Analyzing and Improving Application Launch Time Performance in High Memory Pressures for Android

M Abubakar Siddique*, Hasan Maqbool† and Sheraz Naseer‡
Department of Computer Science, University of Management and Technology
Lahore
Email: *abubakar.siddique@umt.edu.pk, †hasan.maqbool@umt.edu.pk, ‡sheraz.naseer@umt.edu.pk

***Abstract*—Android is the most popular operating system. Although, it has been more than 7 years since its first release, yet there has not been developed any tool(s) that can analyze the performance of any application, especially at the launch time. Application (app) launch time plays an important role in the subsequent success of the application, as it gives the first impression about the application performance to the user. This analysis is of crucial importance in situations when device is running on low available memory. The availability of such automated systems can allow application developers to enhance the application performance thus leading to improved user experience. In this paper, the application launch time performance is our focal point with the detailed analysis of application performance. Particularly, we present a system that analyzes applications launch time in almost any given scenario; for instance, an application is launched in normal and high memory pressure; high memory pressure without network connectivity and cached application. The proposed system can clearly show the difference in the launch time of several types of applications in different scenarios. Finally, we also suggest techniques to improve the application launch time performance.**

I. INTRODUCTION

Android certainly is the most growing [1] operating system (OS). It runs on a variety of devices like smart phones, tablets, TVs, watches, and an un-ending list goes on. Random-access memory (RAM) is a precious resource, especially on a mobile OS. To handle low memory scenarios, android has a low memory killer (LMK). But, when a user quits an application (app), activity manager does not kill the process instead it is kept in a least recently used (LRU) cache to reduce the response time, just in case if user returns to app in the future [2]. Android defines different levels for LMK to kill empty apps to Foreground running apps depending on the free memory size. Table 1 shows, the free memory and type of the apps to kill to free memory on device.

Almost all android devices have a limited memory, so there are more chances to run out of memory or low memory scenarios. Therefore, when most of the apps are launched, android has to free some memory, which adversely affects the app launch time, which is the duration when a user touches the app icon till the first screen is shown for interaction. App launch time plays a vital role in user experience of the app.

In spite of great success of android OS in the developer community and in general public overall, there has not been developed any tool(s) to analyze the performance of any application in general and specifically at the launch time.

TABLE I
NEXUS 5 RAM THRESHOLD IN MB(S) FOR LMK

Type of App	Memory (MB)	Type of App	Memory (MB)
Empty App	180	Content Provider	144
Hidden App	126	Secondary Server	100
Visible App	150	Foreground App	90

Such an analysis has significant importance in various settings, particularly in low available memory scenario. Analysis of app launch time performance is the main contribution of this paper.

The remainder of this paper is organized as following. Section II introduces android OS and its app launch procedure. Section III provides a brief overview of the related work. Section IV presents our proposed app launch time analyzing system. Section V shows several observations and insights into the data acquired. Finally, we conclude with some suggestions to improve app launch time performance in section VI.

II. ANDROID APP LAUNCH PROCEDURE

Every Android application runs in its own process and each process is forked from Zygote [3]. Zygote is the system process preloaded with some common code and framework classes to be shared across all processes. Every process needs some special class files and reading those files individually may not be effective. To make it faster, zygote preloads these class files; and fork from zygote makes it available for all processes.

This method is also helpful in memory management; zygote is forked with copy-on-write (CoW). Thus, no memory overhead will occur, until write operation is performed for individual class file. These classes remain shared across all processes, because most of the time, write operation is not performed for these classes.An android application is launched by two procedures; first one is cold launch, when process is to be forked from Zygote. The second procedure is warm launch, when process already exists in background and background process is brought to the foreground.

