Performance Engineering for Native Mobile Applications - Strategy, Implementation & Execution

13th Annual International Software Testing Conference 2013
Bangalore, 4th-5th December 2013

Arit Kumar Bishwas

Capgemini India Pvt. Ltd.
A-1, Technology Park, MIDC, Talwade,
Pune-412114, Maharashtra, India
Abstract

The native mobile applications are becoming more & more critical to the business. It is now obligatory for businesses to ensure the top class performance of native mobile applications to the end users. Performance engineering of any native mobile application is very complex to handle as there are multi-dimensional matrix combinations for devices (Samsung Series, iPhone, Nokia, Blackberry etc.), operating systems (Android, iOS, Windows etc.), networks (2G, 3G, 4G, WiFi) and different mobile servers engagement. In native mobile application performance engineering, we need to focus on both the sides - client side (Mobile Devices) as well as server side. So we have to figure out an intelligent and innovative strategy to optimize the performance by inspecting both client side as well as server sides of the application. In this paper, an elegant approach & strategy has been projected. Here we will discuss how to implement such a performance engineering framework which offers a detailed set of solutions to engineer the performance of the native mobile applications. As a case study we will investigate & employee the performance engineering of Native Android Mobile Application in this paper.

Key Factors: Performance Engineering, Native Mobile Applications

1. Introduction

This is the era of mobile communication. Few years back when we used to think about mobile devices we thought about calling, taking photos and internet browsing only. Now the era is getting changed so rapidly that mobile devices are dominating the world very stunningly. Thanks to the technology advancements, today’s Mobile devices are highly capable. The mobile devices are capable of handling all kinds of necessary activities that a computer can offer. We don’t need to sit around bulky computer systems for internet and other stuffs. By using mobile devices one can do all the necessary activities such as from “Online-Shopping”, “Watching Movies”, “Playing Video Games”, “Online Banking” etc to lot many more. [1]

In the current era “time” is one of the most important factors of our life, time is everything! So everyone wants the things to be done as fast as possible. When we talk about the usages of mobile phones, speed is one of the most important things to be considered. Nowadays users expect the native mobile application view to be opened within 4-5 seconds in the mobile devices in compare to the response time of 8-10 seconds in case of web pages in PCs. If your application is taking more time to respond, you may lose the user for forever! So in this era of high competition in mobile technologies, your mobile application should meet the speed threshold and it should be as fast as possible.

As per the survey with American audience conducted by Soasta, about 88% felt that they associate negative feelings with brands that have poorly performing websites and apps.

![Figure 1: User Reaction Survey with American audience conducted by Soasta](image)

In this paper we will see how we can ensure the optimum performance of any native mobile application. We are going to see the strategy required to do effective performance engineering of any native mobile application. We will also see how we can implement and execute such strategy for efficient native mobile application performance engineering. [2, 3]

2. Background

The approach to do performance test engineering of a web application is well known to us. With native mobile applications we need a different approach to do
performance engineering. In case of native mobile application’s performance engineering, the focus is on both the sides, client side as well as server side.

2.1. Native Mobile Applications

A native mobile application is an application program that has been developed for use on a particular platform or mobile device. Because native apps are written for a specific platform, they can interact with and take advantage of operating system features and other software that is typically installed on that platform. Because a native app is built for a particular device and its operating system, it has the ability to use device-specific hardware and software, meaning that native apps can take advantage of the latest technology available on mobile devices.

A native mobile application is installed directly on a mobile device and developers create a separate app version for each mobile device. The native app may be stored on the mobile device out of the box, or it can be downloaded from a public or private app store and installed on the mobile device. Data associated with the native app is also stored on the device, although data can be stored remotely and accessed by the native app. Depending on the nature of the native app, Internet connectivity may not be required. [4]

2.2. Hybrid Applications

A hybrid application is one that combines elements of both native and Web applications. Hybrid application features can function whether or not the device is connected to web. They do integration with a device’s file system, and also do integration with Web-based services. They are also having an embedded browser to improve access to dynamic online content. [5]

2.3. Mobile Web Applications

A Web application is an application program that is stored on a remote server and delivered over the Internet through a browser interface. Web services are Web apps by definition and many, although not all, websites contain Web apps. One can open a web application using the mobile browser.

