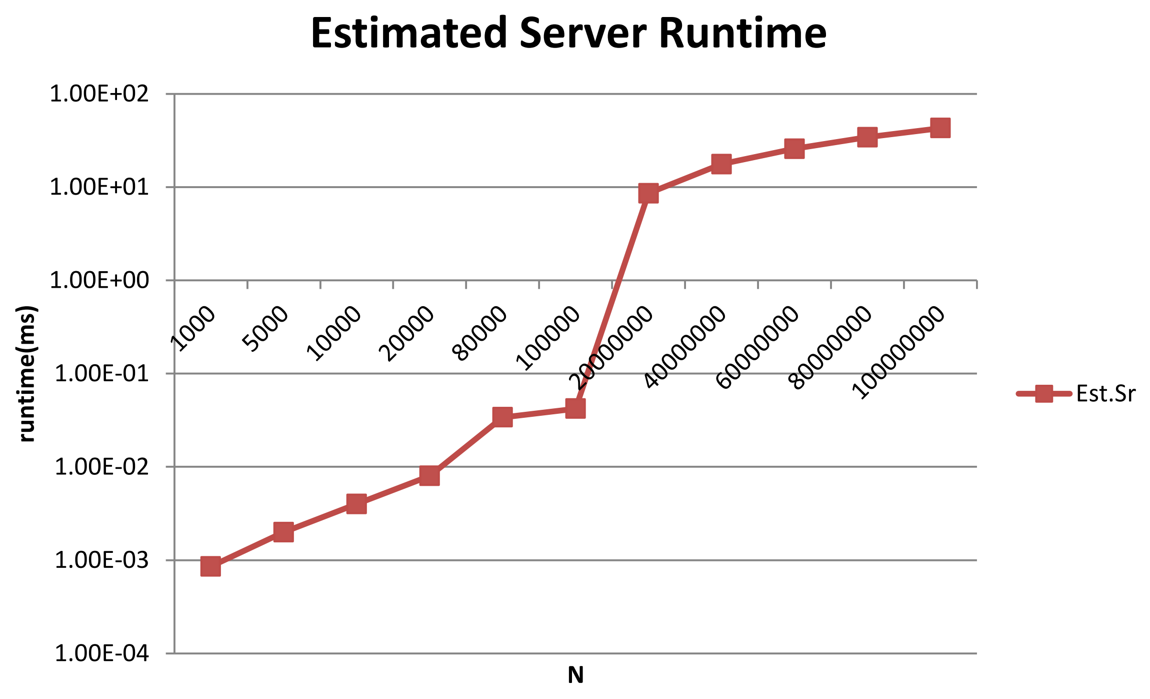




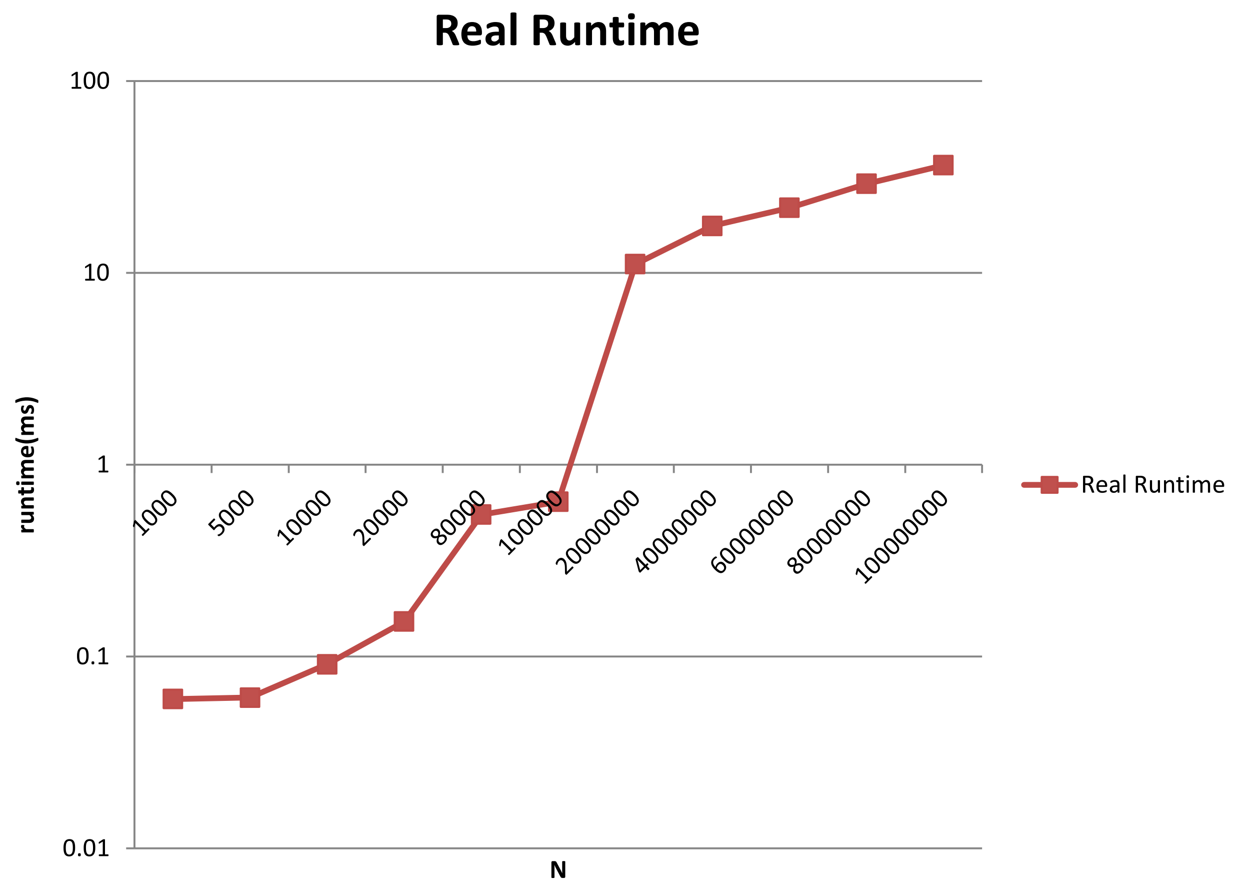
5.2.2 FIBONACCI ITERATIVE

n	Est. Android rt(ms)	Est.Server rt (ms)	Real runtime(ms)	Offloading time(ms)	% saved(time) (ms)
1000	0.02	8.57E-04	0.06	262.7	-200
5000	0.06	0.002	0.061	262.7	-1.666666667
10000	0.136	0.004	0.091	262.7	33.08823529
20000	0.272	0.008	0.152	262.7	44.11764706
80000	1.089	0.034	0.549	271.32	49.58677686
100000	1.36	0.042	0.64	279.6	52.94117647
20000000	272	8.57	11.08	288.533	95.92647059
40000000	544	17.63	17.57	297.1	96.77022059
60000000	817	25.71	21.83	305.67	97.32802938
80000000	1089.84	34.285	29.1	315.66	97.32988329
100000000	1362	42.85	36.358	345.39	97.33054332

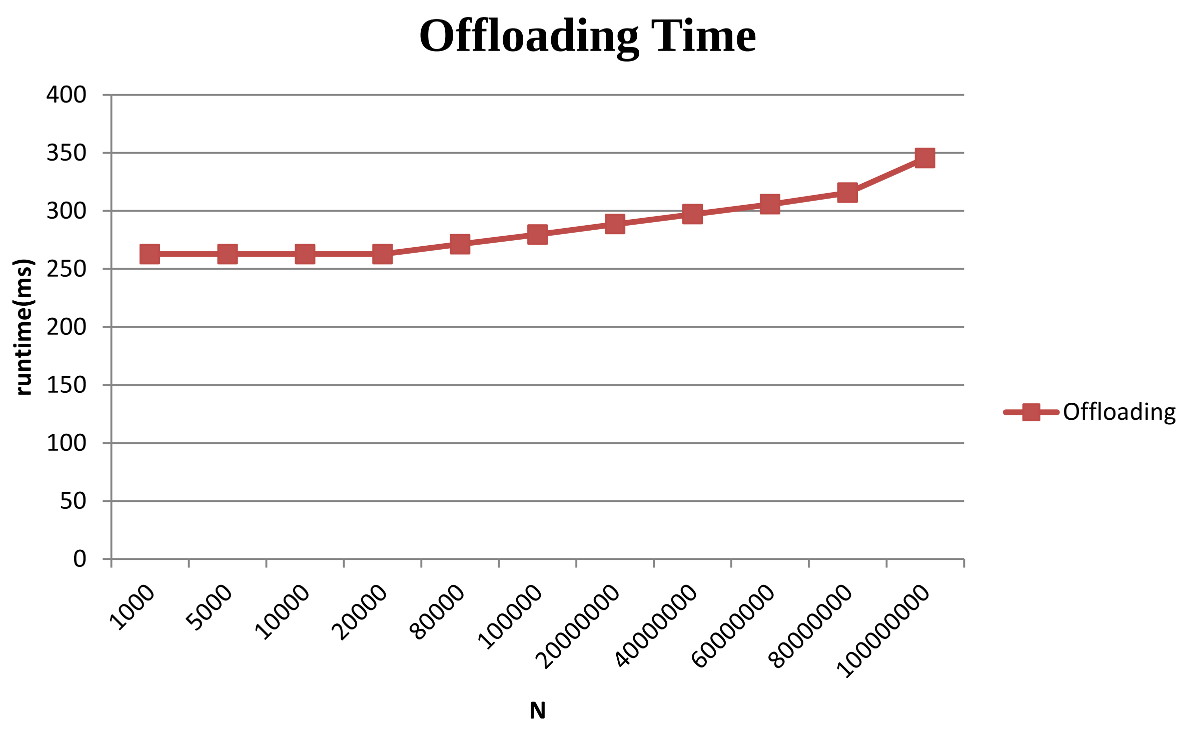




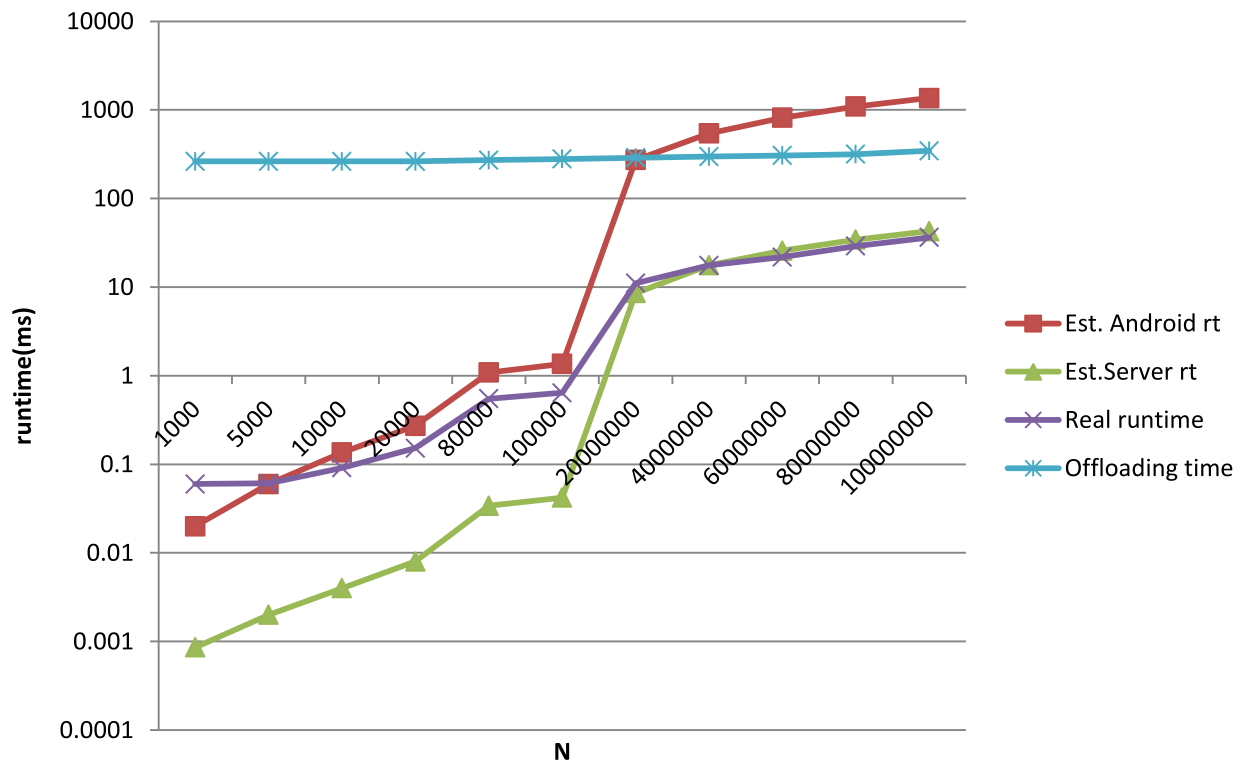
Estimated Server Runtime
1.00E+02
1.00E+01
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
runtime(ms)
1000
5000
10000
20000
80000
100000
20000000
40000000
60000000
80000000
100000000
N
Est.Sr



