Real-time IO Trace Analysis of Smartphone Users

Kyuil Kim, Eunryoung Lim, Seongjin Lee and Youjip Won
Dept. of Computer and Software
Hanyang University
Seoul, Korea
{gaya | erlim | insight | yjwon}@hanyang.ac.kr

Abstract:

*As the number of smartphone users increases, so do the computational power of the smartphone devices and the number of applications that exploit network and storage; however, as multiple applications are running concurrently, fighting for limited resources such as CPU and storage, it is becoming more difficult to satisfy users' everyday needs. Although smartphone manufacturers and application developers are striving to improve the efficiency and the performance of applications, their effort seems to fall far behind in solving low IO performance problem. In this paper, we captured and analyzed IO traces of two users who exhibit different IO patterns; user1 mostly used the smartphone to listen to music and read postings on Facebook application and user2 used Digital Multimedia Broadcasting (DMB) or Internet Protocol Television (IPTV) to watch videos. From this experiment, we found the following: about 28% of all traffic was generated during 11:00 to 12:00 and 21:00 to 22:00; /**data** partition received 37% of all IO traffic; the number of write IOs was 12 times higher than the number of read IOs; about 33% of all IOs accessed miscellaneous files; and SQLite and SQLite-journal accesses constitute about 21% and 26% of all IOs, respectively.*

Keywords-component; *Android, Smartphone, IO Characterization*

I. INTRODUCTION

As smartphones dominate mobile phone market and are becoming a part of our everyday lives, smart devices have replaced many of the special purpose handheld devices such as feature phones, cameras, and MP3 players. The smartphones are now capable of making phones calls, taking pictures, playing music, and even checking security camera installed in a house from anywhere and anytime. Release of wearable devices, such as smart watch and Google glass, is ever diversifying the use of smartphones. Many of recent market analysis show that sales of handheld smart devices have overwhelmed the sales of desktop computers [21].

As the technology trends and interests of research community shifted, researches in the field of IO characterization have also touched various sections of computing environments from enterprise servers [1, 2], web servers [3, 4], and OLTP servers [5] to desktop PCs [6, 7]. Researchers are now actively analyzing smartphones with respect to IO patterns [8, 9] and IO performance analysis [10], as well as optimizing the mobile IO stack [11] and improving the IO performance [12-14]. Although smartphone manufacturers are pushing the performance of their hardware to its limit, users' requirements are not yet satisfied by their effort.

Since smart devices are prone to power failures, reliability and integrity of data are the most important issues in smartphones. The only way to ensure data integrity is to store the data in a persistent and nonvolatile storage medium. Applications and kernel exploit database as a solution to store not only user sensitive data but also managerial information to recover from software and hardware failures. Since storage devices are known as one of the main causes of performance degradation [10], we believe understanding the IO pattern of smart devices gives us insight into designing efficient IO stack in smartphones as well as improving ways to implement software.

In this work, we analyze the IO patterns observed in everyday use of smartphones. We recruited two volunteers to use Nexus5 from 19:00 on April 26, 2014 to 19:00 on April 27, 2014 and acquired IO traces generated by them. After acquiring user IO traces, we analyzed IO frequency, the number of IOs on different partitions, the number of IOs per file type, and the ratio of synchronous IOs to buffered IOs. Some of our findings are as follows: 28% of all traffic was generated during 11:00 to 12:00 and 21:00 to 22:00; **/data** partition received approximately 37% of all IO traffic; the number of *write* IOs was 12 times higher than the number of *read* IOs; 33% of all IOs accessed miscellaneous files; and IOs from SQLite and SQLite-journal constitute about 21% and 26% of total IOs, respectively.

II. BACKGROUND

A. IO Stack of Android

Android is Linux based mobile platform developed by Google and is the most popular mobile platform [15]. Since Android is open source project, users may change it freely to improve its performance and customize various features. Recently, as Android is penetrating areas other than smartphones, understanding how Android IO stack works is becoming more important than before. Android has already entered the TV market and it is expanding its domain to watches and cars.

