IO Workload Characterization of Tizen Based Consumer Electronics

Myungsik Kim, Seongjin Lee,Youjip Won
Department of Computer Sciences
Hanyang University
Seoul, South Korea
{mskim77 | insight | yjwon}@hanyang.ac.kr

Abstract -*In this work, we explore IO characteristics of the Tizen based smartphone to provide insights for optimizing the IO performance and the IO system design in Tizen based Consumer Electronics. Summary of our findings are as follows: 90% of all write IOs are synchronous, 75% of all write accesses are by SQLite, 85% of all IOs are random accesses, and 90% of all writes are in synchronous IO mode. RUA (Recently Updated Application) manager, which is called at the beginning of every application, generates 18 write IOs to update rua.db file.*

Keywords— Tizen; Smartphone; IO workload characterization

I. INTRODUCTION

Recent advancement in CPU power and IO performance in embedded storage devices made possible for the smart devices to seamlessly run a wide variety of applications. However, mix of workloads induced by applications makes difficult to satisfy the performance need such as responsiveness and low IO latency while running multiples of applications.

Previous IO characterization researches focused on Android smartphones only. To best of our knowledge, there is no benchmark and analysis on Tizen environments, and this is the first to present the result of the platform. Kim et al. [1] presented IO characteristic of Android smartphones that points out that storage performance over Wi-Fi can vary between 100% to 300% depending on the choice of applications. Lee et al. [2] showed that 70% of all writes are random in Android platform, and SQLite and EXT4 generates excessive Journaling IOs. Jeong et al. [3] further analyzed the IO characteristics of smartphones to point out that 90% of write requests are issued by SQLite and journal, 30% of all writes are issued by EXT4 filesystem journal, 70% of all writes are synchronous, and 75% of all writes are random. Jeong et al. [3] also proposed to optimize the Android I/O stack by changing the *fsync* to *fdatasync* mode, changing the default DB journal mode to WAL mode, adopting polling-based I/O, and exploiting F2FS filesystem.

Tizen is open software platform that targets consumer electronics. Tizen aims to be a cross-device platform that allows multiple device applications facilitating one unified platform environment. Unlike Android platform, target platforms for Tizen is not limited to smartphones, but includes Smart TV, Automotive Infotainment devices, Camera, Smart watch, and Refrigerator.

In this paper, we present IO workload characterization of Tizen based consumer electronics. To analyze the IO behavior of Tizen applications, we acquired IO traces of eleven applications, and extracted number of features of the workload such as file type, IO size, IO operation type (read/write), random/sequential, block semantics (Data/Metadata/Journal), and IO mode type (buffered vs. synchronous IO). Some of our findings are as follows: 45% of all writes are 4KB in size; 85% of all writes are random accesses; 90% of all writes are in synchronous IO mode; 66.4% of total IO counts are generated by storing metadata and journals; and, RUA (Recently Updated Application) manager, which is called at the beginning of any application, generates 18 write IOs to update `rua.db` file.

This work is sponsored by IT R&D program MKE/KEIT (No.10041608, Embedded system Software for New-memory based Smart Device).
This works is funded in part by "eMMC IO Characterization on Tizen OS", Samsung Electronics

II. BACKGROUND

A. Tizen Platform

Tizen is the open source platform designed for many device platforms [4, 5] such as smartphone, Automotive In-vehicle Infotainment (IVI), Tablet, Smart TV, Smart Watch, Camera, and Refrigerator. To accommodate wide variety of platforms and architectures, Tizen provides device API and libraries that allows to execute HTML5 based Web Runtimes; they are key enabler of cross-platform environment for web based applications. An advantage of HTML5 based web applications is its portability across different platforms.

The Tizen Platform is shown in Figure 1 which is composed of web/native application framework and core on top of Linux Kernel. Core provides common APIs for Web and native frameworks. It consists of a set of open source libraries. Web framework includes HTML5/W3C APIs, device APIs, and web runtime based on WebKit2. Native framework provides APIs and open source libraries in C/C++.

Web and Native apps have their own merits such as ease of portability and low complexity. Web framework facilitates

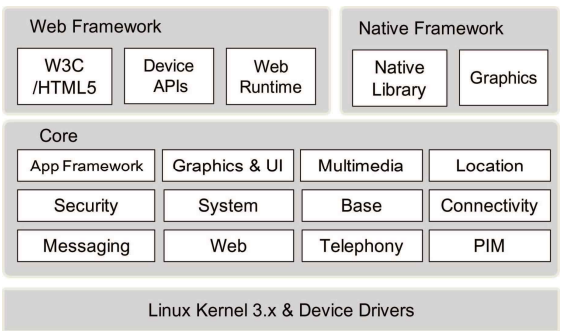

Fig. 1. Tizen Platform Architecture (Version 2.2.1) [6]