Real Runtime
100
10
1
0.1
0.01
runtime(ms)
1000
5000
10000
20000
80000
100000
20000000
40000000
60000000
80000000
100000000
N
Real Runtime



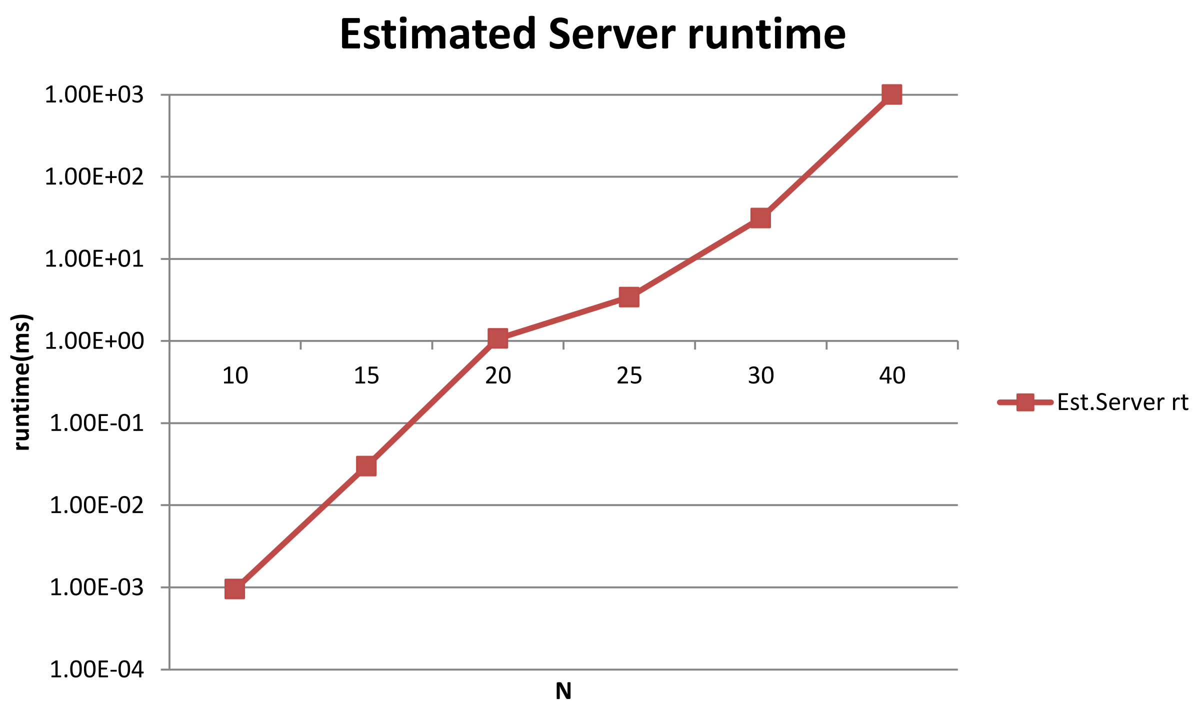
Offloading Time
400
350
300
250
200
150
100
50
0
runtime(ms)
1000
5000
10000
20000
80000
100000
20000000
40000000
60000000
80000000
100000000
N
Offloading

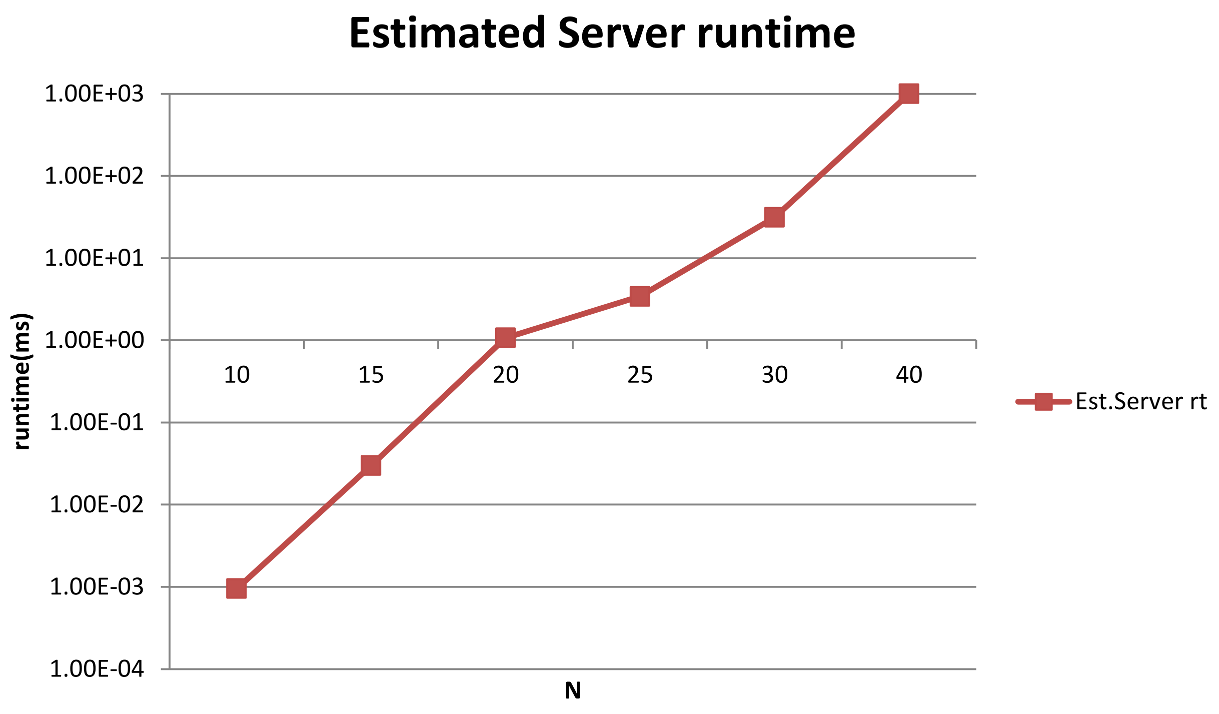


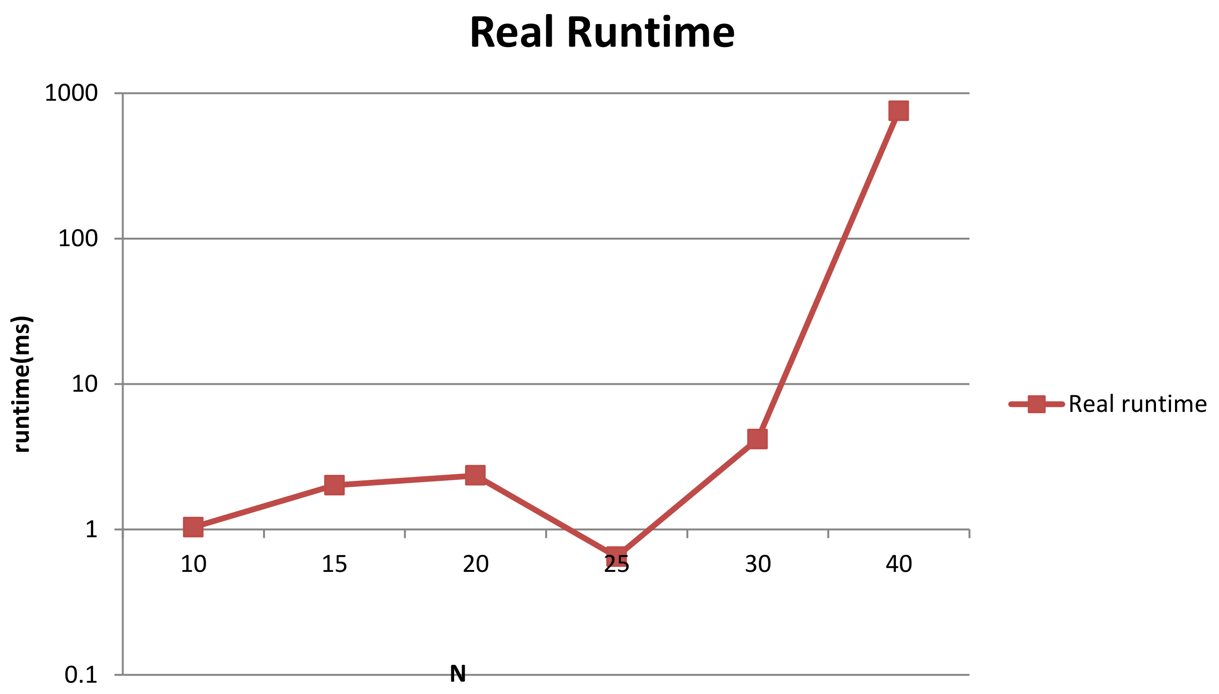
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
runtime(ms)
1000
5000
10000
20000
80000
100000
20000000
40000000
60000000
80000000
100000000
N
Est. Android rt
Est.Server rt
Real runtime
Offloading time