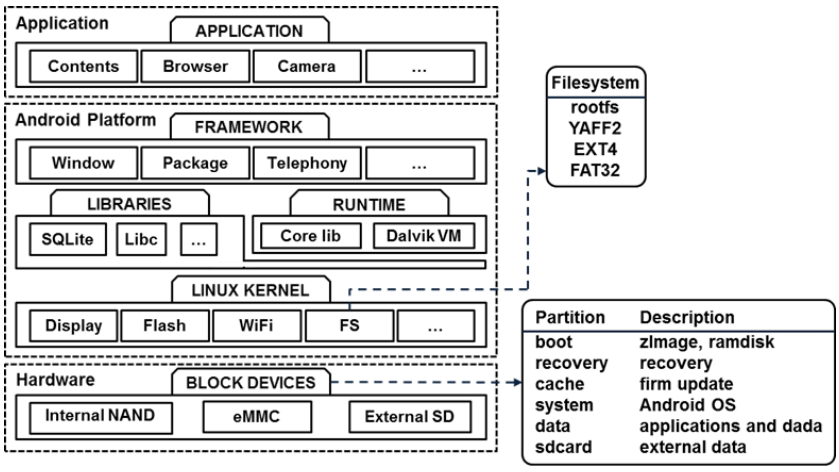

Figure 1. Android Architecture and Storage Partitions

Figure 1 shows the architecture of Android which has three major layers: application, Android platform, and block device layer. All Android applications are written in Java and packaged as .apk (Android application package) files. Android platform provides many libraries, such as SQLite, media framework, and libc, which are

exploited by applications and system components. Also, there is Dalvik virtual machine that runs application code and processes just-in-time compilation to run .dex (Dalvik executable) files. Linux kernel is at the core of Android platform and manages memory, processes, security, and networking.

B. Device: Nexus5

In order to acquire real time IO traces of smartphone users, we recruited two volunteers to use Nexus5. Before handing out the devices, we modified the Android kernel to allow tracing of IO activities. Nexus5 is Google's fifth reference phone that runs KitKat, which was introduced on October 31, 2013 [16, 17]. KitKat is based on Linux kernel 3.4.0. Nexus5 has 16GB storage with Quad-core 2.26Ghz Krait 400 and 2GB Ram. Android has five different partitions: **/boot**, **/recovery**, **/data**, **/system,** and **/cache.** Most of the IOs are observed on three of the partitions: **/data**, **/system**, and **/cache**. Of 16GB in the test device, 13GB is used for **/data** partition which stores user installed applications and user data; approximately 1GB is used for **/system** partition which stores read-only system files; about 700MB is used for **/cache** partition which stores temporary files and updates to files downloaded from Android Market; the rest is used for /recovery and /boot partitions. All the partitions are formatted with EXT4 journaling file system.

III. DESIGN

MOST (Mobile Storage Analyzer) [18-20] is an IO trace analysis tool for smartphones. Compared to blktrace in Linux, MOST provides three new features: LBA-to-file mapping, LBA-to-process mapping, and retrospective mapping which keeps track of deleted files while IO tracing. Although MOST provides useful information to understand captured IO traces, its analysis speed needs much improvement. MOST is not the optimal solution in analyzing IO traces captured over a long time period nor is it suitable for real time analysis. To analyze the captured IO traces from the users in our experiment, we modified the Android kernel to enable the features of MOST; the kernel not only acquires IO traces of the underlying block device but also analyzes the IO traces in a time efficient way.

A. IO Trace Acquisition

The essence of IO tracing capability lies in collecting the information provided by *submit_bio()* function in general block device layer. The kernel module then processes received information and extracts files. User daemon provides interface for acquiring IO traces and storing the information.

