Performance Comparison of Android Messengers

Rama Bhatia *rmabhatia@gmail.com*
Department of Computer Science and Engineering
JaganNath University
Jaipur, India
Lecturer, School of Engineering and Technology,
PSB Academy, Singapore,

Renu Bagoria *renu.bagoria@jagannathuniversity.org*
Department of Computer Science and Engineering
JaganNath University
Jaipur, India

Abstract

The increasing demand of Android applications compels the developers to think and develop applications with efficient use of memory usage, CPU utilisation and UI rendering speed. The literature survey reveals that very few works have been reported for measuring the performance of various android applications. In this paper the performance analysis of most of the popular mobile applications have been carried over using Eclipse with SDK Tools and Android Virtual device. CPU utilisation, Memory usage and User Interface (UI) rendering speed have been considered as the performance metric. The performance of most commonly used apps like WhatsApp, Viber, WeChat & Imo have been analysed. The experimental result shows that

(i) Utilisation of CPU in case of Wechat is more as compared with WhatsApp, Viber, & Imo.
(ii) In case of Memory Usage, java.lang.string class found as a major problem suspect for memory leak problem.
(iii) And for Rendering Speed in case of viber is taking lesser time as compared with WhatsApp,
Viber, & Imo .

Keywords: Android App, WhatsApp, WeChat, Viber , Imo, Performance Analysis.

1. INTRODUCTION

Due to the increasing demand of Android devices and the various mobile applications, the performance assessment of various android apps is required. Mostly, Android apps are easily downloadable from GooglePlay [17] and the no of apps are increasing significantly. WhatsApp, WeChat, Imo and Viber are the most popular messaging apps in present days. According to data from statista (April 2016), there are around 1,000 million monthly active users in WhatsApp and is the most popular messaging app while Viber, Line, WeChat and Imo are the other popular messaging apps. The key to success of any messaging apps is depending on many performance factors[1]. From developers point of view, a good messaging-app is that which effectively utilises CPU, memory and consumes less energy. This motivate us to further asses the performance efficiency of the mobile chat messenger applications in terms of resource utilization.

2. RELATED WORK

Liu Pu[19](2009) discussed the Short Message Service Architecture and their services. **Anthony Gutierrez et al [20] (2011)** developed BBench to assess web-browser's rendering performance. **Dhinakaran Pandiyan et al.[21](2013)** developed a mobile platform benchmark suite, MobileBench and presented the performance and energy characterizations for Mobile Bench.

Thiago Soares Fernandes et al.[18](2014) evaluated the CPU and memory of some android application. **Linares- Vasquez et al. [2](2015)** studied the various practices and tools used by Android developers to detect and fix the performance. Moreover, it was found that for performance assessment of CPU Traceview[7] is the top most used tool followed by MAT and then DDMS, Systrace and Logcat are taking the next place [2]. Many others are also suggesting to use different SDK tools such as DDMS[4], MAT[3][5], Systrace[6][7], Hierarchy Viewer[8] for performance analysis on different parameters of android apps [9]. In this paper, the performance of WhatsApp, WeChat, Viber and Imo(the top most used android mobile chat messengers[9])has been analysed using the Profiling Software - Eclipse with SDK Tools (the known tools for performance analysis [2][3][5][8][6][7][8])and Android Virtual device(Nexsus Emulator). The experimental results provide the resource utilisation of various modules of the messaging apps.

Comparative Research work is published [22].

3. ANALYSIS CRITERIA AND MEASURING TOOLS

This section first gives you an overview of the parameters that are computed and the rest section continues with the tools and methods used to perform the calculation are given.

3.1 Analysis Criteria

This section focussed the methods and techniques used for evaluating the performance. All the experiments are done on the standard APIs of the mentioned messengers on android platforms [13].

3.1.1 CPU Utilization- If an application takes more CPU time then it will impact adversely on other processes running on the device. In the proposed work, the following four common modules have been taken for measuring the CPU utilisation time as these are the commonly used modules in all the apps.