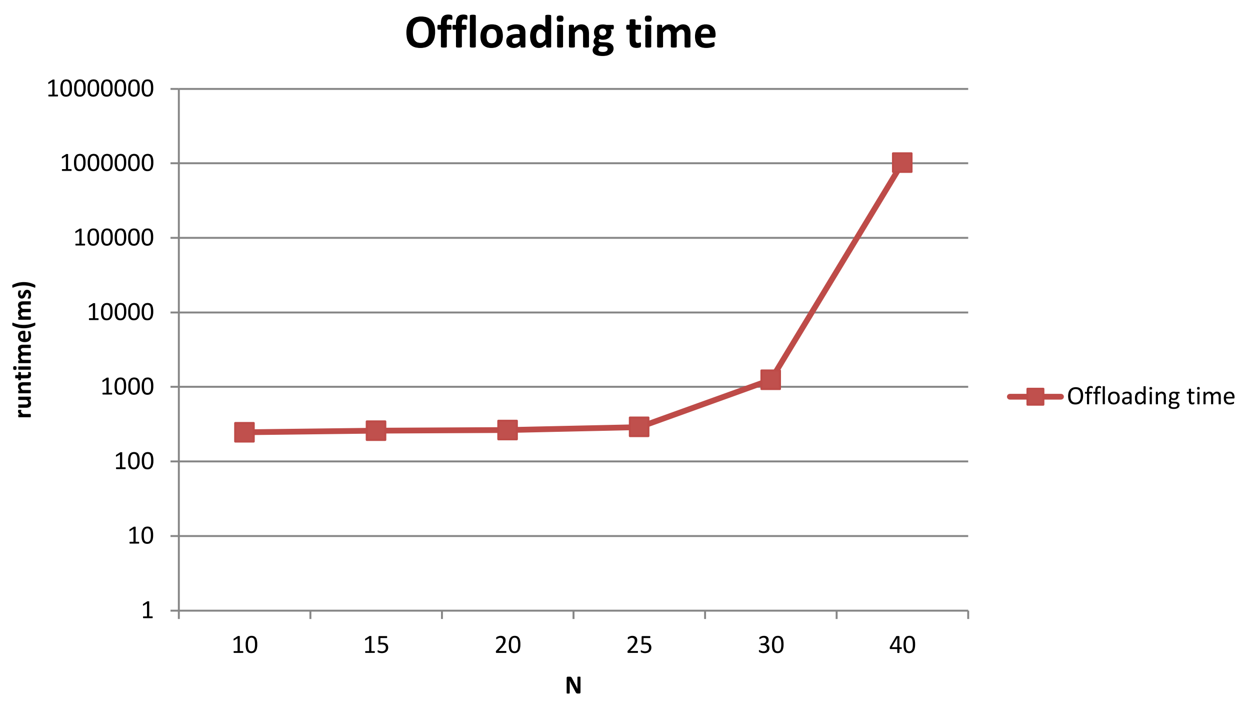
5.2.3 FIBONACCI RECURSIVE

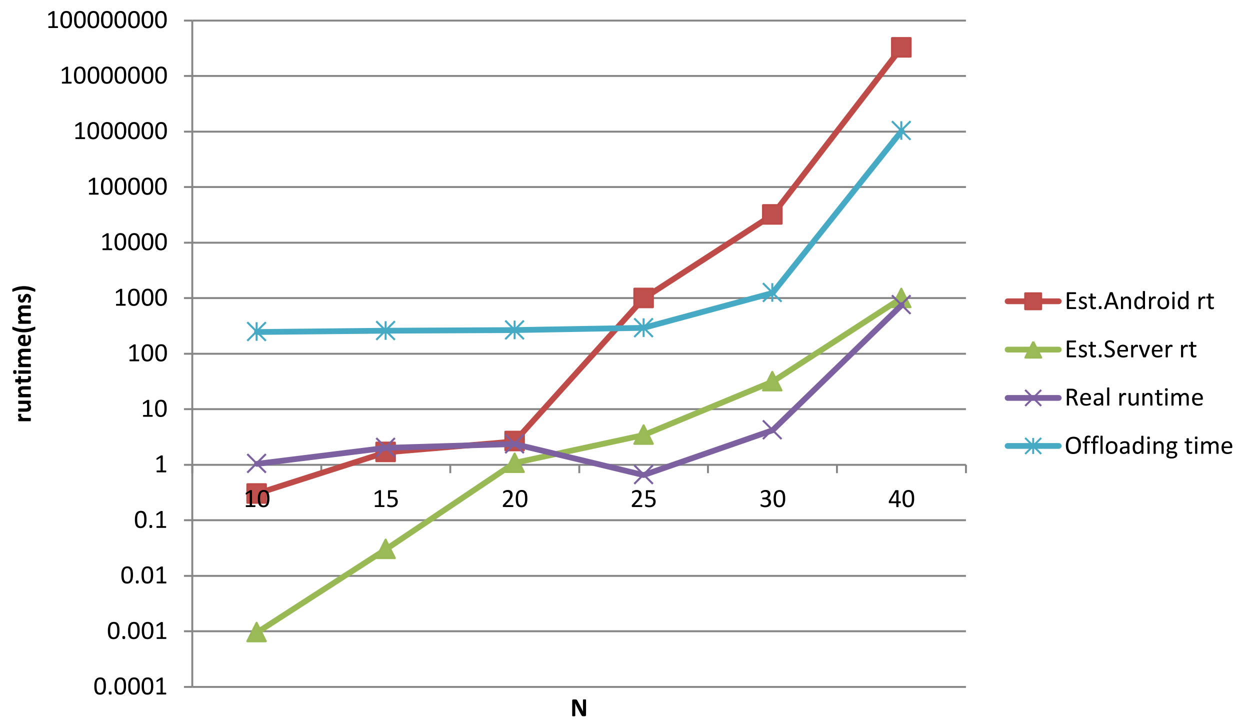
n	Est.Android rt(ms)	Est.Server rt(ms)	Real runtime(ms)	Offloading time(ms)	% saved(time) (ms)
10	0.3	9.50E-04	1.037	246	-245.667
15	1.67	0.03	2.014	258	-20.5988
20	2.63	1.07	2.349	263.74	10.68441
25	990	3.43	0.65	289.44	99.93434
30	31693	31.34	4.174	1243.07	99.98683
40	3.24E+07	997.04	751.79	1021238	99.99768




Estimated Server runtime
1.00E+03
1.00E+02
1.00E+01
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
runtime(ms)
10
15
20
25
30
40
N
Est.Server rt

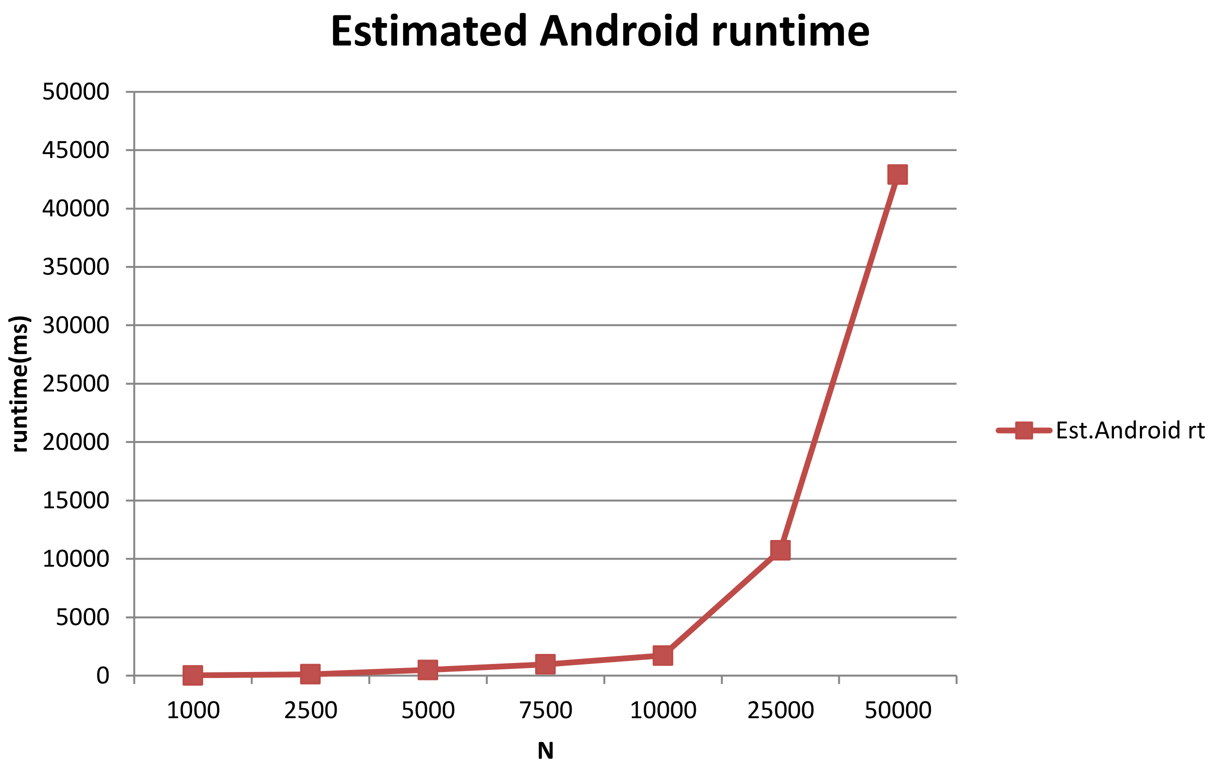

Real Runtime
1000
100
10
1
0.1
runtime(ms)
10
15
20
25
30
40
N
Real runtime

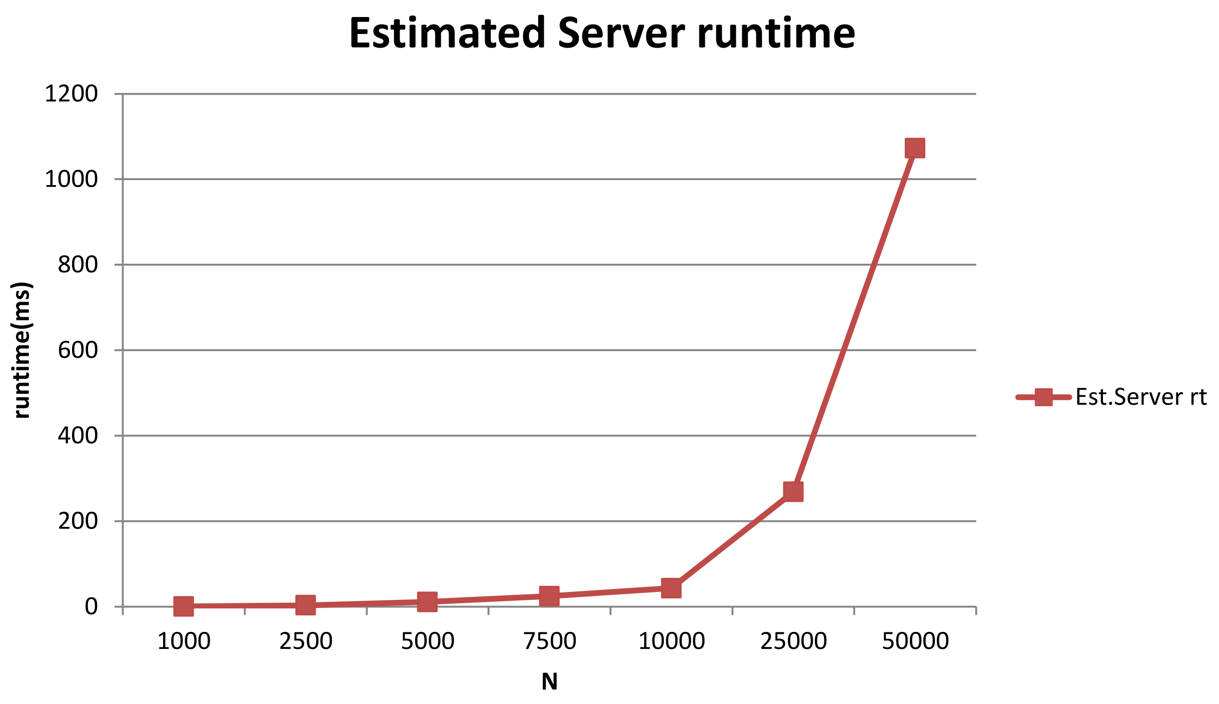

Offloading time
runtime(ms)
10000000
1000000
100000
10000
1000
100
10
1
10
15
20
25
30
40
N
Offloading time


runtime(ms)
100000000
10000000
1000000
100000
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
10
15
20
25
30
40
N
Est.Android rt
Est.Server rt
Real runtime
Offloading time