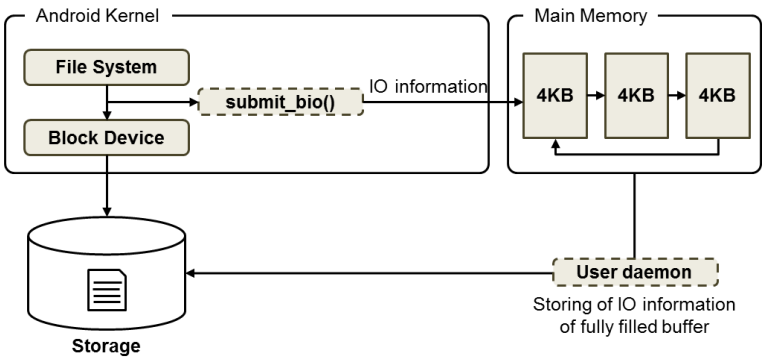

Figure 2. IO Trace Acquisition Process

submit_bio() function is a function in general block device layer that is called when file system approaches block layer. We modified this function and added procedures to extract information on each IO and to store IO traces in a buffer. When the acquired IO information fills the buffer, *submit_bio()* function stores IO traces in binary format [18, 19]. Table I describes format of collected information.

TABLE I. RAW DATA FORMAT

Variable Name	**Collected information**		
	Type	**Size(byte)**	**Description**
time	long long	8	IO occurrence time(year, month, day, hour, min, sec)
usec	long long	8	IO occurrence time(us)
Ext	char	1	File type(refer to table II)
rwbs	char	1	Flags for read, write, readsync, writesync
sector	unsigned int	4	Sector address
block_num	unsigned int	4	Number of blocks
Pid	unsigned int	4	Process ID
pname	char	Max 16	Process name

When the system boots, it loads kernel module, which acquires IO traces, and runs user daemon, which flushes the data to the device. As soon as the kernel module is loaded to the system, it creates three 4KB buffers in the main memory. We use two of the buffer spaces for producer and consumer model. The third buffer is used only when the primary producer buffer is filled up before the contents in the consumer buffer is flushed down to storage. If the primary producer buffer gets full, the third buffer acts as next in line producer buffer to receive IO data from *submit_bio()*. The kernel module is responsible for checking the buffer space to see if it is full and notifying the user daemon that the buffer is ready for flush. Upon receiving the notification from the kernel, user daemon flushes the buffer to designated location in storage.

We grouped the files extracted by the kernel module into seven categories depending on file extension name. We briefly describe them in Table II. Using the file types, we further analyzed the patterns and characteristics of the collected IOs.

TABLE II. FILE TYPE CATEGORIES

File Extension	**Description**
db	SQLite file
db-journal	SQLite journal file
db-mjxxxx, db-wal	SQLite-temp file
jpg, 3gp, mp3, thumb, local	Multimedia file
so, dex, odex, apk	Executable file
localstorage, dat, xml, thumbdata3, cache	Resource file
including directory entry	Miscellaneous file

IV. EVALUATION

The two volunteers in our experiment used modified Nexus5 kernel with IO tracing feature from 19:00 on April 26, 2014 to 19:00 on April 27, 2014. With acquired IO traces from the users, we measured the following: the frequency of IO requests per every hour, IO counts per file category for seven different file categories, and the number of IOs for each application used. By measuring the frequency of IO requests per hour, we learn when the

users use the smartphones the most. The number of IOs per file category tells us which file categories to optimize to improve the overall IO performance. Also, Nexus5 allows analyzing the application usage patterns based on IO frequency.