2.4. Web Application Vs Native Mobile Applications in Terms of Performance Engineering

Table 1: Web Application on PC Vs Native Mobile Application

<table>
<thead>
<tr>
<th>Web Application on PC</th>
<th>Native Mobile Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each page should take approximately less than 8-10 seconds.</td>
<td>Each view should take approximately less than 4-5 seconds.</td>
</tr>
<tr>
<td>Users generally log off the application after use.</td>
<td>Users generally don’t log off the application after use, instead without log off switch to other application. More user sessions in a particular duration!</td>
</tr>
<tr>
<td>Server side analysis is on more focus.</td>
<td>Both Server side as well as client (Mobile device) side analysis are on focus in terms of performance.</td>
</tr>
<tr>
<td>Generally don’t bother about different N/Ws.</td>
<td>Each N/W (2G, 3G, 4G, and WiFi) with different N/W carriers is under consideration.</td>
</tr>
<tr>
<td>Type of PCs does not bother.</td>
<td>Too many devices with different OS versions.</td>
</tr>
</tbody>
</table>

3. MAPE (Mobile Application Performance Engineering) Process & Framework

3.1. MAPE Process

Figure 2: Mobile Application Performance Engineering (MAPE) Process Architecture
The above Figure 2 shows the process architecture for Mobile Application Performance Engineering (MAPE). For more details refer [6].

In the below Figure 2, we see the complete flow of processes that we need to follow for delivering effective and efficient performance engineering solutions for native mobile applications. We need to focus on both the sides, server side as well as client (Mobile devices!) sides.

3.2. MAPE Framework Implementation

The below Figure 3 shows the Mobile Application Performance Engineering (MAPE) process framework which has been designed for providing solutions to the native mobile application performance engineering requirements.

3.2.1. On-Device Monitoring

In case of native mobile applications we have to monitor the mobile devices and analyze the performance of the application in the device. How fast the application is working in the device. Following are some of the most important things to monitor and analyze during the performance test executions in the device.

- **CPU Utilization by the Native Mobile Application**

  We need to monitor the CPU utilization for each individual Native Mobile Application Performance Test Script (NMAPTS) execution.

- **Memory Consumed by the Native Mobile Application**

  We need to monitor the memory consumed for each individual NMAPTS execution.

- **Dalvik Memory Utilization**

  Dalvik Memory utilization is nothing but the memory consumed by the java heap in the device. Dalvik memory utilization is very important to investigate. This analysis helps in exploring if there is any memory leakage problem in the mobile device when native mobile application is running.

- **N/W Data Bytes Sent**

  We need to investigate data bytes sent by the application during NMAPTS executions.

- **N/W Data Bytes Received**

  We need to investigate data bytes received by the application during NMAPTS executions.

- **Battery Utilization**

  Analyze the power consumed by battery during native mobile application view navigation.

- **Method Profiling Analysis**

  If application launch time or any view’s response time is higher than expected, also if there is any memory leak issues got observed etc., we need to investigate the object and methods which causes problem in the device. By method profiling in the device we can figure out the root cause of the higher launching response time and other problems.

3.2.2. Native Mobile Application Launching Time

Native Mobile Application Launch Time is the time that a native mobile application takes to get launched in the device. So it should be as less as possible. Higher the launch time results more battery consumption and also it may give a negative impression to users!

3.2.3. Mobile Base Line Testing

Mobile Baseline Testing is the process of identifying the “Behaviour” of the mobile application with a minimal
user load at servers. Each MPTS is executed with single virtual user. The test is executed for a very small period of time under different mobile N/W (2G, 3G, 4G & WiFi) conditions. This test will be used as a benchmark for further Mobile Load Tests.

3.2.4. Mobile Load Testing

Mobile Load Testing is the process of identifying the “Behaviour” of the mobile application with a heavy user load at servers. High volume of requests with each MPTSs are sent through different mobile N/W (2G, 3G, 4G & WiFi) to servers and investigate the behaviour of the servers during this phase.

3.2.5. Mobile Endurance Testing

Mobile Endurance Testing is the process of identifying the “Memory Leak” in the servers. High volume of requests with each MPTS is sent through different mobile N/W (2G, 3G, 4G & WiFi) to servers and run the test for longer hours to investigate the reliability of the application.

3.2.6. Server Side Monitoring

We need to monitor and analyze the following things (not limited to the list but these are most important!) at server end during the Mobile Baseline, Mobile Load & Mobile Endurance test executions:

1. Application & Database server CPU utilization
2. Application & Database server memory utilization
3. Application Server JVM heap memory utilization
4. Application Server Connection Pool Sizing
5. Application Server Thread Pool Sizing
7. Network bandwidth utilization between servers.
8. Total number of transactions per hour at database
9. Total number of Java operations per hour at JVM, etc.