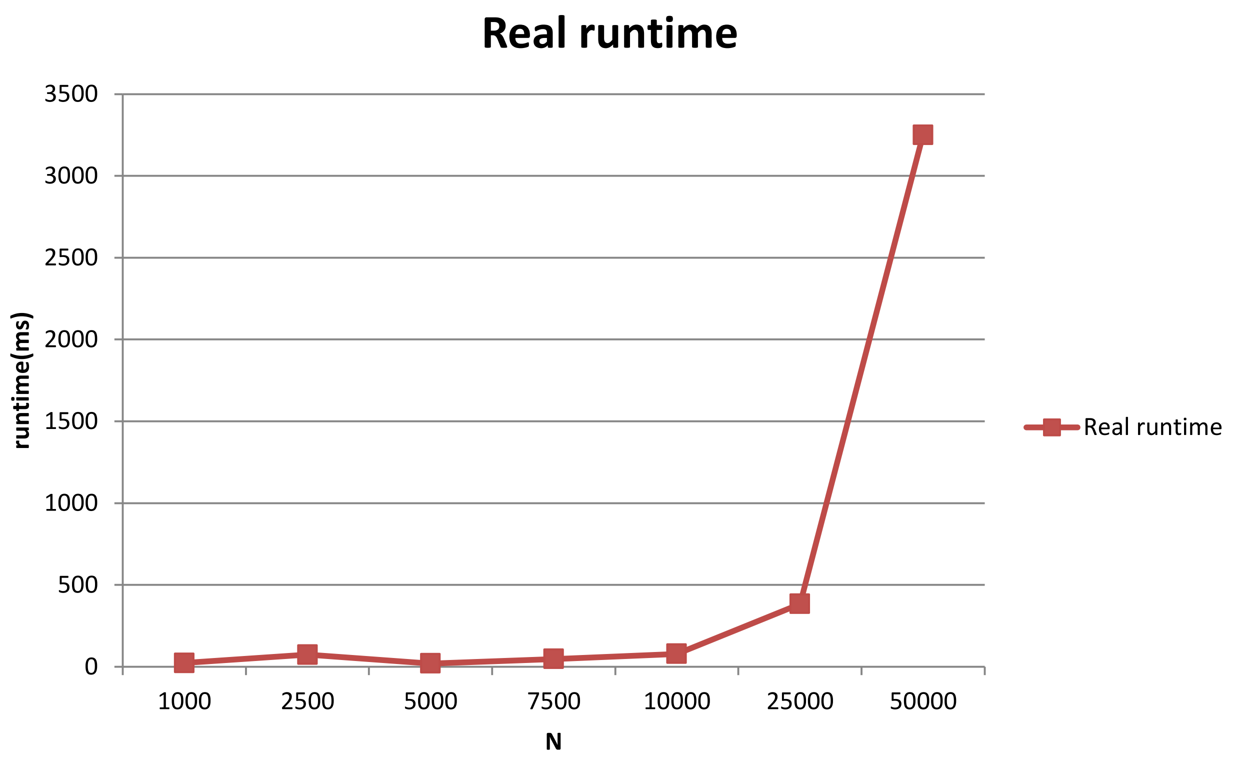
5.2.4 SORT ARRAY

n	Est.Android rt(ms)	Est.Server rt(ms)	Real runtime(ms)	Offloading time(ms)	%saved(time) (ms)
1000	18.46	0.44	22.613	259.17	-22.4973
2500	120.92	2.72	73.36	269.53	39.33179
5000	479.57	10.8	19.76	261.45	95.87964
7500	970.14	24.24	47.74	282.97	95.07906
10000	1722.2	43.04	78.279	301.78	95.45471
25000	10735.83	268.32	384.148	527.33	96.42181
50000	42906.05	1072.36	3248.96	1331.23	92.42773

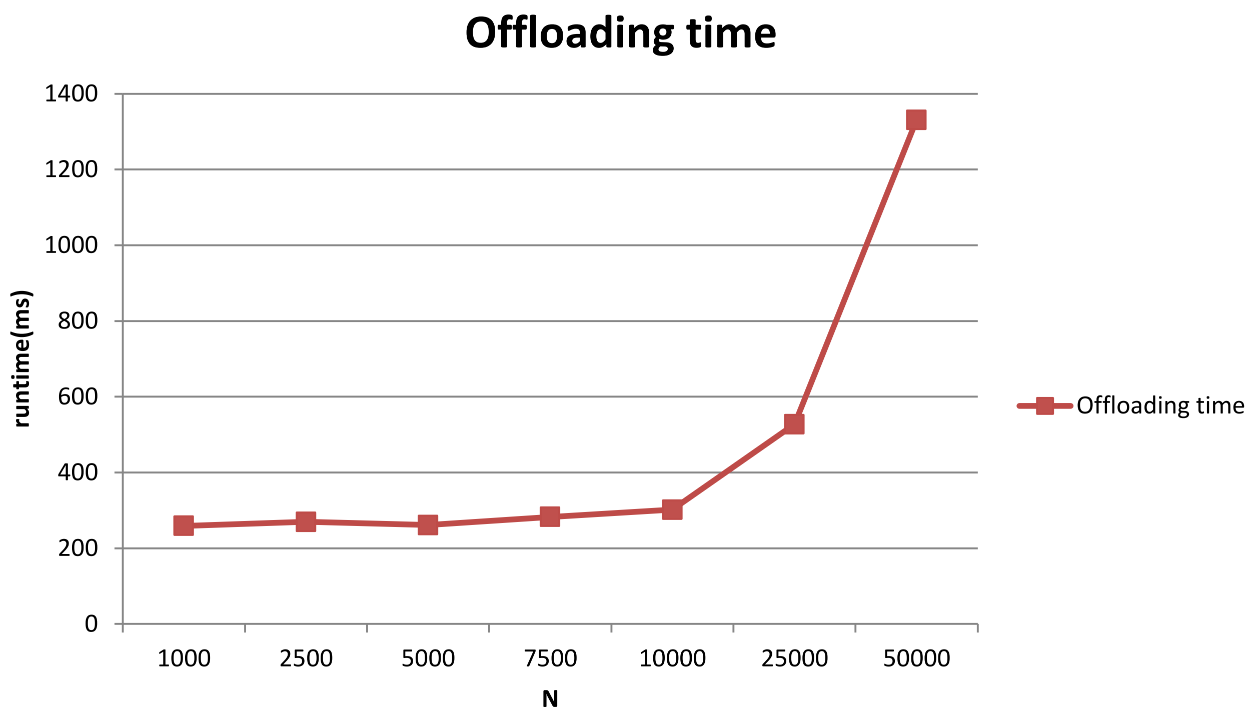


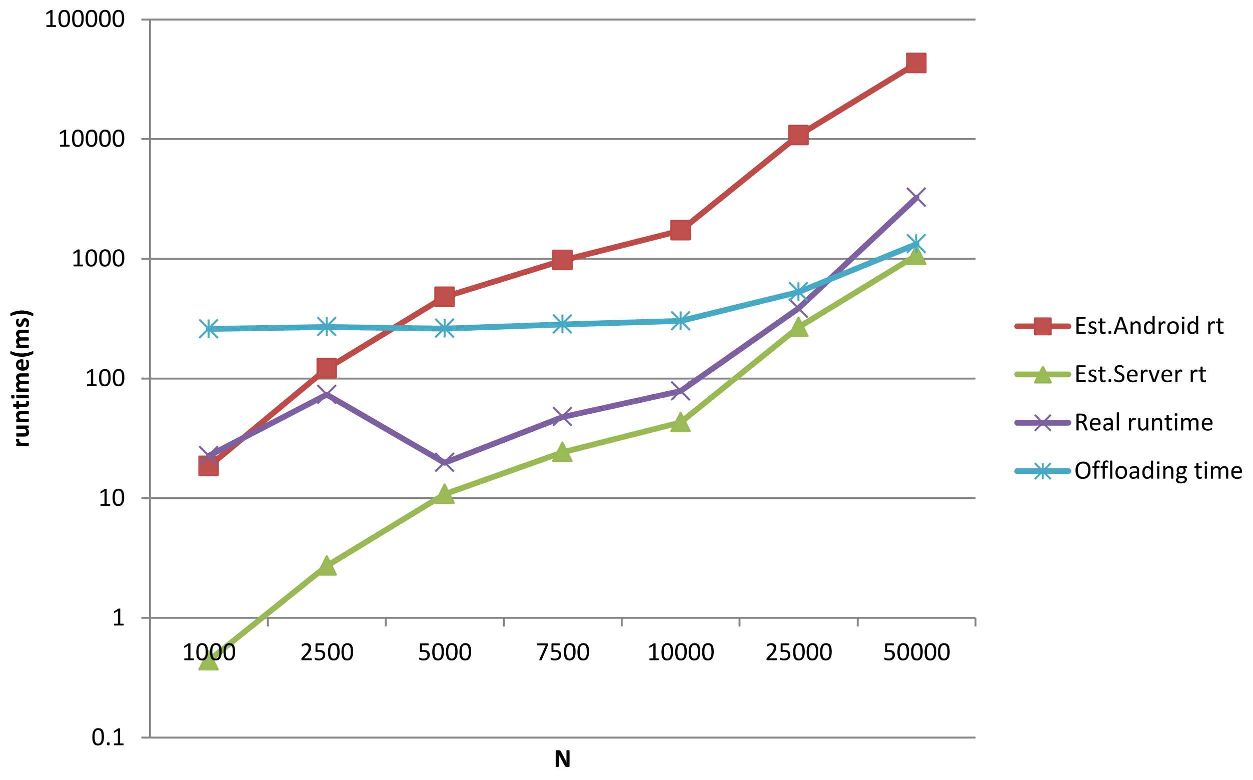


Estimated Server runtime
1200
1000
800
600
400
200
0
runtime(ms)
1000
2500
5000
7500
10000
25000
50000
N
Est.Server rt



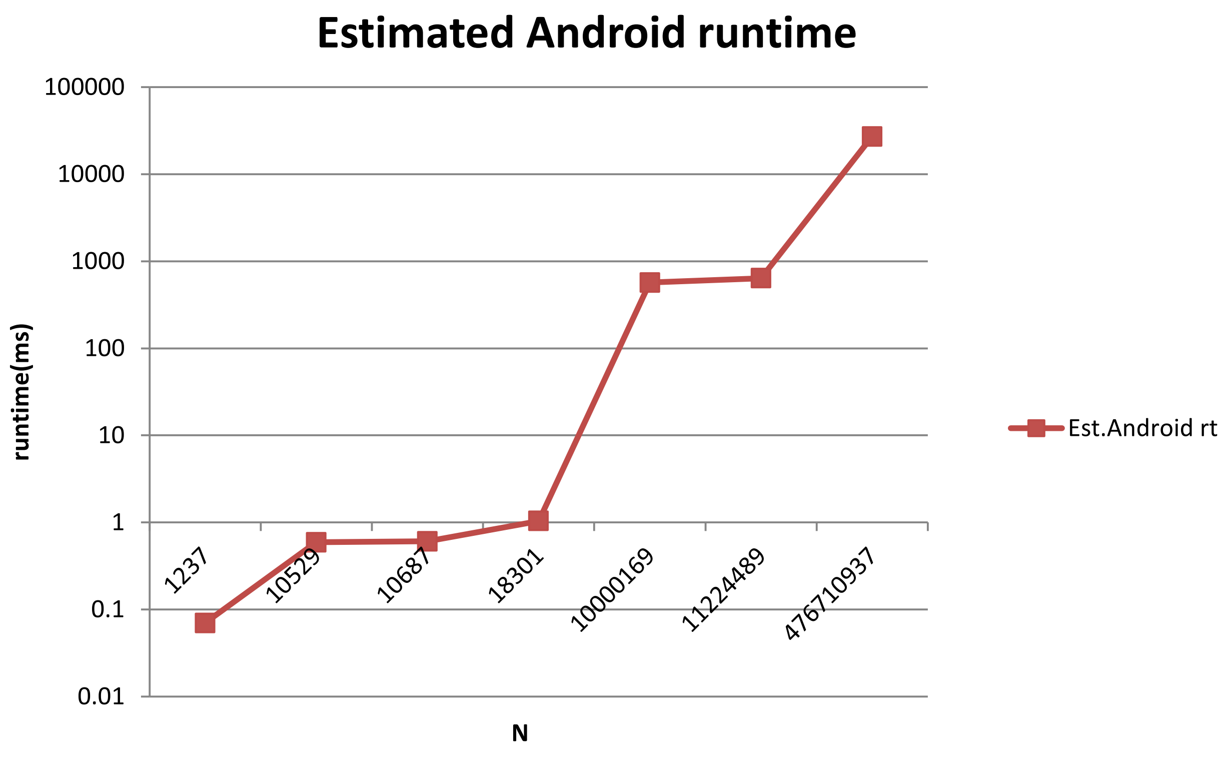
Real runtime
3500
3000
2500
2000
1500
1000
500
0
runtime(ms)
1000
2500
5000
7500
10000
25000
50000
N
Real runtime