A. IO Trace Analysis

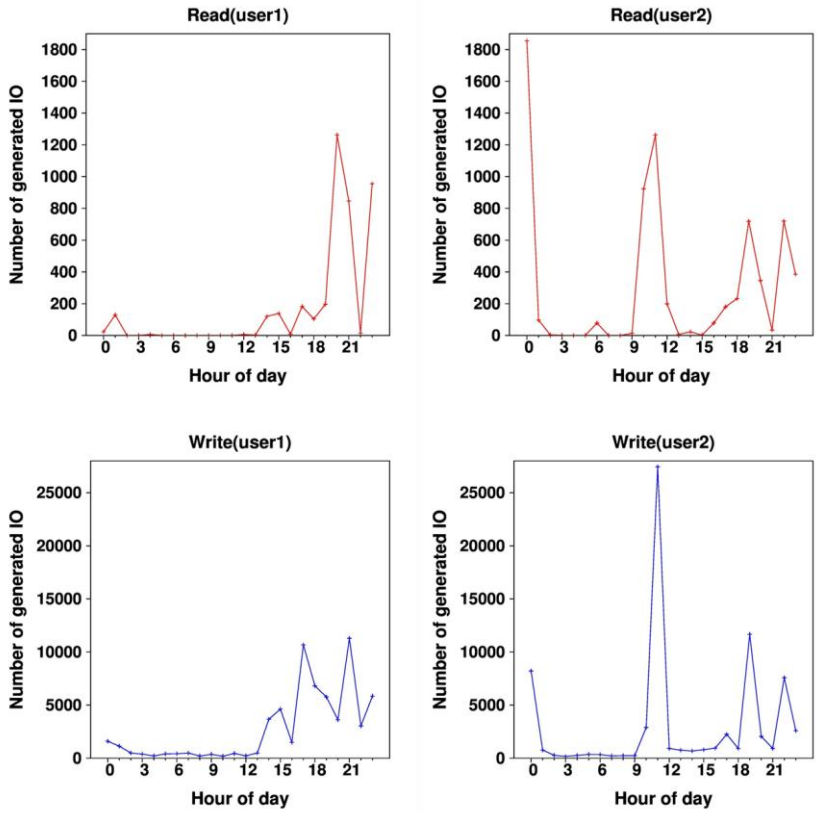

Figure 3. Number of Generated IOs by the Hour

Figure 3 illustrates the number of IOs generated per every hour (the graphs on the left side are for user1 and ones on the right are for user2). The result shows clear sign of diurnal pattern; there are no significant IO activities between 02:00 and 12:00 for user1 and between 02:00 and 09:00 for user2. Although there were no significant IO activities observed from midnight to dawn, some applications synchronized with the application server, along with the phone and message modules, during this time; in fact, a small number of *read* IOs observed at 06:00 for user2 was triggered when Google Plus synchronized with the server.

The peak IO activity for user1 is observed during night hours whereas user2 used the device actively just before lunch and dinner times. We found that Mini T world application, which accesses phone and SMS activities, account for 33% of all *read* IOs in user1. For user2, however, web browser app generated about 14% of all *read* IOs. After analyzing *write* IOs, we found that for user1, using Facebook generated 21% of all *write* IOs. For user2, installing and updating database of a subway route application generated 16% of all *write* IOs. User1 mostly used media player to listen to music and to watch VODs and Facebook application to read postings. User2 mostly used the device to watch streaming data using DMB or IPTV services. We also observed that *write* IOs account for 92% of all IOs traced.

B. IO Distribution on Each Partition

Since **/system**, **/data**, and **/cache** are the three largest partitions that receive most of IOs, we mainly focus on these partitions and analyze how many IOs are directed to each of them. **/boot** and **/recovery** partitions rarely receive any IOs compared to **/system**, **/data**, and **/cache** partitions. We denote **/etc** to represent **/boot** and **/recovery** partitions. We are especially interested in the IO counts for *read*, *write*, and *writesync* issued in each partition. The result is shown in Figure 4.