3.3. Native Mobile Application Performance Test Execution Workarounds

Designing a Native Mobile Application Performance Test Execution framework is a complex task and need proper approach to execute this. We will have to monitor and investigate the health of the servers during heavy load under different N/W conditions (2G, 3G, 4G and WiFi). The heavy load will be deployed by different combinations of simulated mobile users (such as Android, iOS and Windows mobile users). We will also have to monitor and investigate the health of the mobile devices when the native mobile application is running in the devices.

The below Figure 4 demonstrates the device side execution and monitoring.

![Figure 4: Native Mobile Application Performance Engineering Test Execution Phases at Device Side](image1)

The below Figure 5 demonstrates the server side execution and monitoring phases.

![Figure 5: Native Mobile Application Performance Engineering Test Execution Phases at Server Side](image2)

3.3.1. On-Device Performance Test Execution Process

1. Develop NMAPTSs (Native Mobile Application Performance Test Scripts) for each business scenarios.
2. Do execute three to five continuous iterations with each NMAPTS on the mobile devices and monitor the mentioned counters. In this case we can consider
single or multiple N/W based on the time line and requirements.

3.3.2. Mobile Baseline, Load & Endurance Test Execution Process

For Mobile Baseline Test:
1. Simulate the N/W conditions (2G, 3G, 4G or WiFi) with each test run in the load test tool.
2. Select the mobile OS (Android, iOS etc.) specific MAPTSs (Mobile Application Performance Test Scripts)
3. Execute the test with single VUser per MAPTS using load test tool under different N/W conditions.
4. Monitor all the servers during execution.

For Mobile Load Test:
1. Simulate the N/W conditions (2G, 3G, 4G or WiFi) with each test run in the load test tool.
2. Select the mobile OS (Android, iOS etc.) specific MAPTSs
3. Execute the test with concurrent VUsers load using load test tool for a specific duration (as per requirement) with each set of OS specific MAPTS under different N/W conditions.
4. Monitor all the servers during executions.

For Mobile Endurance Test:
1. Simulate the N/W conditions (2G, 3G, 4G or WiFi) with each test run in the load test tool.
2. Select the mobile OS (Android, iOS etc.) specific MAPTSs
3. Execute the test with concurrent VUsers load using load test tool for longer duration (based on requirements but at least more than 8 Hrs.) with each set of OS specific MAPTS under different N/W conditions.
4. Monitor all the servers during executions.

3.3.3. Network Considerations

We have to run separate tests with each different N/W conditions (2G, 3G, 4G & WiFi).

4. Experimental Case Study

Here I am presenting a case study for native mobile application performance engineering with Android mobile device which I prepared for one of our esteemed client. This case study will demonstrate the strategy, implementation and execution of the proposed performance engineering solution.

4.1. Experiment Set up

Table 2: Experiment Setup

<table>
<thead>
<tr>
<th>Client Side Setup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Device: Samsung S4 [7]</td>
<td></td>
</tr>
<tr>
<td>OS: Android</td>
<td></td>
</tr>
<tr>
<td>Tools Used: For CPU, Memory &amp; Network Utilization, Battery Utilization: Little Eye [8]</td>
<td></td>
</tr>
<tr>
<td>Scope In: CUP Utilization, Memory Utilization, Network Utilization, Application Launch Time Capturing, Method Profiling</td>
<td></td>
</tr>
<tr>
<td>Network: WiFi</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Server Side Setup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Server: IBM WAS 7.0</td>
<td></td>
</tr>
<tr>
<td>Database Server: Oracle 11g</td>
<td></td>
</tr>
<tr>
<td>OS: Windows 2008 R, UNIX</td>
<td></td>
</tr>
<tr>
<td>Tools Used: For Load Test: Neoload [10]</td>
<td></td>
</tr>
<tr>
<td>For Application Server Side Diagnostics: WAS Internal Tools</td>
<td></td>
</tr>
<tr>
<td>For Database Server Side Diagnostics: Oracle AWR Report</td>
<td></td>
</tr>
<tr>
<td>Test Type: Load Test with 200 Virtual Users</td>
<td></td>
</tr>
<tr>
<td>Network: AT&amp;T, Verizon and WiFi</td>
<td></td>
</tr>
</tbody>
</table>

4.2. On-Device Performance Statistics

During the on-device test execution with android mobile device, the health of the native mobile application has been monitored and investigated. CPU, Memory, Data In/Out through Network and Battery utilizations have been analyzed for each NMATS.