TABLE I. WORKLOAD SCENARIOS

Category	*Name (abbreviation)*	*Scenario*
Basic	Contact1 (Cn1), Contact2 (Cn2)	Adding contact list
Web	Google (Br1), Daum (Br2), Naver (Br3)	News reading, web searching
Mail	Mail (Ml)	Read and reply mail
Camera	Camera (C), Camcorder (Cc)	Take a picture, record movie
Multimedia	Media (Me), Music (Mus), Gallery (Gal)	Play movie, music, image in storage
Webstream	Youtube (You)	Play web-streaming movie
HTML5	Fishbowl (Fsb), Game (Gam)	Webapp benchmark

developing Web apps, and Native framework exhibits a better performance than Web apps because Native framework offers better resource optimization. Through unified SDK for web and native app, developers can make hybrid apps exploiting the best of the two frameworks. The force that drives the development of Tizen is governed by TSG (Technical Steering Group) which is consist of Intel and Samsung. Tizen Association is the union of industry stakeholders (e.g., device manufacturers, chipset vendors, carriers, etc.). The two bodies are trying to compete with iOS[7], Android[8], and minimize the dependencies on specific platform [9].

III. PROBLEM ASSESSMENT

Storages are a low speed device compared to the other components in a computer system. IO transactions in storage are one of the major factor that affects performance of the computer system; inefficient IO behavior leads severe performance degradation. Since embedded devices use Flash memory-based storages, inefficient IO behaviors reduces the lifespan of the storage; NAND Flash memory that are used in mobile devices have limited program/erase cycles.

Jeong et al. [3] recently pointed out that journaling of journal (JOJ) is a major problem in mobile devices. JOJ describes a phenomenon that JBD of EXT4 filesystem logging SQLite journal data, wasting not only the space in storage but also degrades the performance by processing many synchronous writes. In this work, we analyze Tizen, cross-device platform, to understand the pros and cons of the platform in supporting better IO performance across heterogeneous systems.

IV. EXPERTIMENT

A. Mobile Storage Analyzer (MOST)

MOST is a tool to analyze the IO characteristics of Android platform [2]. In general, block device layers do not keep application level information. MOST is a *blktrace* [10] based IO tracing tool with additional features such as reverse mapping (finding an *inode* for a given block) and retrospective reverse mapping (finding an *inode* for a deleted file). MOST is also capable of categorizing each IO to its file types such as filesystem data, metadata, and journal. With the help of cross-layer IO tracking features, MOST makes possible to obtain more comprehensive understanding of the complex interaction between the application and their IO behaviors.

B. Workload

We managed to establish seven categories in smartphone apps and select total of 14 scenarios using seven applications. The scenarios allow evaluating and understanding the IO behavior in different situations. Although Tizen does not have as many applications as Android or iOS, some of the applications such as 'contacts' and 'mail' are packaged with Tizen and the rest are developed for HTML5 compatible platform; thus it is running on browser. Table 3 summarizes the scenarios and actions we conducted while tracing the applications.

TABLE II. SPECFICATION OF TIZEN REFERENCE SMARTPHONE

Processor	Samsung Exynos 4412 Quad-core 1.4GHz Cortex-A9
Storage	KMVTU000LM-B503 eMMC (16 GB)
Memory	K3PE7E700M-XGC2 Mobile-DDR2 SDRAM (1GB)
Display	4.8 inch AMOLED 16M colors. 720 X 1280 pixels
Sensors	Accelerometer, Gyro, Proximity, Magnetic etc.
Camera	Primary 8MP, Secondary 1.9MP

TABLE III. STORAGE PARTITION INTERNAL EMMC (16 GB) STROAGE

Mount Point	*Purpose*	*Filesystem*	*Capacity*
/csa	CSA[a]	EXT4	8 MB
/boot	kernel, boot-loader, modem	EXT4	60 MB
Not mount	reserved	EXT4	100 MB
Not mount	CSC[b]	EXT4	150 MB
/	Platform, Linux utility	EXT4	1.5 GB
/opt, /var	data	EXT4	3 GB
/opt/user	user media space	EXT4	10 GB

[a.] A (Configuration Saved Area) is for calibration of modem, [b.]CSC (Customer Software Configuration) can store the customer's software configuration such as default language, time zone.

C. Expertiment Setup

We use Tizen reference smartphone named RD-PQ, which runs Linux kernel 3.0 on Tizen 2.2.1 Platform. The device specification is described in Table 1. RD-PQ has a built-in 16 GB eMMC storage with seven EXT4 filesystem partitions, which are described in Table 2. We capture IO trace of the platform (/) and user partition (/opt, opt/usr).