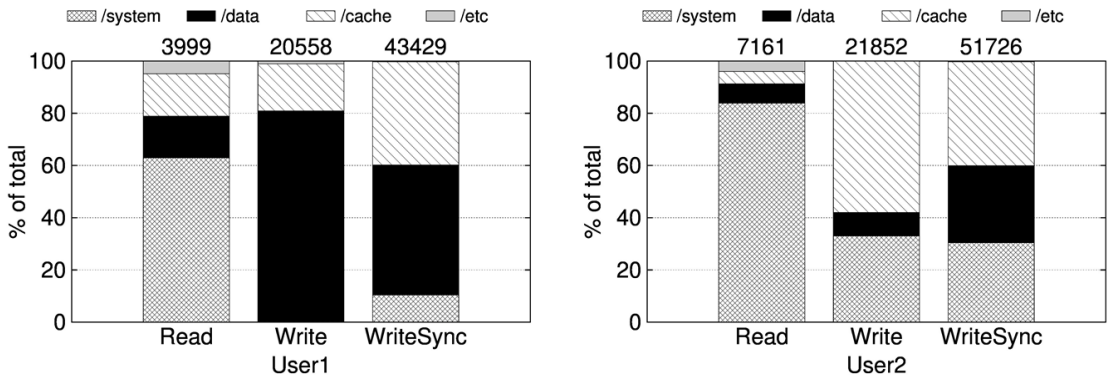

Figure 4. IO Distribution on Each Partition

On user1's device, **/data** partition received 57% of all IOs. For user2, about 41% of all IOs were targeted to **/cache** partition. We see that there are a lot of *write* and *writesync* IOs on the devices; *write* and *writesync* IOs account for about 94% and 91% of all IOs for user1 and user2, respectively. In other words, the number of *write* IOs was 16 times higher than the number of *read* IOs for user1 and 10 times higher for user2.

We analyzed our experiment result to find applications that generated the most IOs for each user. For user1, running Facebook and updating the application generated 34% of all IOs. Of the 34%, 12% were stored on **/system** partition and another 12% were stored on **/data** partition. The process that generated the second highest number of IOs for user1 was voice system application, with 30% of all IOs, all of which were stored on **/cache** partition. For user2, web browser, e-mail, and gallery application generated the most IOs. About 15% on **/system**, 10% on **/data**, 5% on **/cache**, and 13% on **/etc** of all IOs are for processing subway route application, mail service, web browser, and gallery application, respectively.

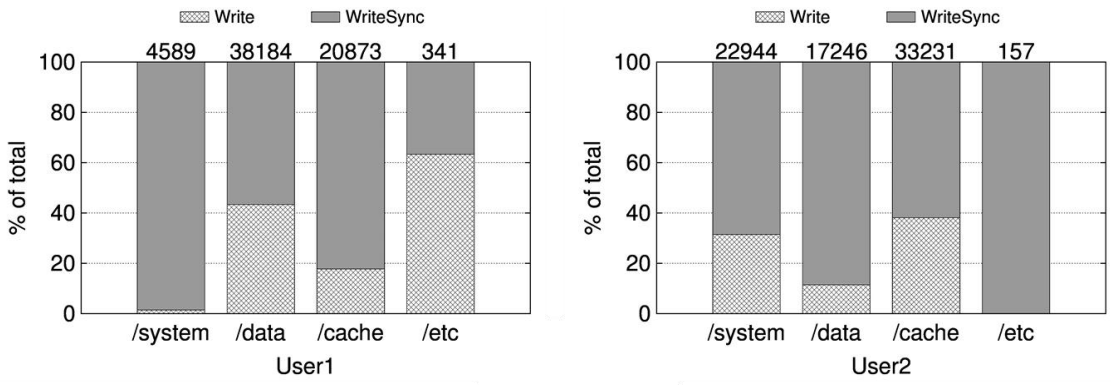

Figure 5. Distribution of write and *writesync* in Each Partition

Figure 5 shows the ratio of *write* and *writesync* IOs in each partition. It shows that 69% of all IOs were *writesync* for both users. **/system** and **/cache** partitions on user1's device received the most *writesync* IOs. On

user2's device, **/data** and **/etc** partitions received the most *writesync* IOs. For user1, Facebook application created the highest number of IOs. IOs created by Facebook accounted for 36% of all IOs on **/cache** partition and 34% of all IOs on **/system** partition. On the other hand, for user2, about 12% of IOs on **/data** partition were from e-mail service and 29% of IOs on **/etc** partition were from processing streaming based music player.