In case of cold launch, as shown in Figure 1, firstly the user touches an application icon. Launcher application sends intent to be launched which is a request to launch the desired application. Activity manager, that is being run inside System server will receive intent and prepares it to be launched. Activity manager requests to the window manager to prepare a

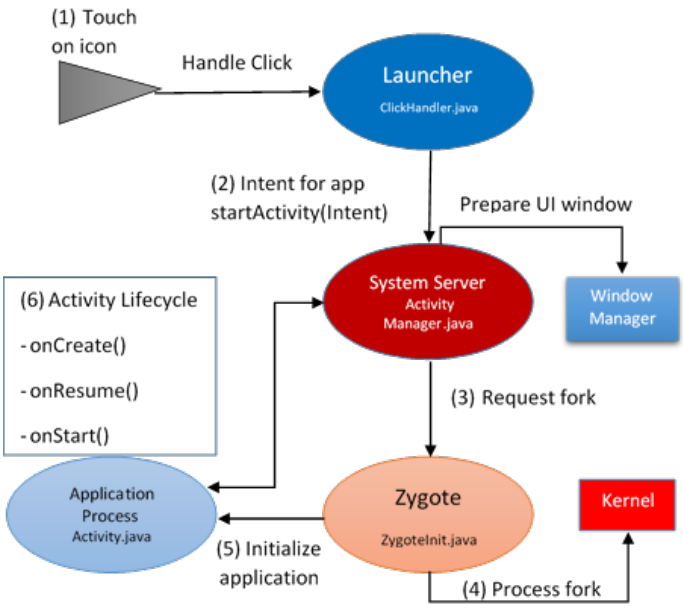

Fig. 1. Android app cold launch

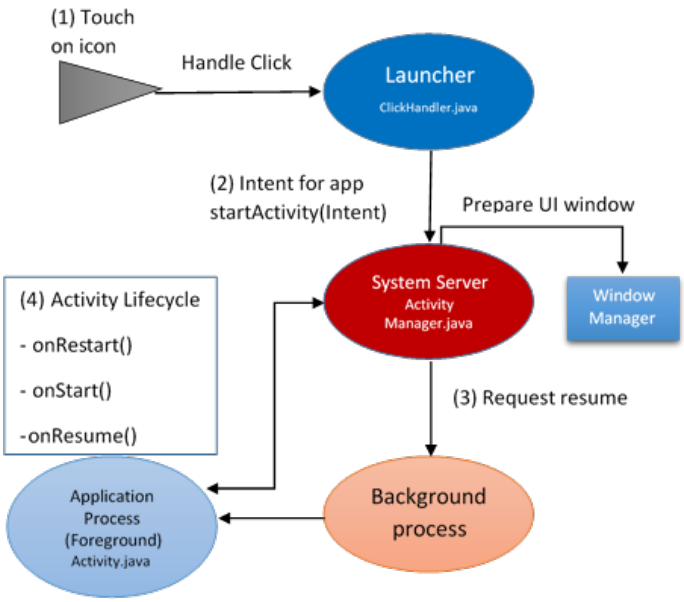

Fig. 2. Android app warm launch

window with token. Then, activity manager sends the request to fork zygote itself and generate child process. A new process is initialized and activity manager brings it to the foreground. Further, activity of that process starts calling its onCreate, onStart and onResume methods; and continues with its typical lifecycle [4].

During warm launch, as presented in Figure 2, the user touches an application icon; the launcher application sends intent to be launched to the activity manager, which sends request to the window manager to prepare a window with token. Furthermore, the activity manager sends resume request to the existing process present in background. The background process resumes itself and activity manager brings it to the foreground. Further, the lifecycle of activity continues with onRestart, onResume and onStart procedures.

As it is given in Figure 3, cold launch takes long time to launch and warm launches are quick. Android does not kill every process on exit, to make apps switching faster. Android keep apps cached in RAM, until there is not enough memory available to load new application. In this scenario, Android uses LRU algorithm to kill processes in background and keeps killing until there is sufficient memory available.

III. RELATED WORK

Android OS performance for real-time embedded system has been evaluated in [5]; this study exposed various limitations of android OS. An analysis of android mobile web developed using PhoneGap has been discussed in [6]. Authors in [7] suggest a way of minimizing launch time of apps up to 6 seconds per app using predictive user context. Windows phone OS was modified as an implementation of the system; however, such analysis was not performed for android OS.

Only a few studies have been conducted on the subject. Comparison of app launch time till 8th launch has been performed in [8], thus considering cached apps takes significant less time after 4th launch. Several scenarios have not been considered in this study, e.g. network requests, high memory pressure, etc.

It is suggested in [9] that app launch time can be minimized by preloading a few variants of java packages and some improvements in app launch time have also been shown. Preloading such packages may fasten some specific type of applications, but can it handle all types of applications is a big question mark. Moreover, overhead of preloading such java packages have been completely ignored.