(i) android.os.handler.dispatchMessage
(ii) android.os.handler.handleCallback
(iii) android.view.View.layout
(iv) android.view.ViewGroup.layout

Initially the evaluation has been done by restricting the no of contacts to three and finally the evaluation is done by increasing the contacts to thirty.
During the initial stage of the application i.e. when the application initially started, we send the messages and use the call procedure within 90,000 msec. This is the common evaluation criteria used in all the applications.

3.1.2 Memory Utilization- Memory utilization is the total usage of memory used by the processes which Includes allocated Heap, Free memory and percentage of used memory. Memory leakage is the one of the major constraint which degrades the performance of any application. Hence monitoring of memory utilization is one of the important step for the developers to check the instability of the application due to the memory leakage. In addition to this, garbage collection is also an important criteria for efficient use of memory. This parameter is evaluated using repeated Garbage Collection (GC) calls during application launching state.

3.1.3 UI Rendering Speed- Good UI of any application is one of the key parameter for success of any application. The bad design of the application layout interrupts the loading speed of the application. To evaluate this parameter, the following four criteria were considered:

(i) No. of views of the application
(ii) Rendering speed parameters like Measure time(ms), Layout time (ms) and Draw time(ms) [8].

3.2 Measuring Tools

In the proposed work, Eclipse is used extensively for evaluation of all the parameters. CPU utilisation has been measured by using Traceview whereas for Memory measurement, Eclipse

Memory Analyzer (MAT), Heap and HPROF has been used. Hierarchy Viewer has been used for measuring UI rendering speed. The advantage of using Traceview for measuring the CPU is that it gives an graphical presentation of CPU utilisation time [15]. Similarly the advantage of Eclipse Memory Analyzer (MAT) is that it gives a detailed analysis of used memory and memory leaks[16]. In addition to this it also generates the memory leak suspects report. For measurements of UI rendering speed, Hierarchy Viewer[11]is selected as it visualize the nested behaviour of screen views and the tool also finds the flaws in design layout. In addition to the above mentioned tools, Nexsus_S, Android 6 has been used as an emulator.

4. RESULTS AND DISCUSSION

In this section, the quantitative assessment of the mentioned messenging applications with respect to CPU Utilization, Memory usage and Rendering Speed have been presented.

4.1 CPU Utilization

In this paper, the CPU Utilization of different modules is measured on the basis of load factors (α) where α represents number of contacts in an application.

The following modules have been considered:

4.1.1 android.os.handler.dispatchMessage

	α =3		α =30	
Application	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**
WhatsApp	410	65.1	900	59.2
Viber	320	61.5	870	45.8
WeChat	10290	70.3	10020	70.2
Imo	6520	68.1	6000	70.5

TABLE 1: CPU Utilization of android.os.handler.dispatchMessage.

From the above table 1 it is observed CPU Utilization in case of Wechat is more as compared to WhatsApp, Viber and Imo.

4.1.2 android.os.handler.handleCallback

	α =3		α =30	
Application	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**
WhatsApp	290	46	500	32.9
Viber	210	40.04	570	30
WeChat	13130	65.6	9270	64.9
Imo	5490	57.3	4760	55.9

TABLE 2: CPU Utilization of android.os.handler.handleCallback.

From the above table 2, it is observed that in case of Callback module, CPU Utilization in case of WeChat is more as compared to WhatsApp, Viber and Imo.

4.1.3 android.view.View.layout

	α =3		α =30	
Application	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**
WhatsApp	30	4.8	30	4.8
Viber	70	13.5	110	5.8
WeChat	1790	8.9	2120	14.8
Imo	1390	14.5	2610	30.7

TABLE 3: CPU Utilization of android.view.View.layout.

From the above table 3, it is observed that in case of View layout module WeChat CPU Utilization is more as compared to WhatsApp, Viber and Imo.