C. IO Distribution on File Type Categories

As described in Table II, we categorized the files extracted by the kernel module into seven categories depending on their extension names. Figure 6 shows the distribution of *read*, *write*, and *writesync* IOs for each file type category. Note that we separated SQLite data type from SQLite-journal data type based on recent work on Android IO characterization [19] which identified SQLite-journal as the main cause of excessive synchronous IOs in smartphones.

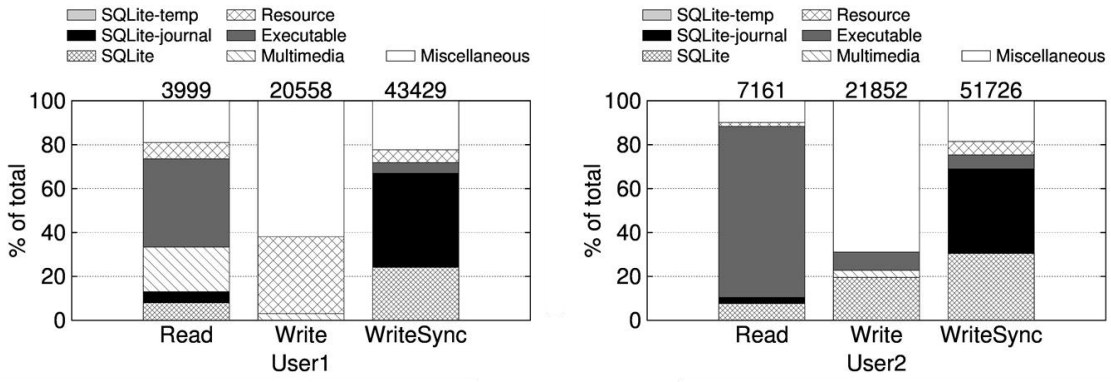

Figure 6. IO Distribution of Each File Type

For user1, accesses to multimedia and executable files constitute about 20% and 40% of all *read* IO, respectively, and resource and miscellaneous files generated 35% and 62% of all *write* IOs, respectively. For user2, accesses to executable and miscellaneous files constitute about 78% and 10% of all *read* IOs, respectively, and SQLite and miscellaneous files received about 20% and 69% of all *write* IOs, respectively. Also, SQLite and SQLite-journal files generated 21% and 26% of all IOs, respectively. It implies that optimizing SQLite and SQLite-journal will reduce the number of IOs to storage.

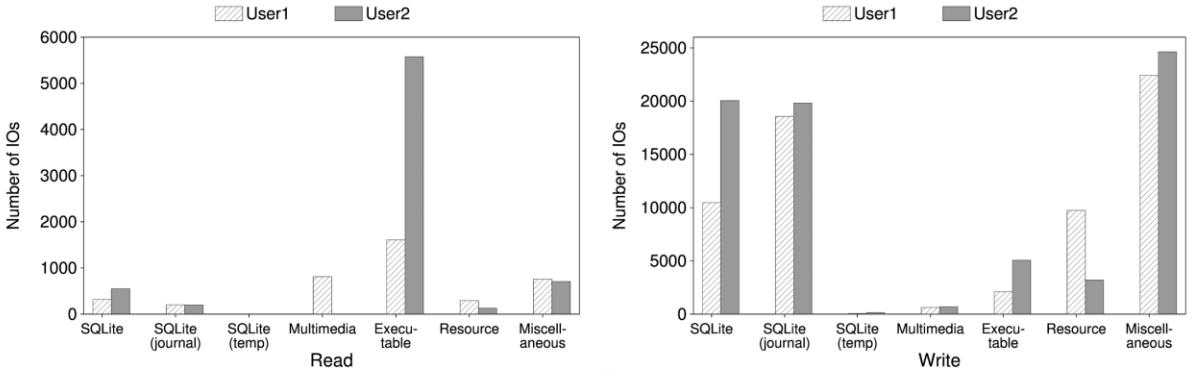

Figure 7. IO Counts for Each File Type

Figure 7 shows the number of *read* and *write* IOs for each file type category for both user1 and user2. Note that *write* IO count includes *write* and *writesync* IOs. For the two users, executable files constitute 64% of all *read* IOs on average. SQLite files, SQLite-journal files, and miscellaneous files constitute 22%, 28%, and 34% of

all *write* IOs, respectively. As a result, of all generated *write* IOs 22% and 28% of all IOs are for SQLite and SQLite-journal, respectively.