The app launch time performance is the main focus of this paper; we also perform detailed analysis of app performance, almost all possible scenarios have been considered including high memory pressures. Our work also adds various suggestions to minimize the app launch time without any overhead and caters the usage behavior, thus considering all possible types of apps.

IV. OUR PROPOSED SYSTEM

We have developed a system that analyzes the app launch during normal and high memory pressures scenarios. We have chosen a set of top rated and well performing apps from Google Play and results can be viewed in graph. Our system always kills required application process to make sure we are always performing cold launch. Then it calculates the free memory (excluding cached memory) and further creates high memory pressure by launching multiple apps and keeps switching between them until free memory is lower than low memory threshold (when android starts killing processes from LRU).

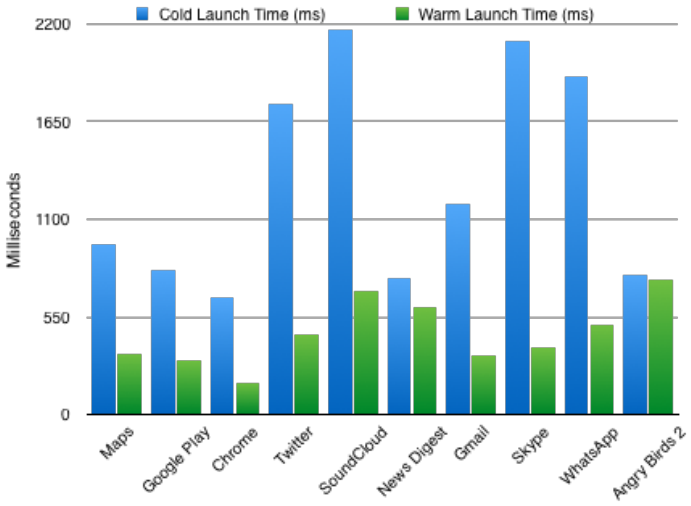

Fig. 3. App launch time in cold and warm launch scenarios.

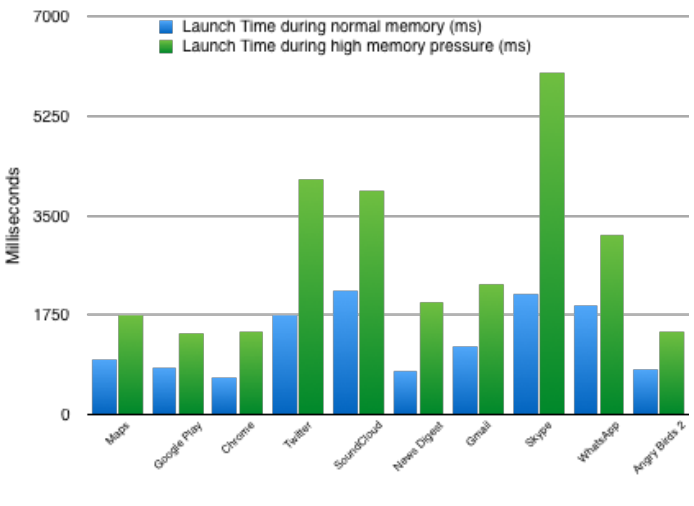

Fig. 4. App launch time in normal and high memory pressure scenarios.

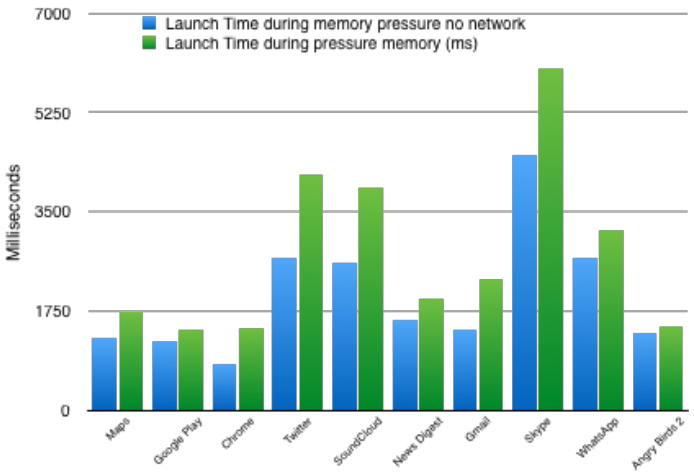

Fig. 5. Effect of network communication on app launch time.