4.2.1. CPU Utilization

- CPU Utilization by Native Mobile Application
This is the total CPU utilized by the native mobile application in the device during application view navigation. The Figure 6 shows the CPU utilisations graph for each NMPATS execution.

![Figure 6: CPU Utilization by Native Mobile Application](image)

4.2.2. Memory Utilization
- Total Memory Utilization by Device

The total memory consumption by the device during native application view navigation. The Figure 7 shows the memory utilisations graph for each NMPATS execution.

![Figure 7: Total Memory Utilization by Device](image)

- Total Memory Utilization by Native Mobile Application

This is the total memory utilization by the native mobile application during application view navigation in the device. The Figure 8 shows utilizations graph for each NMPATS execution.

![Figure 8: Memory Utilization by Native Mobile Application](image)

- Memory Utilization by Android Application Methods

This is the memory utilization by the android application methods during application view navigation in the device. The Figure 9 shows the utilizations graph for each NMPATS execution.

![Figure 9: Memory Utilization by Android Application Methods](image)

- Memory Utilization by Android Dalvik

This is the memory consumption by the android Dalvik during application view navigation in the device. The Figure 10 shows utilizations graph for each NMPATS execution.

![Figure 10: Memory Utilization by Dalvik](image)

- Garbage Collection Details in Device during Execution
The following below Table 3 shows the GC statistics for each business NMAPTS during the execution.

### Table 3: GC Statistics during Execution in Device

<table>
<thead>
<tr>
<th>Business Flows</th>
<th>Application Running Time</th>
<th>Application Pause Time</th>
<th>Number of Concurrent GC's</th>
<th>Device Dalvik Threshold (MB)</th>
<th>Number Of Allocated GC's</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account Statement</td>
<td>133.66</td>
<td>0.49</td>
<td>16</td>
<td>64</td>
<td>12</td>
</tr>
<tr>
<td>Recent Activity</td>
<td>147.58</td>
<td>0.53</td>
<td>15</td>
<td>64</td>
<td>13</td>
</tr>
<tr>
<td>Make A Payment</td>
<td>155.57</td>
<td>0.58</td>
<td>16</td>
<td>64</td>
<td>14</td>
</tr>
<tr>
<td>Send Money</td>
<td>177.73</td>
<td>0.49</td>
<td>16</td>
<td>64</td>
<td>9</td>
</tr>
<tr>
<td>Redeem Card</td>
<td>79.73</td>
<td>0.41</td>
<td>14</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>Redemption history</td>
<td>187.34</td>
<td>0.91</td>
<td>30</td>
<td>64</td>
<td>21</td>
</tr>
<tr>
<td>Quick View</td>
<td>267.98</td>
<td>0.39</td>
<td>44</td>
<td>64</td>
<td>32</td>
</tr>
</tbody>
</table>

4.2.3. **Network Utilization**

- **Network Data Sent by Native Mobile Application**

The data transferred through n/w by the native mobile application during application view navigation. The Figure 11 shows the utilization graph.

![Figure 11: Network Data Sent by Native Mobile Application](image)

- **Network Data Received by Native Mobile Application**

The data received through n/w by the native mobile application during application view navigation. The Figure 12 shows the graph for the utilization.

![Figure 12: Network Data Received by Native Mobile Application](image)

4.2.4. **Battery Utilization**

- **Power Consumptions by CPU**

The power consumed by the CPU during native mobile application view navigation. The Figure 13 shows the graph for the utilization.

![Figure 13: Power Consumptions by CPU during On-Device Native Mobile Application Execution](image)

- **Power Consumptions during Application Data Usages**

The power consumed by the native mobile application during native mobile application view navigation. The Figure 14 shows the graph for the utilization.

![Figure 14: Power Consumptions during Application Data Usages](image)
Figure 14: Power Consumptions during Application Data Usages

- **Power Consumptions by Display**

The power consumed by the display panel during native mobile application view navigation. The Figure 15 shows the graph for the utilization.

Figure 15: Power Consumptions by Display during On-Device Native Mobile Application Execution

### 4.2.5. Application Launch Time

This is the average time taken by the native mobile application to get launched in the mobile device. The following Table 3 shows the application launch times under different N/W conditions. The average application launch times of 10 test iterations run has been taken for analysis.