4.1.4 android.view.ViewGroup.layout

	α =3		α =30	
Application	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**	**Inclusive CPU Time (ms)**	**Inclusive CPU Time%**
WhatsApp	30	4.8	30	4.8
Viber	70	13.5	110	5.8
WeChat	1790	8.9	2120	14.8
Imo	1390	14.5	2610	30.7

TABLE 4: CPU Utilization of android.view.ViewGroup.layout.

From the above table 4, it is observed that in case of less load CPU Utilization of Wechat is more as compared to WhatsApp, Viber and Imo whereas WhatsApp is taking the lesser time and in case of more load Imo is taking the maximum time.

Hence it is concluded that Viber and WhatsApp are better in case of CPU Utilization factor whereas WeChat and Imo are wasting the resource by more utilizing CPU.

4.2 Memory Utilization

The table data shows that all the applications before started occupy 75% of the total allocated heap size.

ID	**Application**	**Heap Size (MB)**	**Allocated(MB)**	**Free(MB)**	**%used**	**#Objects**
1956	WhatsApp	10.902	6.902	4	63.31%	79577
1845	Viber	10.992	6.992	4	63.61%	48814
914	WeChat	14.914	10.914	4	73.18%	69901
1283	Imo	6.536	3.921	2.614	60.00%	39218

TABLE 5: Memory Analysis after launching of the mentioned apps.

4.2.1 WhatsApp Memory Leak Suspects

Below are the major Problem Suspects and their Biggest instances found during WhatsApp

Launching:-

a) Problem Suspect 1
55,804 instances of **"java.lang.String"**, loaded by **"<system class loader>"** occupy **4,834,912 (42.21%)** bytes.

b) Problem Suspect 2
5,109 instances of **"java.lang.Class"**, loaded by **"<system class loader>"** occupy **3,261,968 (28.48%)** bytes.

Biggest instances measured in bytes are:

i. class com.whatsapp.App - 880,056 (7.68%)
ii. class com.whatsapp.fu - 246,520 (2.15%)
iii. class android.text.Html$HtmlParser - 126,616 (1.11%)
iv. class android.icu.impl.UCharacterProperty - 117,536 (1.03%)

c) Problem Suspect 3
14 instances of **"java.lang.DexCache"**, loaded by **"<system class loader>"** occupy **1,704,672 (14.88%)** bytes.

Biggest instances measured in bytes are:

i. java.lang.DexCache - 547,136 (4.78%)
ii. java.lang.DexCache - 291,632 (2.55%)
iii. java.lang.DexCache - 253,176 (2.21%)
iv. java.lang.DexCache - 231,944 (2.02%)

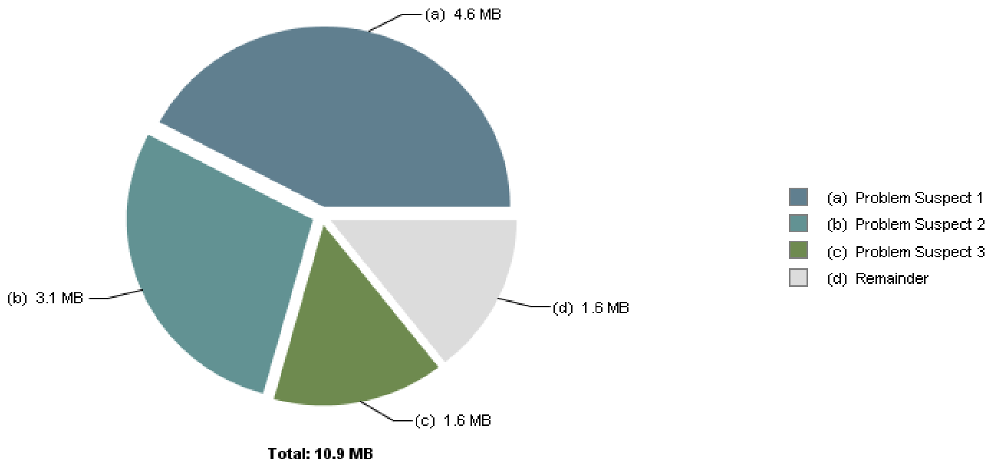

FIGURE 1: Whats App Memory Leak Problem Suspects.