Finally, it launches the application and calculates its launch time. We have used Nexus ROM to test and we have not added any monitor in android ROM. Hence, there is no memory overhead. Our system only uses the android logs to calculate the launch time for an app that starts with touch on app icon and ends when onResume method is called.

V. EVALUATION

The proposed system has been used to analyze launch performance of several top rated apps and achieved performance statistics are presented and discussed in this section. Android allows a maximum number of memory to allocate by an application, depends on vendors, because memory for different handsets range from 512 Megabytes (MB) to 4 Gigabytes (GB). For instance, LG Nexus 5 allows 192 MB in standard mode and 512 MB in cases where application developers require enhanced graphics and high memory consumption during application usage. Google Nexus 5 running Android (5.1.1) has been used to analyze our system.

The app launch time results during normal and high memory pressure scenarios are presented in Figure 4. Acquired results show that there is a significant difference between normal launch and high memory pressure launch time. All selected apps have twice or greater launch time during high memory pressure. Further it is also found that apps using network calls at launch have higher launch time.

Further we have investigated how the network communication affects the app launch time during high memory pressure. Figure 5 gives comparison between high memory pressure launch time of apps with network connectivity and during no internet connection. It shows that launch time in no network connectivity scenario is much reduced for apps using network requests at launch time, e.g., Twitter, Sound Cloud and Skype.

Presented results clearly indicate that games (high graphics) and apps requesting network connectivity at earliest stage or during launch are taking long time. During high memory pressure, every single app takes higher launch time, which suggests that high memory usage increases the launch time of apps.

VI. CONCLUSION

In this paper, we proposed a system to analyze the android app launch time for almost any possible given scenario. Our system only uses android OS logs to monitor the app launch events, thus neither custom ROM nor extra usage of memory occurs. Evaluation of the app launch performance using the proposed system revealed that cold launch is always expensive than warm launch. The launch time of all the apps tested increased by a factor of at least two in high memory pressure. Performance of the apps making network request at app launch degraded even more; and launch time increased by at least three times.

Based on the investigation using the proposed system, we propose a few techniques to minimize the app launch time with

existing android ROM and no memory overhead. Cold launch time can be reduced by minimizing network requests during launch. Apps should request memory allocations gradually, to make sure that OS will need to kill minimum number of processes, this will in effect reduce the app launch time as well. Another effective way to reduce launch time would be to use use warm launches instead of cold launches when possible. User context can be used to learn users app usage behavior, thus keeping LRU cache updated, can also lead to minimize the app launch time.

We also plan to implement a modified version of android ROM for fast app launch using predictive user context, which will rank apps on device and try to keep them in cache, hence, making it possible to use warm launches.

REFERENCES

[1] Gartner Smart Phone Marketshare 2015 Q2. Gartner, Inc. Retrieved 2015-08-21. http://www.gartner.com/newsroom/id/3115517

[2] Managing Your Apps Memory. Android Developer portal, Retrieved 2015-08-29. https://developer.android.com/training/articles/memory.html

[3] Processes and Threads. Android Developer portal, Retrieved 2015-08-29. http://developer.android.com/guide/components/processesandthreads.html

[4] Activities. Android Developer portal, Retrieved 2015-08-29. http://developer.android.com/guide/components/activities.html

[5] Maia, Cludio, Luis Miguel Nogueira, and Luis Miguel Pinho. Evaluating android os for embedded real-time systems. 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications. 2010.

[6] Corral, Luis, Alberto Sillitti, and Giancarlo Succi. Mobile multiplatform development: An experiment for performance analysis. Procedia Computer Science 10 (2012): 736-743.

[7] Yan, Tingxin, et al. Fast app launching for mobile devices using predictive user context. Proceedings of the 10th international conference on Mobile systems, applications, and services. ACM, 2012.

[8] Nagata, Kyosuke, and Saneyasu Yamaguchi. An Android application launch analyzing system. Computing Technology and Information Management (ICCM), 2012 8th International Conference on. Vol. 1. IEEE, 2012.

[9] Nagata, Kazuyuki, et al. Measuring and Improving Application Launching Performance on Android Devices. Computing and Networking (CANDAR), 2013 First International Symposium on. IEEE, 2013