Table 3: Native Mobile Application Launch Time

<table>
<thead>
<tr>
<th>Launch Times</th>
<th>AT&amp;T</th>
<th>WiFi</th>
<th>Verizon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Average Launch Time Captured (Sec)</td>
<td>5.8</td>
<td>6.9</td>
<td>7.9</td>
</tr>
<tr>
<td>Highest Average Launch Time Captured for each NMAPTS (Sec)</td>
<td>10.3</td>
<td>12.2</td>
<td>11.1</td>
</tr>
<tr>
<td>Highest Average Launch Time Captured (Sec)</td>
<td>67.2</td>
<td>73</td>
<td>66.6</td>
</tr>
</tbody>
</table>

### 4.3. Server Side Performance Statistics

We have executed Mobile load tests with three different N/W conditions. The following subsections will describe the monitoring and investigated results. [12]

#### 4.3.1. Native Application View Response Time

![Figure 16: Mobile Native Application Views Response Time with Different N/W Conditions](image)

Table 4: Native Mobile Application View Response Times

<table>
<thead>
<tr>
<th>Transactions (Containers)</th>
<th>Web AT&amp;T 5G N/W</th>
<th>Web WiFi N/W</th>
<th>Web Verizon N/W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Avg</td>
<td>90%</td>
</tr>
<tr>
<td>Search</td>
<td>6.1</td>
<td>6.9</td>
<td>7.3</td>
</tr>
<tr>
<td>IC-Log</td>
<td>3.1</td>
<td>3.5</td>
<td>3.8</td>
</tr>
<tr>
<td>Log</td>
<td>3.1</td>
<td>3.5</td>
<td>3.8</td>
</tr>
<tr>
<td>Event</td>
<td>6.4</td>
<td>6.9</td>
<td>7.3</td>
</tr>
<tr>
<td>Account Statement</td>
<td>7.2</td>
<td>7.6</td>
<td>8.1</td>
</tr>
<tr>
<td>Account Statement</td>
<td>5.3</td>
<td>5.7</td>
<td>6.1</td>
</tr>
<tr>
<td>Sign</td>
<td>5.3</td>
<td>5.7</td>
<td>6.1</td>
</tr>
<tr>
<td>Login</td>
<td>5.3</td>
<td>5.7</td>
<td>6.1</td>
</tr>
<tr>
<td>Register</td>
<td>5.3</td>
<td>5.7</td>
<td>6.1</td>
</tr>
<tr>
<td>Forgot Password</td>
<td>5.3</td>
<td>5.7</td>
<td>6.1</td>
</tr>
</tbody>
</table>

#### 4.3.2. 90th Percentile

The 90th percentile is the value (or score) below which 90 percent of the response time observations may be found. This analysis explores whether the application’s response time SLA is getting met 90% of time during mobile load test. The Figure 17 shows the 90th percentile values for all the transactions when the load test run with different N/W conditions.
4.3.3. Standard Deviation (SD)

SD analysis shows how reliable your application is. The lower the standard deviation, the more consistent the response times are, so more reliable will be the application. Here in this case study we found that the SD for most of the transactions is less than the half of the mean values, so the system indicates the high probabilities of reliability.

4.3.4. Server Side CPU Utilization

The Figure 18 shows the CPU utilization of the servers during mobile load test with different N/W conditions.

4.3.5. Server Side CPU Queue Length

The Figure 19 shows the application server side CPU Queue length statistics during mobile load testing.

4.3.6. Application Server Memory Utilization

The Figure 20 shows the memory utilization of the servers during mobile load test with different N/W conditions.

4.3.7. JVM Heap Utilization

Table 5: Heap Memory GC Details for Primary Application Server

<table>
<thead>
<tr>
<th>Summary for App Server-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forced collection count</td>
</tr>
<tr>
<td>Full collections</td>
</tr>
<tr>
<td>Mean garbage collection pause (ms)</td>
</tr>
<tr>
<td>Mean interval between collections (minutes)</td>
</tr>
<tr>
<td>Number of collections triggered by allocation failure</td>
</tr>
<tr>
<td>Proportion of time spent in garbage collection pauses (%)</td>
</tr>
<tr>
<td>Proportion of time spent unpaused (%)</td>
</tr>
<tr>
<td>Rate of garbage collection (MB/minutes)</td>
</tr>
</tbody>
</table>
Table 6: Used Heap after GC Collection in Primary Application Server