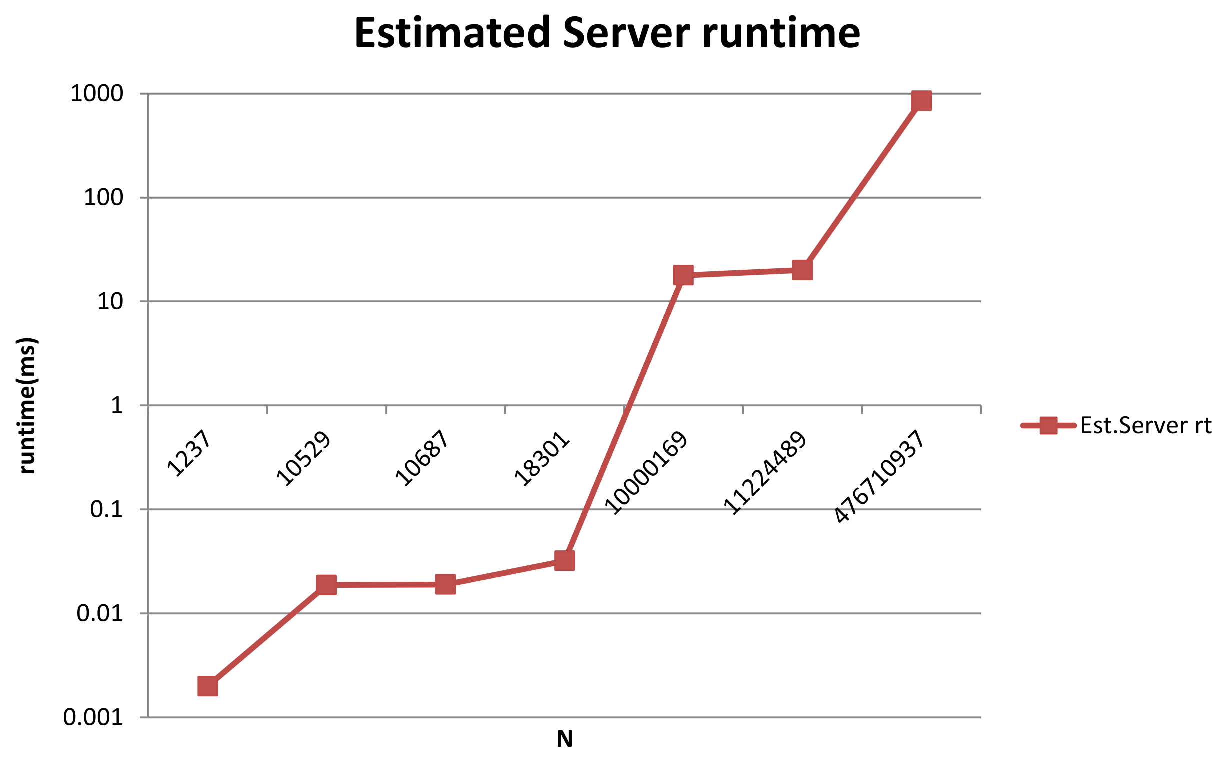

Offloading time
runtime(ms)
1400
1200
1000
800
600
400
200
0
1000
2500
5000
7500
10000
25000
50000
N
Offloading time

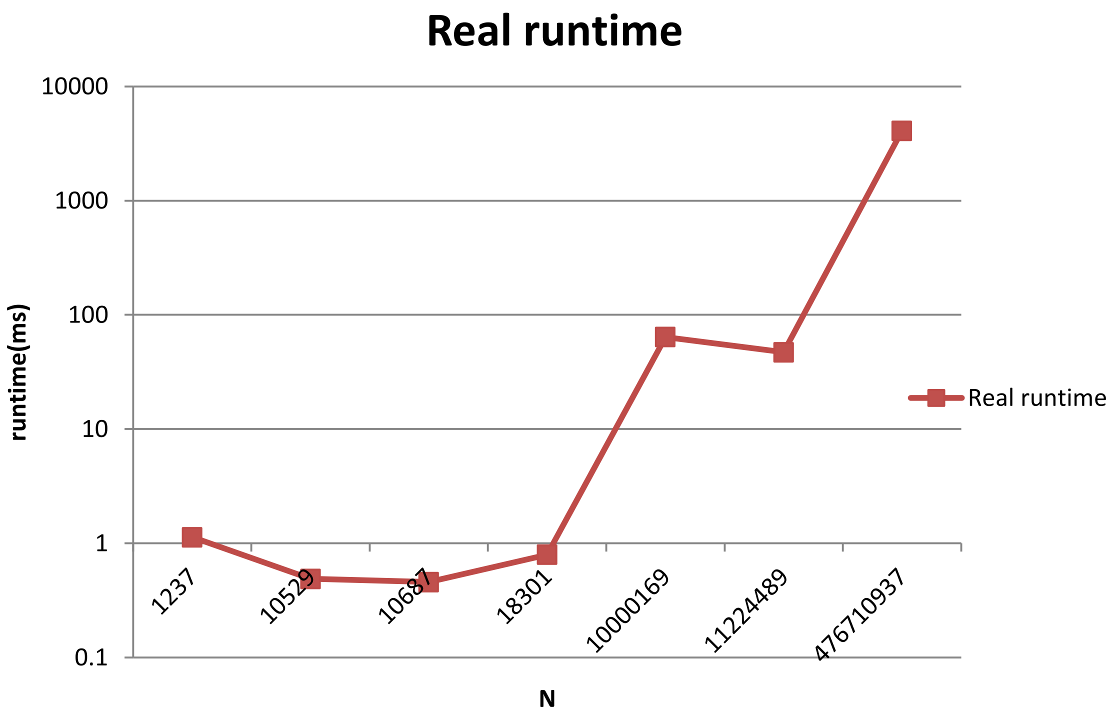

runtime(ms)
100000
10000
1000
100
10
1
0.1
1000
2500
5000
7500
10000
25000
50000
N
Est.Android rt
Est.Server rt
Real runtime
Offloading time

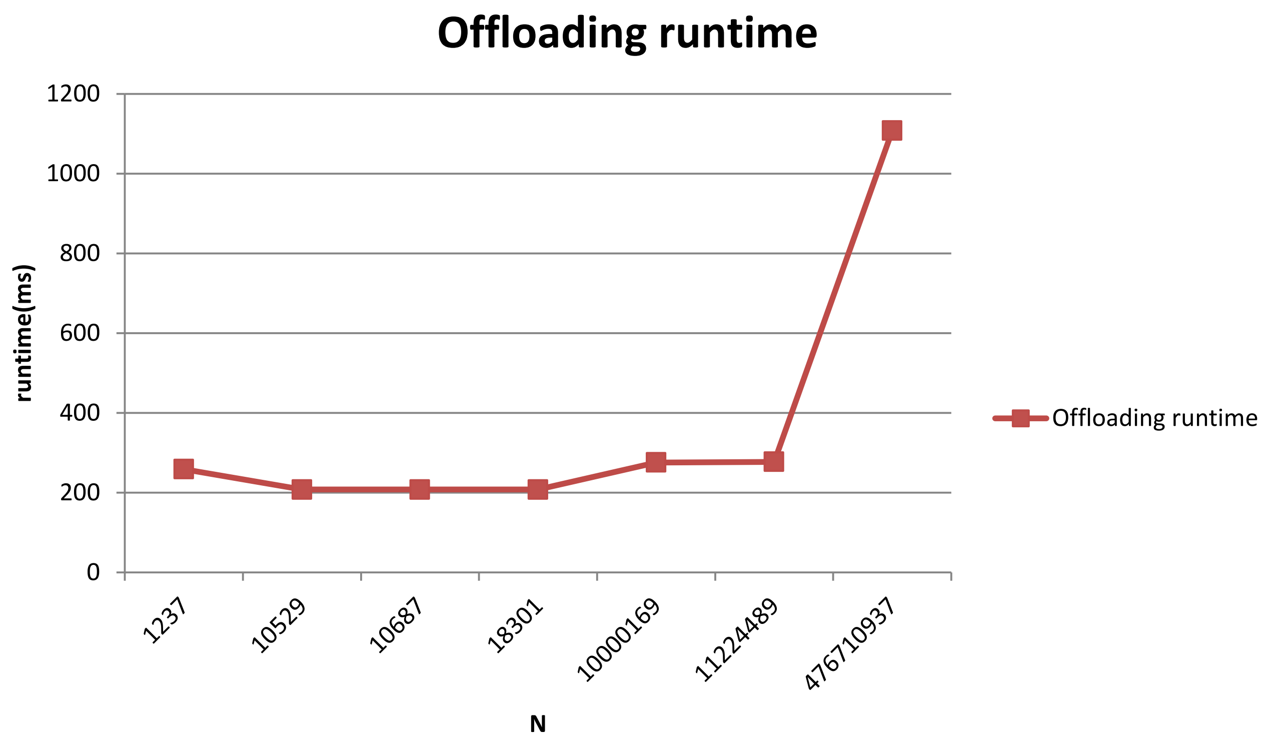
5.2.5 IS PRIME ?

n	Est.Android rt(ms)	Est.Server rt(ms)	Real runtime(ms)	Offloading runtime(ms)	%saved(time) (ms)
1237	0.07	0.002	1.1291	258.73	-1513
10529	0.59	0.0188	0.488	207.82	17.28814
10687	0.606	0.019	0.4577	207.82	24.47195
18301	1.038	0.032	0.793	207.82	23.60308
10000169	569.63	17.85	64	275.66	88.76464
11224489	637.13	20.04	47	277.34	92.62317
476710937	27059	851	4091	1108	84.88119