4.2.2 Viber Memory Leak Suspects
Below are the major Problem Suspects and their Biggest instances found during the Launch of Viber:-

a) Problem Suspect 1
48,167 instances of **"java.lang.String"**, loaded by **"<system class loader>"** occupy **4,191,368 (32.72%)** bytes.

b) Problem Suspect 2
15 instances of **"java.lang.DexCache"**, loaded by **"<system class loader>"** occupy **1,865,432 (14.56%)** bytes.

Biggest instances measured in bytes are:

i. java.lang.DexCache - 550,168 (4.30%)
ii. java.lang.DexCache - 294,176 (2.30%)
iii. java.lang.DexCache - 253,176 (1.98%)
iv. java.lang.DexCache - 231,672 (1.81%)
v. java.lang.DexCache - 141,360 (1.10%)

c) Problem Suspect 3
6,255 instances of **"java.lang.Class"**, loaded by **"<system class loader>"** occupy **1,619,808 (12.65%)** bytes.

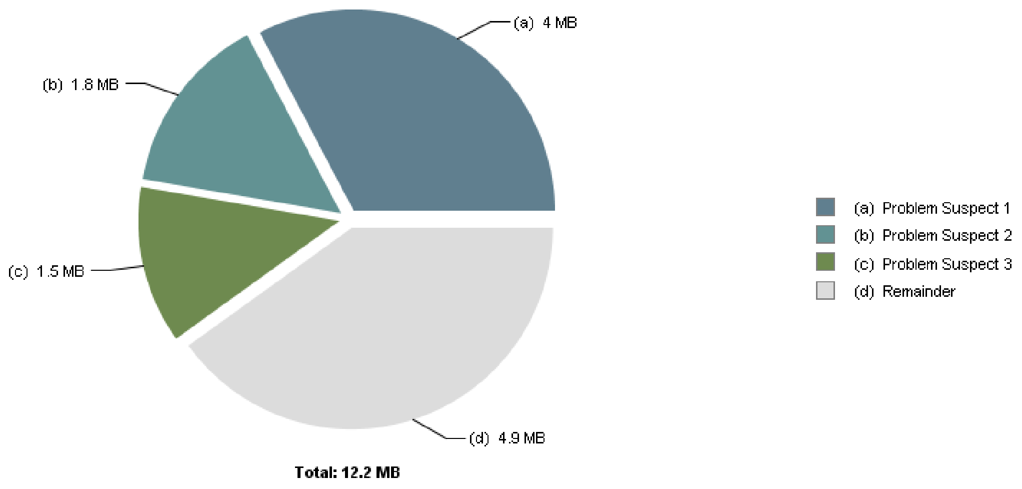

FIGURE 2: Viber Memory Leak Problem Suspects.

4.2.3 Wechat Leak Suspects
Below are the major Problem Suspects and their Biggest instances found during the Launch of Wechat:-

a) Problem Suspect 1
50,960 instances of **"java.lang.String"**, loaded by **"<system class loader>"** occupy **4,493,696 (30.69%)** bytes.

b) Problem Suspect 2
9,316 instances of **"java.lang.Class"**, loaded by **"<system class loader>"** occupy **3,611,368 (24.67%)** bytes.

Biggest instances are:

i. class com.tencent.mm.av.b- 1,423,864 (9.73%) bytes.

c) Problem Suspect 3
15 instances of **"java.lang.DexCache"**, loaded by **"<system class loader>"** occupy **2,447,248 (16.71%)** bytes.