Also, the number of *write* IOs is 10 to 16 times higher than that of *read* IOs depending on the user. *Read* IOs for user1 and user2 account for about 6% and 9% of all IOs, respectively. It is interesting to see that SQLite did not make any *read* accesses to temporary database files, even though SQLite writes them to the device.

Figure 8 illustrates the ratios of accessed file types in each partition. It shows that there are a lot of accesses to SQLite files (SQLite, SQLite-journal, and SQLite-temp) on all partitions for both user1 and user2 and indicates that SQLite files cannot be taken lightly. Of all the IOs received on **/cache** partition of user1's device, about 74% were SQLite files. On user2's device, 65% of the IOs received on **/data** partition were SQLite files.

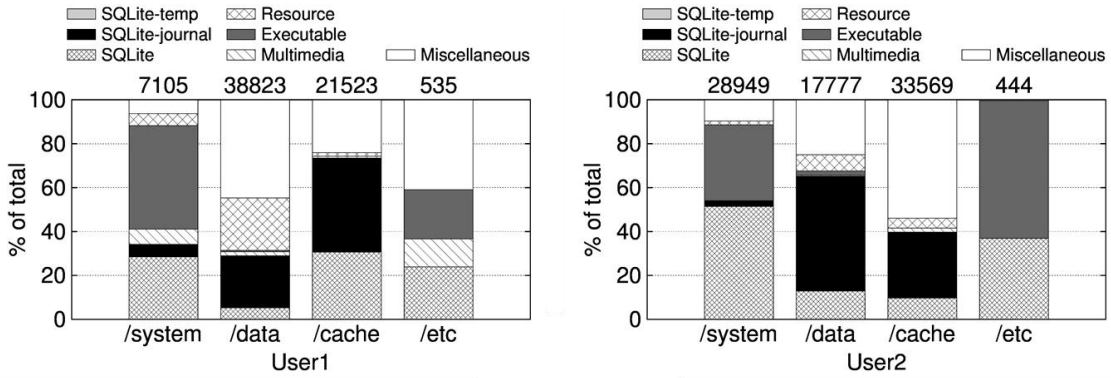

Figure 8. Ratio of Accessed File Types on Each Partition

Table III shows the most frequently accessed applications. The result shows that Facebook application generated the most IOs for user1. The application that generated the most IOs for user2 is subway route finding application. Although the run time of the subway application was short, around one minute, it generated the most IOs because it downloaded and updated its database. Ringtone is an application to create a ringtone and alarm for the device and Naver Music is streaming based music player.

TABLE III. MOST FREQUENTLY ACCESSED PROCESSES

	User1	**User2**
Rank	**Application Name**	**Application Name**
1	Facebook	Subway application
2	Ringtone	Naver Music
3	Android push alarm	IPTV

V. CONCLUSION AND FUTURE WORK

In this paper, we analyzed IO traces of two Nexus5 users from 19:00 on April 26, 2014to 19:00 on April 27, 2014. We modified general block device layer and implemented kernel module and user process to capture user generated IO activities. Some of our findings from analyzing acquired IO traces are as follows. IO traces show distinct diurnal pattern, where most of IO accesses are observed from 11:00 to 12:00 and from 21:00 to 22:00; sum of all IOs during these two time slots accounts for about 28% of all IOs on the devices. Sum of *write* and *writesync* IOs is approximately 10 to 16 times higher than the number of *read* IOs depending on the user. About 47% of all IOs are SQLite and SQLite-journal files; SQLite and SQLite-journal received 21% and 26% of all IOs,

respectively. This shows that optimizing SQLite and SQLite-journal IO activities is important in improving the overall IO performance.