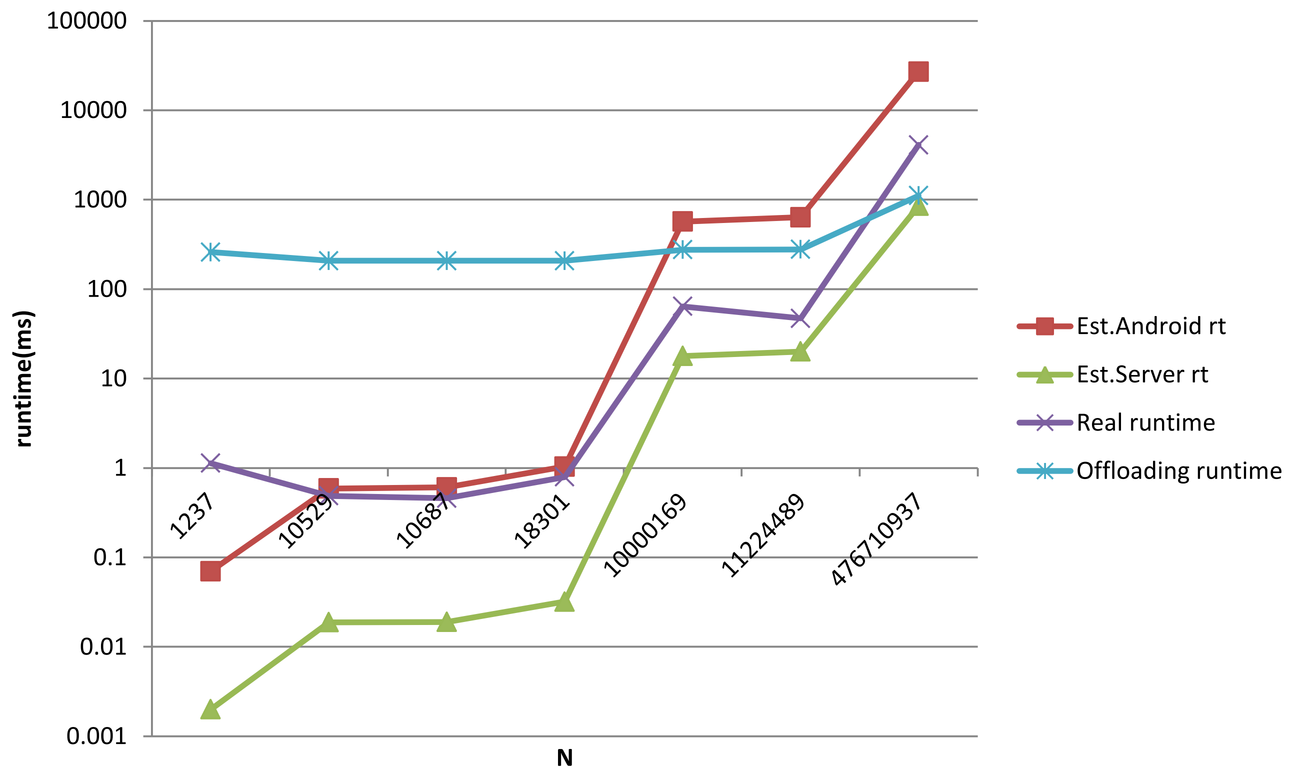



Estimated Server runtime
1000
100
10
1
0.1
0.01
0.001
runtime(ms)
1237
10529
10687
18301
10000169
11224489
476710937
N
Est.Server rt


Real runtime
10000
1000
100
10
1
0.1
runtime(ms)
1237
10529
10687
18301
10000169
11224489
476710937
N
Real runtime

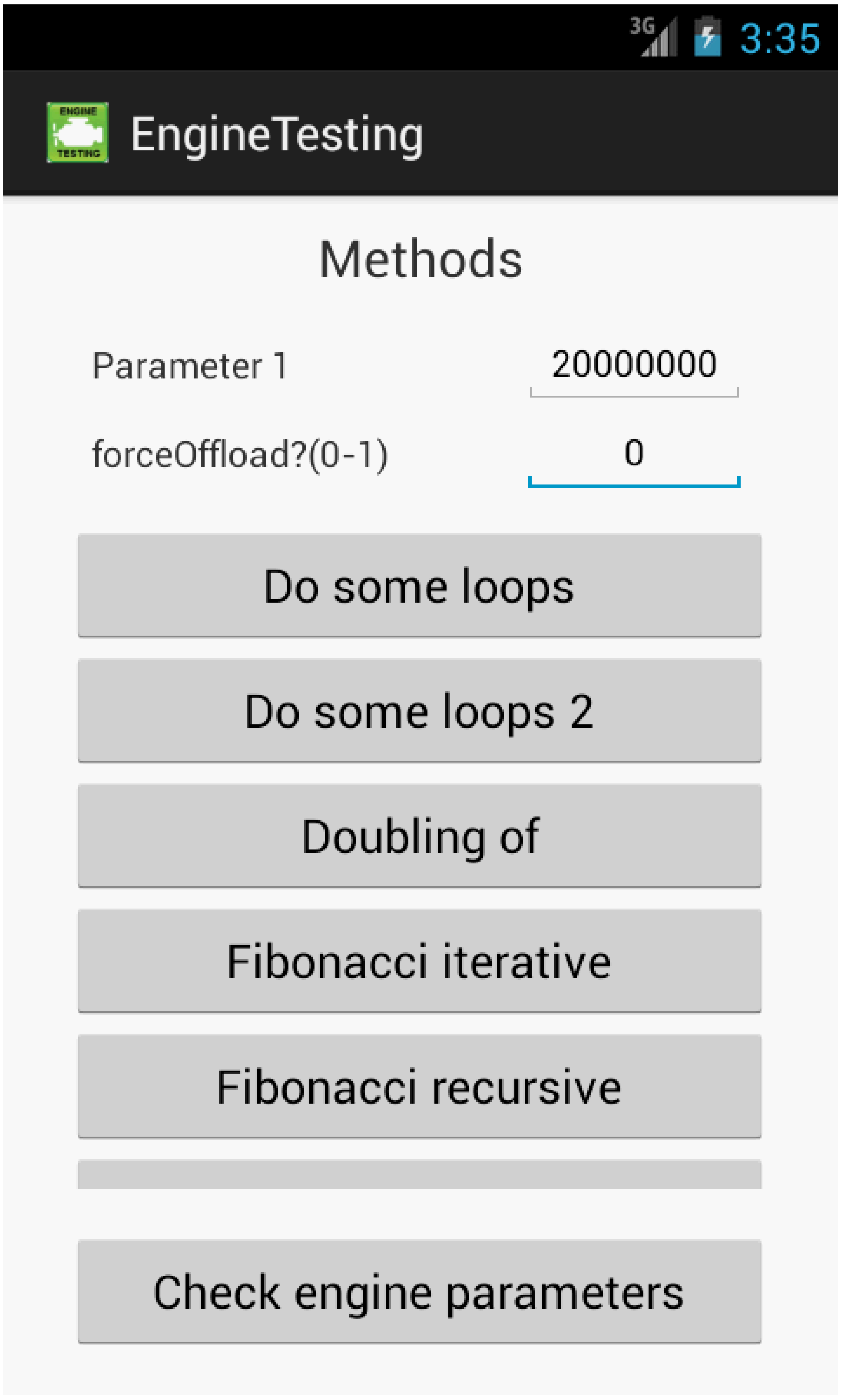


Offloading runtime
runtime(ms)
1200
1000
800
600
400
200
0
1237
10529
10687
18301
10000169
11224489
476710937
N
Offloading runtime



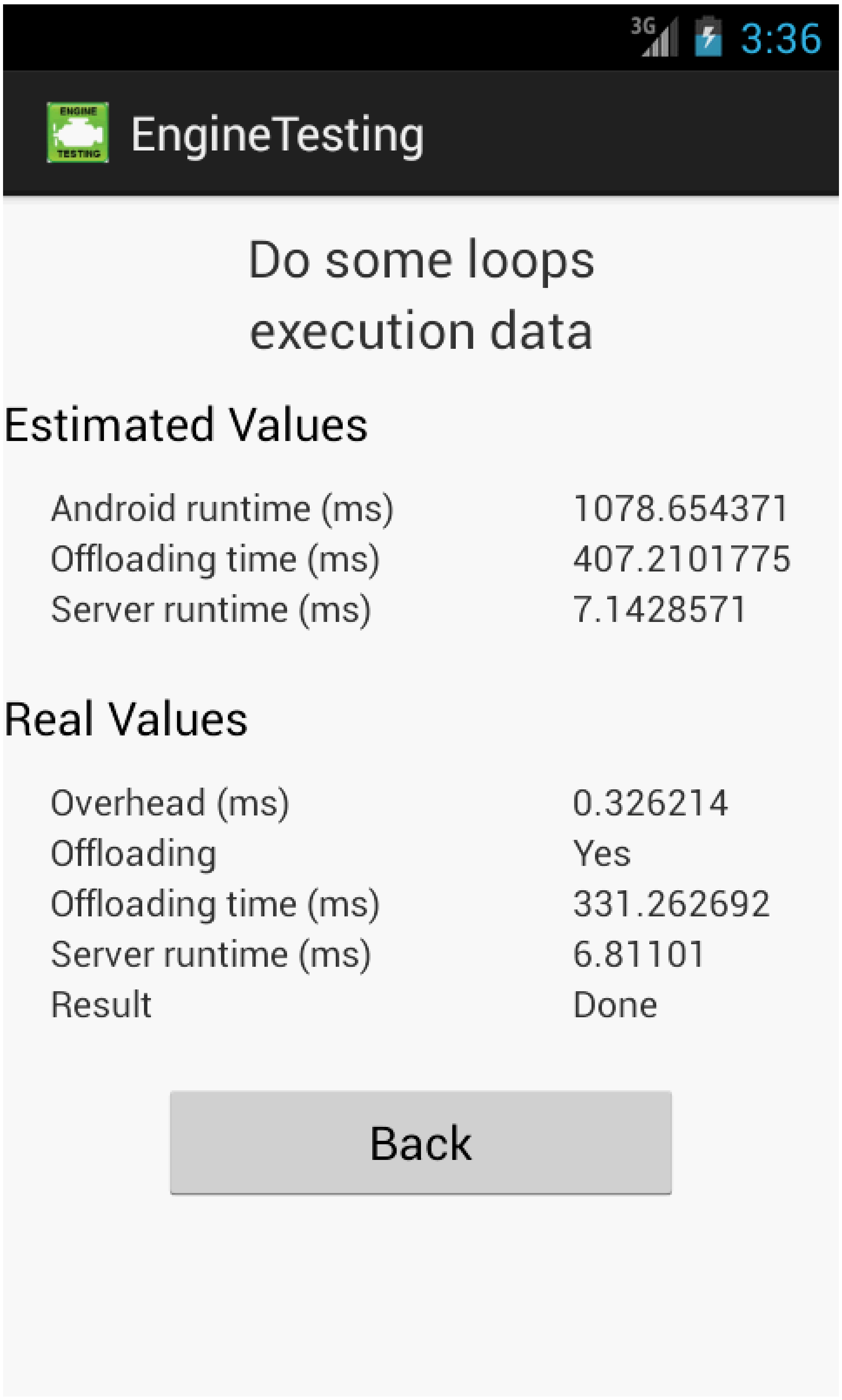
runtime(ms)
100000
10000
1000
100
10
1
0.1
0.01
0.001
1237
10529
10687
18301
10000169
11224489
476710937
N
Est.Android rt
Est.Server rt
Real runtime
Offloading runtime