Biggest instances measured in bytes are:

i. java.lang.DexCache- 584,960 (4.00%)
ii. java.lang.DexCache- 547,088 (3.74%)
iii. java.lang.DexCache- 405,640 (2.77%)
iv. java.lang.DexCache- 253,888 (1.73%)

v. java.lang.DexCache- 231,744 (1.58%)

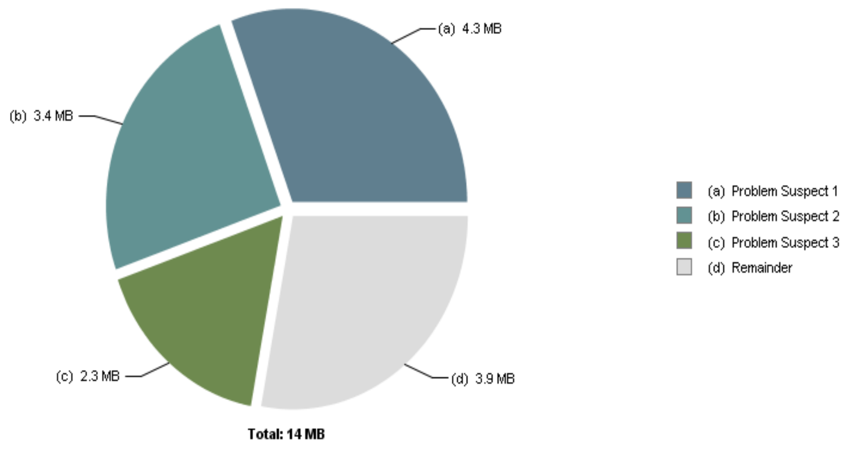

FIGURE 3: Wechat Memory Leak Problem Suspects.

4.2.4 Imo Leak Suspects

Below are the major Problem Suspects and their Biggest instances found during the Launch of Wechat:-

a) **Problem Suspect 1**
46,764 instances of **"java.lang.String"**, loaded by **"<system class loader>"** occupy **4,078,880 (37.88%)** bytes.

b) **Problem Suspect 2**
16 instances of **"java.lang.DexCache"**, loaded by **"<system class loader>"** occupy **1,845,632 (17.14%)** bytes.

Biggest instances are measured in bytes:

i. java.lang.DexCache (5.08%)
ii. java.lang.DexCache- 316,048 (2.94%)
iii. java.lang.DexCache- 252,992 (2.35%)
iv. java.lang.DexCache- 231,632 (2.15%)
v. java.lang.DexCache- 141,320 (1.31%)

c) **Problem Suspect 3**
5,085 instances of **"java.lang.Class"**, loaded by **"<system class loader>"** occupy **1,534,144 (14.25%)** bytes.

Biggest instances are:

i. class android.text.Html$HtmlParser- 126,616 (1.18%) bytes.
ii. class android.icu.impl.UCharacterProperty- 117,536 (1.09%) bytes.

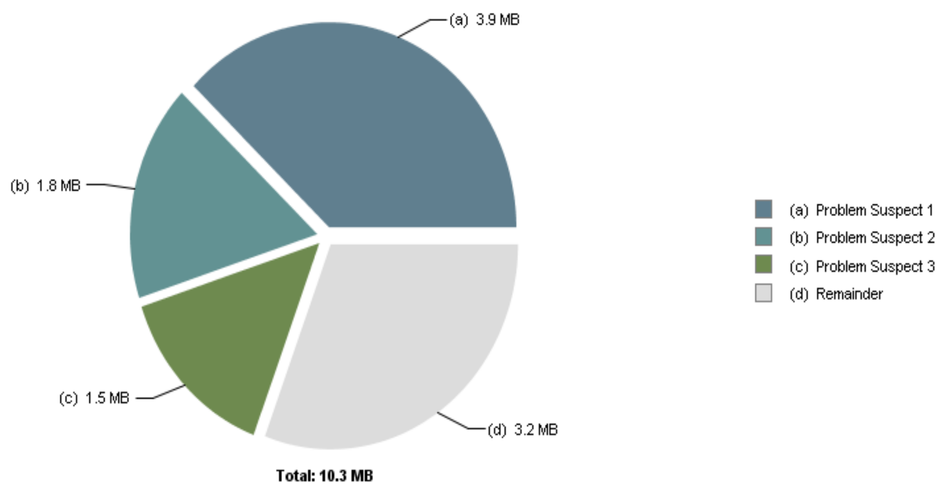

FIGURE 4: Imo Memory Leak Problem Suspects.