<table>
<thead>
<tr>
<th>Mean Heap (MB)</th>
<th>Minimum Heap (MB)</th>
<th>Maximum Heap (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>550</td>
<td>315</td>
<td>913</td>
</tr>
</tbody>
</table>

Figure 21: Heap Memory Utilization Trend in Primary Application Servers during Load Test with Different Mobile N/W Conditions

Table 7: Heap Memory GC Details for Secondary Application Server

<table>
<thead>
<tr>
<th>Summary for App Server-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forced collection count</td>
</tr>
<tr>
<td>Full collections</td>
</tr>
<tr>
<td>Mean garbage collection pause (ms)</td>
</tr>
<tr>
<td>Mean interval between collections (minutes)</td>
</tr>
<tr>
<td>Number of collections triggered by allocation failure</td>
</tr>
<tr>
<td>Proportion of time spent in garbage collection pauses (%)</td>
</tr>
<tr>
<td>Proportion of time spent unpaused (%)</td>
</tr>
<tr>
<td>Rate of garbage collection (MB/minutes)</td>
</tr>
</tbody>
</table>

Table 8: Used Heap after GC Collection in Secondary Application Server

<table>
<thead>
<tr>
<th>Mean Heap (MB)</th>
<th>Minimum Heap (MB)</th>
<th>Maximum Heap (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>575</td>
<td>332</td>
<td>914</td>
</tr>
</tbody>
</table>

Figure 22: Heap Memory Utilization Trend in Secondary Application Servers during Load Test with Different Mobile N/W Conditions

5. Observation & Analysis

Here some of the most important observations have been posted with analysis.

5.1. Client Side

1. The native mobile application’s average launch time is high in the device. We can see from Table 3 that the application launch time in the device is very high. This is a serious concern and need to be investigated and come up with a solution to optimize the application launch time in android device.

2. We can see clearly continuous increasing pattern of memory utilization in the memory graphs. So this may be the case of memory leaking. So the probability of crashing of application is high due to the pattern of memory leaking. Recommending for optimizing the application in terms of memory utilization. Each time a new activity was loaded; the application downloaded some data, and forgot to clear it when the user returned to the home screen to start the iteration again, resulting in the leak.
3. During some NMAPTS execution, the CPU utilization is high. High CPU utilization may cause more battery consumption, so by proper optimization of the application the CPU utilization can be reduced to some level. This will save battery power.

4. It is also noted by analysing the graph “Power Consumption by Display during Application View Navigation” that power consumption is high. The power consumption can be reduced if low intensity or dark colour images are used. More vibrant colour images consume more battery power.

5.2. Server Side

1. The average response time for the following native mobile application view

   \textit{OfferDeal->Extras}

   is more than 9 seconds on average with all N/W conditions. The expected average response time of the native mobile application view is not more than 5 seconds.

2. The heap utilization graphs show high heap memory utilization for both the application servers. Most of the time threshold value is getting touched. Here it is being recommended to increase the heap memory size and also optimize the GC cycles frequencies.

3. By analysing memory utilization graphs we found that there are probabilistic patterns of memory leak in both the JVM of the application server machines.

6. Some Tuning Recommendations for Native Mobile Applications

1. Try to optimize pre-fetch data handling mechanism.
2. Try to spread out the computationally intensive jobs.
Appendix B

No of Figures: 22
No of Tables: 8

References

[1]. Smart Mobs: The Next Social Revolution by Howard Rheingold


[4]. Pro Android Scripting with SL4A: Writing Android Native Apps Using Python, Lua, and Beashell by Paul Ferrill

[5]. Building Hybrid Android Applications with Java and JavaScript by Nizamettin Gok and Nitin Khanna (9 August 2013)


[8]. http://www.littleeye.co/


About the Author

Arit Kumar Bishwas, having a MS degree from BITS, Pilani. He is currently working with Capgemini India Pvt. Ltd. as a Performance Engineering Consultant. He is having 9+ years of total work experience out of which he has about 7 years of Software Industry experience and about 2+ years of computer science teaching experience. He is Sun Certified Java Programmer (SCJP 1.5) as well as ISTQB certified. Earlier he has worked with IBM India Pvt Ltd., Mindtree Ltd. and Zensar Technologies Ltd. In 2010, his paper was selected as one of the top ten best papers at STC.