5.3 SCREENSHOTS



Parameter 1 20000000

forceOffload?(0-1) 0



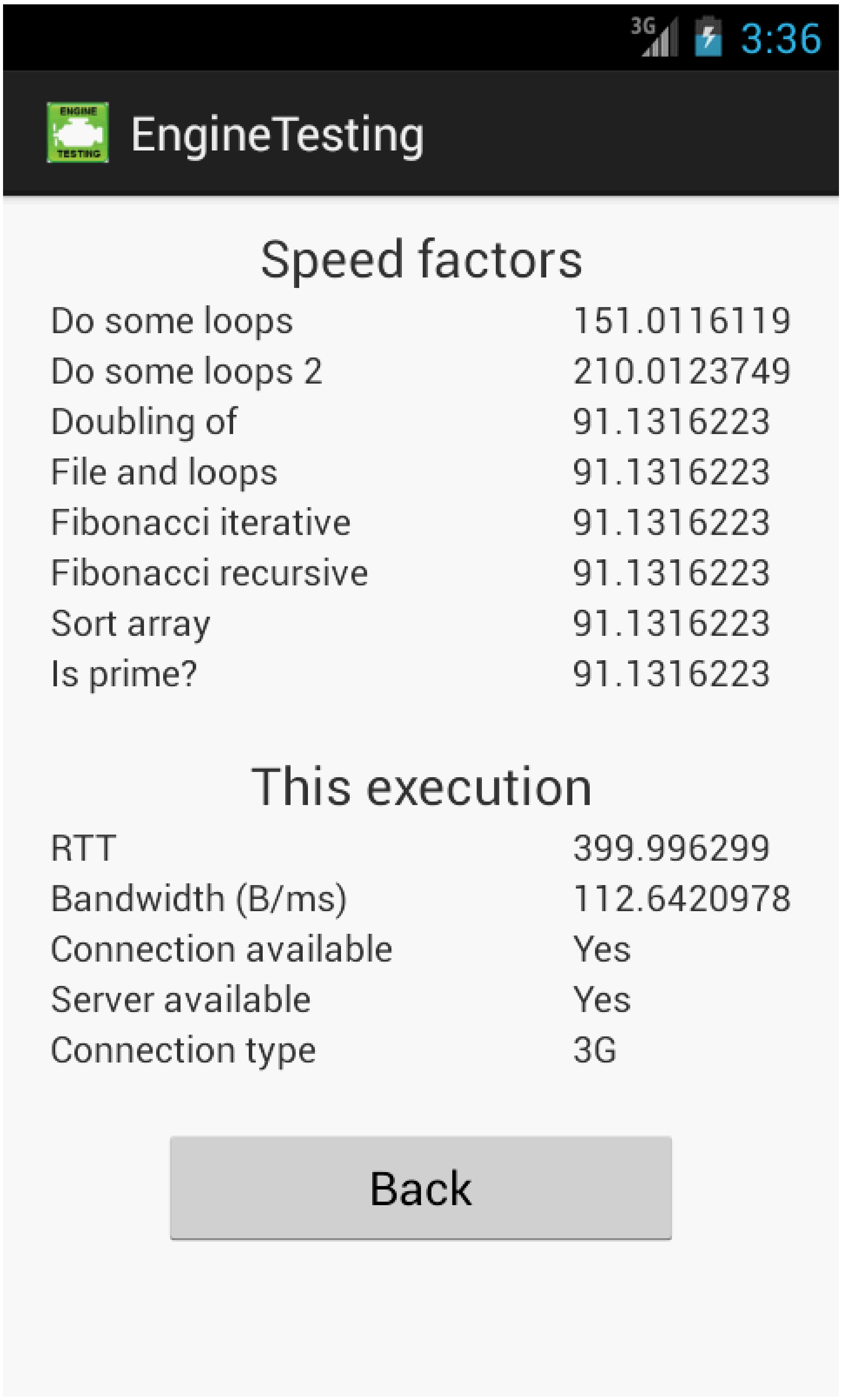
Do some loops execution data

Estimated Values

Android runtime (ms)	1078.654371
Offloading time (ms)	407.2101775
Server runtime (ms)	7.1428571

Real Values

Overhead (ms)	0.326214
Offloading	Yes
Offloading time (ms)	331.262692
Server runtime (ms)	6.81101
Result	Done

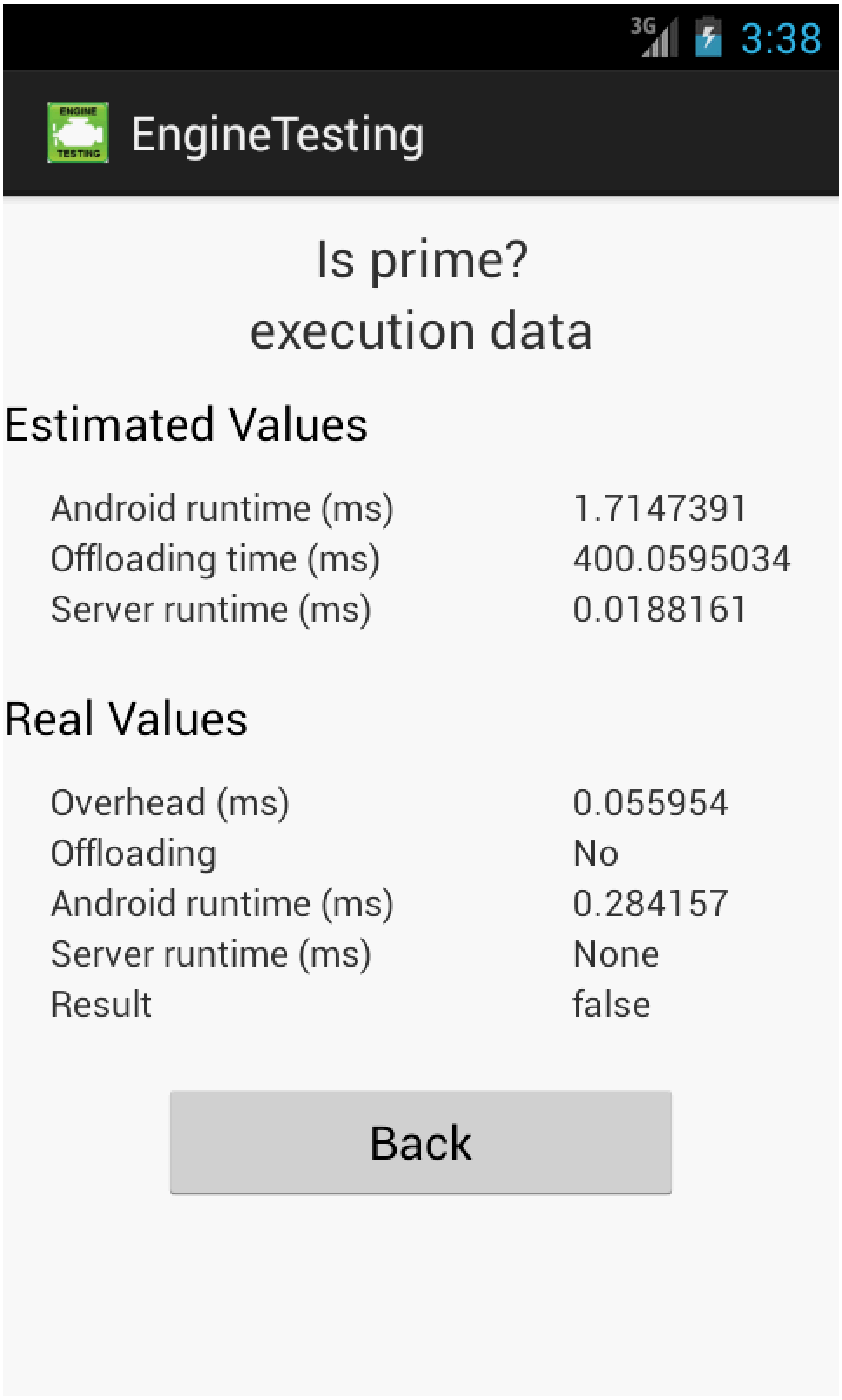


Speed factors

Do some loops	151.0116119
Do some loops 2	210.0123749
Doubling of	91.1316223
File and loops	91.1316223
Fibonacci iterative	91.1316223
Fibonacci recursive	91.1316223
Sort array	91.1316223
Is prime?	91.1316223

This execution

RTT	399.996299
Bandwidth (B/ms)	112.6420978
Connection available	Yes
Server available	Yes
Connection type	3G



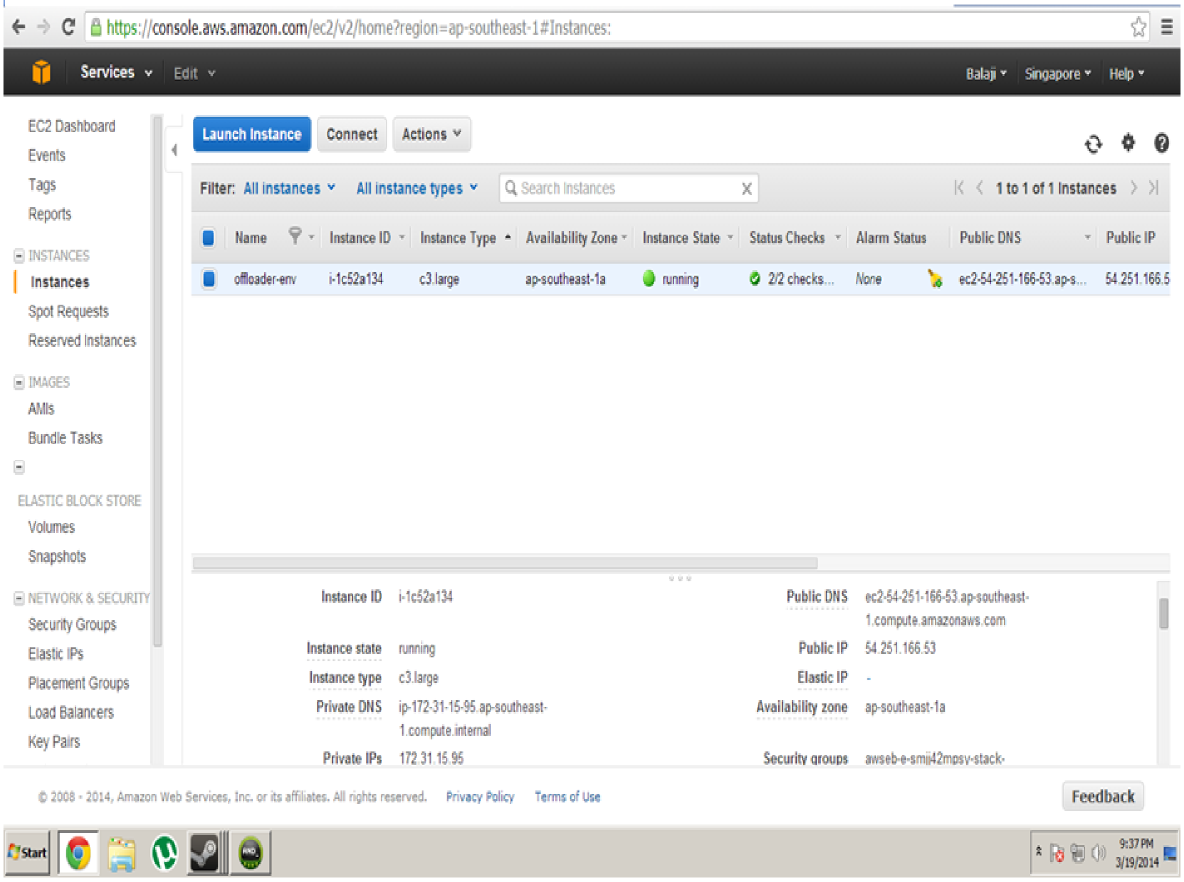
Is prime?
execution data

Estimated Values

Android runtime (ms)	1.7147391
Offloading time (ms)	400.0595034
Server runtime (ms)	0.0188161

Real Values

Overhead (ms)	0.055954
Offloading	No
Android runtime (ms)	0.284157
Server runtime (ms)	None
Result	false



https://console.aws.amazon.com/ec2/v2/home?region=ap-southeast-1#Instances:

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP
offloader-env	i-1c52a134	c3.large	ap-southeast-1a	running	2/2 checks...	None	ec2-54-251-166-53.ap-s...	54.251.166.5

Instance ID	i-1c52a134	Public DNS	ec2-54-251-166-53.ap-southeast-1.compute.amazonaws.com
Instance state	running	Public IP	54.251.166.53
Instance type	c3.large	Elastic IP	-
Private DNS	ip-172-31-15-95.ap-southeast-1.compute.internal	Availability zone	ap-southeast-1a
Private IPs	172.31.15.95	Security groups	awseb-e-smij42mpsv-stack-

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CHAPTER 6

CONCLUSION

In this work, we constructed a middleware which is used for adaptively offloading processes involving high computation from the mobile phone to the cloud. Computational power of mobile phone is limited, this work paves way to improve the computational power of mobile device so that to match the computational power of a standalone PC. By this offloading of computation we save a large amount of energy (i.e.) battery thereby increasing the battery life of android mobiles. This type of adaptive offloading paves way to implementation of high processing applications which involves high graphics and calculation like AutoCAD ,image processing etc.. We found out that almost 95% of energy is saved. Thus we can conclude that adaptive computational offloading using cloud is going to be the future of mobile technology which would explore new possibilities with mobile phones.