4.2.5 Comparative Analysis

The following table shows that java.lang.String class is utilizing more memory in all the applications. Hence it is concluded that java.lang.String class need to be efficiently used to counter memory leak problem.

Retained Heap - Table	Shallow Heap #3 (Imo)	Shallow Heap #2 (Wechat)	Shallow Heap #1 (Viber)	Shallow Heap #0 (Whats App)	Objects #3 (Imo)	Objects #2 (Wechat)	Objects #1 (Viber)	Objects #0 (WhatsA pp)	Class Name
4722032	3668856	4115640	3675712	4722032	54112	60299	54569	72431	char[]
1871312	1353792	2724840	3152000	1871312	2081	2938	2483	2597	byte[]
6359416	1296648	1444896	1307736	1736232	54027	60204	54489	72343	java.lang.String
1651528	1680880	2245336	1808824	1651528	2791	4235	3077	3646	int[]
2020280	1323528	1657768	1303144	1321000	1598	1869	1571	2855	java.lang.String[]
3388288	117096	192096	121208	127304	5085	9316	6255	5469	java.lang.Class

TABLE 6: Comparison Table of Memory Analysis.

4.3 UI Rendering Speed

The UI rendering speed comparison is done on the basis of the results obtained by three factors i.e Measure, Layout & Draw. Profiling has been done to understand the rendering speed closely. The table 7 shows that viber is taking lesser time time than rest 3 applications.

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	No. of Views	Application
0	0	0	0	364	hatsApp
0	0	0	0	204	Viber
100	70	10	20	498	WeChat
150	70	70	10	186	Imo

TABLE 7: Comparison Table of of U I Rendering(Profile1).

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	No. Of Views	Application
20	0	20	0	364	WhatsApp
10	10	0	0	204	Viber
160	110	30	20	498	WeChat
110	60	40	10	186	Imo

TABLE 8: Comparison Table of of U I Rendering(Profile2).

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	No. Of Views	Application
0	0	0	0	364	WhatsApp
0	0	0	0	204	Viber
140	80	40	20	498	WeChat
130	70	50	10	186	Imo

TABLE 9: Comparison Table of of U I Rendering(Profile3).

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	No. Of Views	Application
10	0	10	0	364	WhatsApp
10	0	10	0	204	Viber
160	100	40	20	498	WeChat
120	70	50	0	186	Imo

TABLE 10: Comparison Table of of U I Rendering(Profile4).

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	No. Of Views	Application
20	10	10	0	364	WhatsApp
0	0	0	0	204	Viber
140	80	40	20	498	WeChat
100	50	40	10	186	Imo

TABLE 11: Comparison Table of of U I Rendering(Profile 5).

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	No. Of Views	Application
0	0	0	0	364	WhatsApp
10	0	0	10	204	Viber
140	80	40	20	498	WeChat
100	50	40	10	186	Imo

TABLE 12: Comparison Table of of U I Rendering(Profile6).

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	No. Of Views	Application
20	0	20	0	364	WhatsApp
20	10	0	10	204	Viber
140	80	40	20	498	WeChat
100	50	40	10	186	Imo

TABLE 13: Comparison Table of of U I Rendering(Profile7).