This paper shows IO pattern information of smartphone users. However, since the experiment only included two users, it is hard to generalize our findings. Henceforward, we plan to collect and analyze data of many smartphone users using the techniques presented in this paper.

VI. ACKNOWLEDGEMENTS

New Memory: This work is supported by IT R&D program MKE/KEIT (No. 10041608, Embedded System Software for Newmemory based Smart Device).

REFERENCES

[1] A. Riska and E. Riedel, "Disk Drive Level Workload Characterization," in *USENIX Annual Technical Conference, General Track*, 2006, pp. 97-102.

[2] C. Ruemmler and J. Wilkes, "UNIX disk access patterns," in *USENIX Winter*, 1993, pp. 405-420.

[3] M. F. Arlitt and C. L. Williamson, "Internet web servers: Workload characterization and performance implications," *IEEE/ACM Transactions on Networking (ToN)*, vol. 5, pp. 631-645, 1997.

[4] K. Kant and Y. Won, "Server capacity planning for web traffic workload," *Knowledge and Data Engineering, IEEE Transactions*, vol. 11, pp. 731-747, 1999.

[5] S.-W. Lee, B. Moon, and C. Park, "Advances in flash memory SSD technology for enterprise database applications," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 863-870.

[6] T. Harter, C. Dragga, M. Vaughn, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "A file is not a file: understanding the I/O behavior of Apple desktop applications," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, p. 10, 2012.

[7] M. Zhou and A. J. Smith, "Analysis of personal computer workloads," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1999. Proceedings. 7th International Symposium on*, 1999, pp. 208-217.

[8] H. Verkasalo, "Analysis of smartphone user behavior," in *Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR), 2010 Ninth International Conference on*, 2010, pp. 258-263.

[9] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 179-194.

[10] K. Lee and Y. Won, "Smart layers and dumb result: IO characterization of an Android-based smartphone," presented at the Proceedings of the tenth ACM international conference on Embedded software, Tampere, Finland, 2012.

[11] S. Jeong, K. Lee, S. Lee, S. Son, and Y. Won, "I/O stack optimization for smartphones," *Presented as part of the 2013 USENIX Annual Technical Conference}*, 2013, pp. 309-320.

[12] Y. Nakamura, K. Nagata, S. Nomura, and S. Yamaguchi, "I/O scheduling in Android devices with flash storage," in *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, 2014, p. 83.

[13] M. Choi and S. H. Lim, "x86 - Android performance improvement for x86 smart mobile devices," *Concurrency and Computation: Practice and Experience*, 2014.

[14] C.-M. Lin, J.-H. Lin, C.-R. Dow, and C.-M. Wen, "Benchmark dalvik and native code for android system," in *Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference* 2011, pp. 320-323.

[15] *Android, the world's most popular mobile platform*. Available: http://developer.android.com/about/index.html

[16] A. Chauhan and V. Gaikar, "LG Google Nexus 5 Launched with Android 4.4 KitKat," 2013.

[17] *Nexus 5(google/LG)*. Available: http://en.wikipedia.org/wiki/Nexus_5

[18] S. Jeong, K. Lee, J. Hwang, S. Lee, and Y. Won, "Androstep: Android storage performance analysis tool," in *ME13: In Proc. of the First European Workshop on Mobile Engineering, Aachen, Germany*, 2013, pp. 327-340.

[19] S. Jeong, K. Lee, J. Hwang, S. Lee, and Y. Won, "Framework for Analyzing Android I/O Stack Behavior: From Generating the Workload to Analyzing the Trace," *Future Internet*, vol. 5, pp. 591-610, 2013.

[20] *Mobile storage analyzer (most)*. Available: http://dmclab.hanyang.ac.kr/sub/main_most.htm

[21] N. Hildebrand (May 9, 2014), "The Changing World of Mobile Electronic Devices", Display Central, Retrieved May 12, 2014, from https://www.display-central.com/free-news/display-daily/havent-you-heard-the-pc-market-is-changing/