Future enhancement would be to implement a generic middleware which would make decision on its own depending upon its own calculations for any application installed in the android mobile device whether to offload a process or to run locally

APPENDIX A

SOURCE CODE

Listen for network activity

```
privateclassNetworkReceiverextendsBroadcastReceiver {

		@Override
		publicvoidonReceive(Context context, Intent intent) {
			ConnectivityManager conn =  (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
			NetworkInfonetworkInfo = conn.getActiveNetworkInfo();
			if (networkInfo != null&&networkInfo.isConnected()) {
				if (connAvailable == false) calcPingAndBandwidth();
				connAvailable = true;
				intnetType = networkInfo.getType();
				intnetSubtype = networkInfo.getSubtype();
				if (netType == ConnectivityManager.TYPE_WIFI)
connType = "Wi-Fi";
				else {
					if (netType ==
ConnectivityManager.TYPE_MOBILE&&netSubtype ==
TelephonyManager.NETWORK_TYPE_UMTS) connType = "3G";
					elseconnType = "Other";
				}
			}
			else {
				connType = "None";
				connAvailable = false;
				serverAvailable = false;
```

```
                    }
                }}
```

The above code uses a Broadcast receiver to continuously listen for connection and server availability in the network. If a connection is available the connection type is also determined. A broadcast receiver is registered for a particular event which triggers its corresponding code when the registered event occurs, in this case a network connection.

Calculating computation speed relation

```
privatevoidcalculateRelation() {
                doublestartAlgorithmTime = ((double) System.nanoTime()) / 
1000000.0;
                Algorithms.executeLocally(AlgName.doSomeLoops, 
Long.valueOf(1000000).toString());
                doubletimeTaken = ((double) System.nanoTime()) / 1000000.0 - 
startAlgorithmTime;
                doubleAndroidInstMs = 
Algorithms.getCost(AlgName.doSomeLoops, 
Long.valueOf(1000000).toString()) / timeTaken;
                floatfirstTimeCsr = (float) (SERVER_INST_MS / AndroidInstMs);
                Iterator<AlgName>enumKeySet = algCsrs.keySet().iterator();
while (enumKeySet.hasNext()) {
        AlgNameitAlgName = enumKeySet.next();
        CsrPairitCsrPair = algCsrs.get(itAlgName);
        itCsrPair.csrServerDevice = firstTimeCsr;
        itCsrPair.csrUpdatesCounter++;
      } }
```

The above code is used to calculate the initial Computation speed relation by running a sample looping algorithms 100000 times and finding the ratio of the server time to android runtime. The is ratio is assumed for all the algorithms initially.

Decision making Algorithm

```
privateboolean decide(AlgNamealgName, String... parameters) {

                estAndroidRuntime = -1;
                estOffloadingTime = -1;
                estServerRuntime = -1;

                if (connAvailable&&serverAvailable) {

                        //Only the very first time that there is the possibility to do
offloading, a initial computation speed relation is calculated
                        if (getCsrUpdCountFromAlg(algName) == 0)
calculateRelation();

                        doublestartAlgorithmTime = ((double) System.nanoTime())
/ 1000000.0;
                        estServerRuntime = Algorithms.getCost(algName,
parameters) / SERVER_INST_MS; //Estimated server execution time
                        overhead = ((double) System.nanoTime()) / 1000000.0 -
startAlgorithmTime;
                        estAndroidRuntime = estServerRuntime *
getCsrFromAlg(algName); //Estimated Android execution time
```

```
                        for (int i = 0; parameters != null&& i <parameters.length;
i++) parametersSize += parameters[i].length();
                        estOffloadingTime = ping + estServerRuntime +
parametersSize/transferredBytesMs;

                        //Time saving criteria
                        returnestOffloadingTime<estAndroidRuntime;

                }
                elsereturnfalse;
        }
```

The decision making algorithm is an important part of the engine on which the offloading decision depends. It first makes an estimation of the android runtime, server runtime and offloading time. It returns true if the android runtime is greater than the sum of the offloading time and server runtime. Otherwise it returns false.

Code to execute the algorithm

```
public String execute(booleanforceOffloading, AlgNamealgName, String...
parameters) {

                parametersSize = 0;
                overhead = -1;

                if (forceOffloading) doOffloading = true;
                elsedoOffloading = decide(algName, parameters);
```

```
        String algResult = "";
        realServerTime = -1;
        overallTime = -1;

        doublestartTime = ((double) System.nanoTime()) / 1000000.0;

        if (doOffloading) { //Do offloading
            intexecParamsLength = 1;
            if (parameters != null) execParamsLength +=
parameters.length;
            String[] execParams = new String[execParamsLength];
            execParams[0] = algName.toString();
            for (int i = 1; i <execParams.length; i++) execParams[i] =
parameters[i-1];
            GetServerDatagetServerData = newGetServerData();
            getServerData.execute(execParams);
            try {
                algResult = getServerData.get(); //This can also return
the word "Error"
            } catch (Exception e) {
                e.printStackTrace();
                serverAvailable = false;
                algResult = "Error";
            }
            if (algResult.equals("Error")) { //Unable to retrieve the data.
URL may be invalid or the server may be down.
                serverAvailable = false;
            }
            else {
```

```
                                        try {
                                                  realServerTime =
Double.parseDouble(getElementValueFromXML(algResult, "runtime"));
                                                  algResult =
getElementValueFromXML(algResult, "result");
                                        } catch (Exception e) {
                                                  e.printStackTrace();
                                                  algResult = "Error";
                                        }
                              }
                    }

                    //In case of a failed offloading attempt, we don't retry offloading,
we make the system behave like the decision would have been to not offload
                    //(the main reason of failing is losing the network connection,
which takes too long to recover)
                    if (algResult.equals("Error")) {
                              doOffloading = false;
                              startTime = ((double) System.nanoTime()) / 1000000.0;
                    }
                    //Do not offload, execute locally in the Android mobile device
                    if (!doOffloading) algResult =
Algorithms.executeLocally(algName, parameters);

                    //This can be either the Android runtime or the total offloading
time
                    overallTime = ((double) System.nanoTime()) / 1000000.0 -
startTime;
```

```
                if (doOffloading&& !algResult.equals("Error")) { //If offloading
was done successfully
                        if (parametersSize == 0) { //decide was not called
                                for (int i = 0; parameters != null&& i
<parameters.length; i++) parametersSize += parameters[i].length();
                        }
                        doubletransferDataTime = overallTime - realServerTime -
ping;
                        //If the size of the sent parameters was big enough to be
significant and the transferDataTime is also significant (bigger than 5
milliseconds), update the transferredBytesMs.
                        if (parametersSize> 1024 &&transferDataTime>= 5.0) {
                                if (transferredBytesMsUpdatesCounter< 20)
transferredBytesMsUpdatesCounter++;
                                transferredBytesMs = transferredBytesMs *
transferredBytesMsUpdatesCounter / (transferredBytesMsUpdatesCounter+1) +
(parametersSize / transferDataTime) / (transferredBytesMsUpdatesCounter+1);
                        }
                }

                //Updates the Csr and/or the costs DB when needed
                if (!doOffloading || !algResult.equals("Error")) {

        UpdateCostCalcSystemsThreadupdateCostCalcSystemsThread =
newUpdateCostCalcSystemsThread(algName);
                        updateCostCalcSystemsThread.start();
                }
                returnalgResult;
}
```

The execution algorithm first calls the decision maker to determine whether to offload the code or not. The code is always offloaded if force offloading is set true. Based on the decision by the decision module the code is executed in the cloud using the communication module or executed locally on the android platform.

Code to calculate ping and Bandwidth

```
privatevoidcalcPingAndBandwidth() {
		pingCounter = 0;
		pingsArray = newdouble[10];
		timePingStart = ((double) System.nanoTime()) / 1000000.0;
		//The next will call itself recursively 10 times and calculate the
average ping (removing outliers)
		//Once done, it will call calcTransferredBytesPerMs() to calculate
the bandwidth quality
		newGetPing().execute(SERVER_URL);
	}
```

The bandwidth and the ping of the network connection to the cloud is calculated by pinging the ip of the server to determine the RTT.
The bandwidth is calculated by sending a sample file to the server and recording the time require to send. The bandwidth is calculated as the file size divided by the time taken to transfer.

Code of sample algorithms and cost functions

```
privatestatic String doSomeLoops(longnLoops) {
		long i = 0;
```

```
        while (i <nLoops) i++;
        return"Done";
}
privatestaticdoubledoSomeLoopsCost(longnLoops) {
        returnnLoops * 5.0;
}
privatestaticintfileAndLoops(longnLoops, String fileContents)
{
        long i = 0;
        while (i <nLoops) i++;
        returnfileContents.length();
}

privatestaticdoublefileAndLoopsCost(longnLoops)
{
        returnnLoops * 5.0;
}


privatestaticdoublefibonacciIterativeCost(int n)
{
        return n * 6.0;
}


privatestaticdoublefibonacciRecursiveCost(int n)
{
        returnMath.pow(2.0, n) * 13.0;
}
```

```
privatestaticdoublerandomArraySelectionSortCost(int n)
{
	return n * 255.0 + ((n * (n + 1.0)) / 2) * 12.0;
}

privatestaticdoubleisPrimeCost(int n)
{
	return n * 25.0;
}
```

The MACS library contains the code of the potentially offload able code and their cost functions which are an approximation of the number of low level instructions that would be required to run the algorithm depending on the input parameter. The number of low level instructions is calculated using the javap –c program available with the jdk.

REFERENCES

1.K. Yang, S. Ou, and H.H. Chen, “On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications,” IEEE Comm. Magazine, vol. 46, no. 1, 2008, pp. 56-63.

2. C. Wang and Z. Li, “Parametric Analysis for Adaptive Computation Offloading,” ACM SIGPLAN Notices, vol. 39, no. 6, 2004, pp. 119-130.

3.R. Wolski et al., “Using Bandwidth Data to Make Computation Offloading Decisions,” Proc. IEEE Int’l Symp. Parallel and Distributed Processing (IPDPS 08), 2008, pp. 1-8.

4. E. Cuervo, A. Balasubramanian, D. 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 onMobile Systems, Applications, and Services (ACM MobiSys ’10)*, pp.49–62, San Francisco, CA, USA, 2010. ACM.

5.X. Zhang, S. Jeong, A. Kunjithapatham, and S. Gibbs, “Towards anElastic Application Model for Augmenting Computing Capabilities of Mobile Platforms,” in *Third International ICST Conference on MobileWireless Middleware, Operating Systems, and Applications*, pp.161-174, Chicago, IL, USA, 2010.

6. X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, “Towards an Elastic Application Model for Augmenting the Computing Capabilitiesof Mobile Devices with Cloud Computing,” Mobile Networks and Applications, vol. 16, pp. 270–284, 2011.

7. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The
Case for VM-based Cloudlets in Mobile Computing. IEEE
Pervasive Computing, 8(4), 2009.

8. Amazon Elastic Computing. http://aws.amazon.com/ec2/.

9. ADT Eclipse plugin. http://developer.android.com/sdk/eclipse-adt.html.

10. Android. http://developer.android.com/.