Total(ms)	Draw(ms)	Layout(ms)	Measure(ms)	Application
10	1.4286	8.5714	0	WhatsApp
7.1429	2.8571	1.4286	2.8571	Viber
140	85.714	34.286	20	WeChat
115.71	60	47.143	8.5714	Imo

TABLE 14: Comparison Table of Average Total of all the Rendering Profiles.

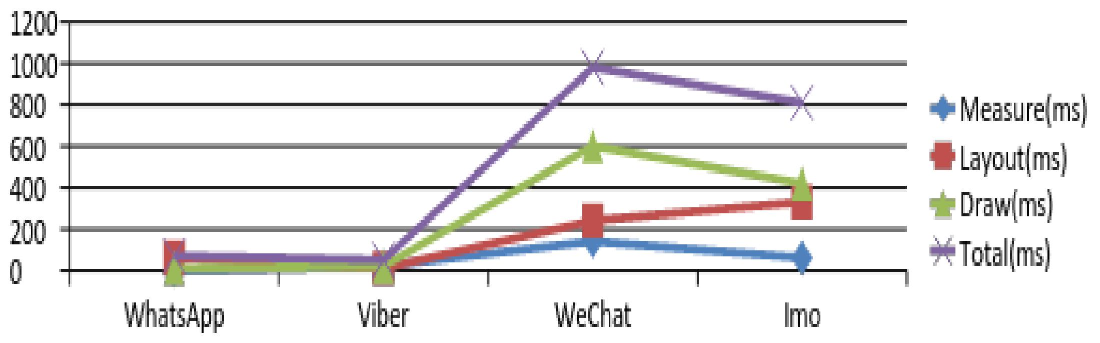

FIGURE 5: Graphical view of Grand Total of all the Profiles.

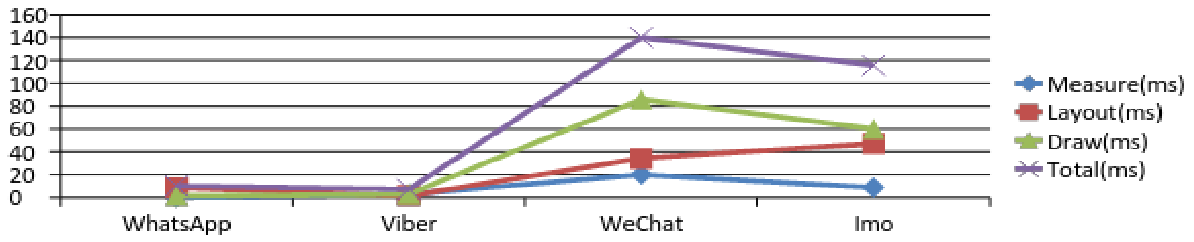

FIGURE 6: Graphical view of Table No. 17.

From the above Profile Tables and Figures , it has been shown that the rendering speed of the Viber is less as compare to the rest of mentioned applications.

5. CONCLUSION

In this Paper, We have analysed the performance analysis of popular used apps like WhatApp, WeChat, Imo and Viber. The performance of these apps have been evaluated in terms of CPU, Memory & UI Rendering Speed. Experimental results yield that CPU utilization of WhatsApp and Viber is better than others.

6. FUTURE SCOPE

The Study opens the window for android researchers to work on various Performance parameters and it is recommended to analyze more modules in case of CPU Utilization. For Memory assessment it is suggested to find out the shallow size results.

7. REFERENCES

[1] Lars Vogel. “Android application (performance and more) analysis tools” http://www.vogella.com/tutorials/AndroidTools/article.html, July 5 , 2016[12-08-2016].

[2] Of Bytes and Battery. ”Of Bytes, Cycles and Battery Life”, http://www.slideshare.net/mariozechner5/of-bytes-cycles-and-battery-life. Oct. 26, 2013[12-08-2016].

[3] Patrick Dubroy. ”Memory Analysis For Android Applications”, http://android-developers.blogspot.in/2011/03/memory-analysis-for-android.html, March 24, 2011[14-08-2016].

[4] Profile. “Android DDMS”, https://developer.android.com/studio/profile/ddms.html, March 24, 2011 [15-08-2016].

[5] MAT. ”Memory Analyzer (MAT)”, http://www.eclipse.org/mat/, April 15, 2016[19-08-2016].

[6] Command Line. “systrace”, https://developer.android.com/studio/profile/systrace-commandline.html, April 15, 2016[19-08-2016].

[7] Roman Guy. “Android Performance Case Study”, http://www.curious-creature.com/2012/12/01/android-performance-case-study/ , Dec. 1, 2012[20-08-2016].

[8] Profile. “Profile your layout with Hierarchy Viewer”, https://developer.android.com/studio/profile/optimize-ui.html 20-08-2016

[9] The Statistics Portal. “Most popular global mobile messenger apps”, http://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/ May 15, 2016[24-08-2016].

[10] Doug Sillars. ”CPU and CPU Performance”, https://www.safaribooksonline.com/library/view/high-performance android/9781491913994/ch06.html, Oct, 2015 [25-08-2016].

[11] Doug Sillars. ”Screen and UI Performance”, https://www.safaribooksonline.com/library/view/high-performance-android/9781491913994/ch04.html, Oct, 2015 [25-08-2016].

[12] Doug Sillars. “Memory Performance”, https://www.safaribooksonline.com/library/view/high-performance-android/9781491913994/ch05.html, Oct, 2015 [25-08-2016].

[13] Android Studio User Guide. ”Test your app”, https://developer.android.com/studio/test/index.html, April 4, 2016 [27-08-2016].

[14] Analyzing Java Memory. “Memory Management”, http://www.dynatrace.com/en/javabook/analyzing-java-memory.html, April 4, 2016 [27-08-2017].

[15] Inspect trace logs with Traceview. “Traceview”, https://developer.android.com/studio/profile/traceview.html, April 5, 2016 [15-08-2016].

[16] MAT.”Memory Analyzer (MAT)”, http://www.eclipse.org/mat/, April 15, 2016 [19-08-2016].

[17] “How many app downloads are there per day for both the App Store and Google Play”, https://www.quora.com/How-many-app-downloads-are-there-per-day-for-both-the-App-Store-and-Google-Play, Nov, 2016[26-08-2016].

[18] Thiago Soares Fernandes, Erika Cota , Alvaro Freitas Moreira “Performance Evaluation of Android Applications: a Case Study”, SBESC '14 Proceedings of the 2014 Brazilian

Symposium on Computing Systems Engineering, IEEE Computer Society Washington, DC, USA, 978-1-4799-8559-3, 2014

[19] Liu Pu, “Performance Analysis Of Short Messages”, Intelligent Ubiquitous Computing and Education, 2009 International Symposium on Intelligent Ubiquitous Computing and Education, IEEE, 10859941, Chengdu, China, 2009.

[20] Anthony Gutierrez, Ronald G. Dreslinski, Thomas F. Wenisch, Trevor Mudge, Ali Saidi, Chris Emmons, and Nigel Paver, “Full-system analysis and characterization of interactive smartphone applications”, In Proceedings of the 2011 IEEE International Symposium on Workload Characterization, IISWC ’11, pages 81–90, Washington, DC, USA, 2011.

[21] Dhinakaran Pandiyan, Shin-Ying Lee, and Carole-Jean Wu, “Performance, Energy Characterizations and Architectural Implications of An Emerging Mobile Platform Benchmark Suite – MobileBench”, IEEE International Symposium on Workload Characterization, IISWC 2013 - Portland, OR, United States, 2013.

[22] Rama Bhatia, Dr. Renu Bagoria, Dr. A. K. Mohapatra, 2016 “An analytical approach towards CPU, Memory & UI Performance Assesment of Android Apps”, “International Journal Of Control Theory and Application”,Volume : No.9 (2016) Issue No. :23 (2016) Pages : 247-251.