D. Experiment Procedure

Figure 2 illustrates the procedure of IO workload analysis. We followed workload scenarios as described in Table 2, and collected traces on a separate partition in real-time. all running applications are terminated before capturing IOs of only selected application. User Application request an event for the their services, then Tizen core handles it with framework API and

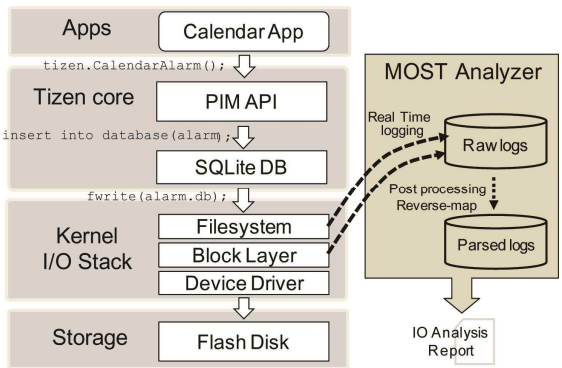

Fig. 2. IO analysis procedure

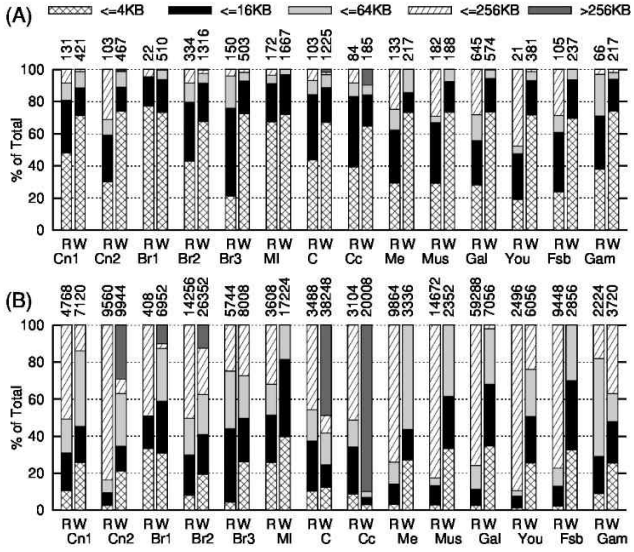

Fig. 3. IO Size distribution by IO count (A), sector count (B)

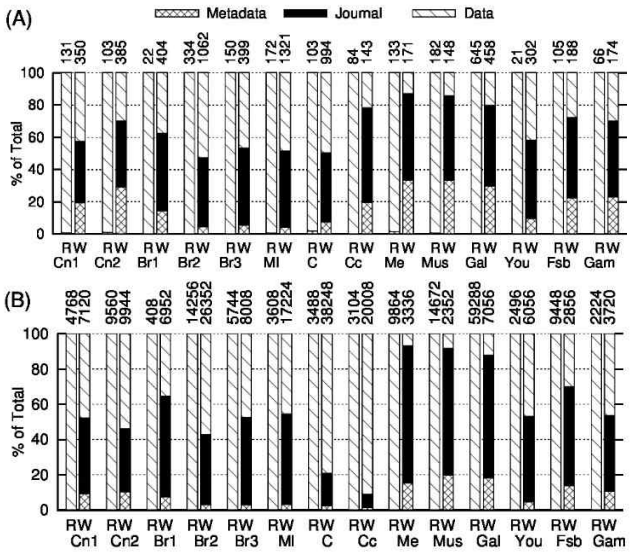

Fig. 4. *Block semantics distribution by IO count (A), sector count (B)*

library. If the event is needed to update DB, then the library makes a file access request to stores a data. Finally, MOST capture the IO event at kernel IO stack. We collected first 60 seconds of each scenario. For comparison of workloads, we normalized the raw data and illustrated total sum of respective information.

V. RESULT

We analyze 14 scenarios and present their IO characteristics. We illustrate the analyzed result of each scenario with respect to their size distribution, block level semantics, file type distribution, IO randomness, and IO mode type. We measure the number of IO and requested sectors size (extent) for each scenario as well. The number at top of the each bar of graphs indicates IO count or sector (512 Byte) count. 'R', 'W' denotes Read and Write, respectively.

TABLE IV. FILETYPE CATEGORIZATION FOR IO ANALYSIS

File type	Extension name
Executable	.tpk[c], .wgt[d], .exe, .so
SQLite	SQLite DB (.db)
SQLite-temp	SQLite-temp(.db-journal)
Multimedia	.mp4, .jpg, .mp3, .png, etc
Resources	.dat, .xml, .js, .cache, etc
Others	Others including directory entry

[c]-Tizen native app package (.tpk), [d]-Tizen web app package (.wgt)

Figure 3 shows IO size distribution with respect to IO counts. The result shows that 4 KB is the most common size, which is the minimum block size of EXT4 filesystem. We find that 4KB IO constitutes 45% of the total IO count. Figure 4 shows IO size distribution by sector counts statistics. The average IO size is 21 KB. In Camcorders (C) and Cameras (Cc) scenarios, 512 KB write IOs constitute about 60% to 80% of total sector.

Figure 4 illustrates the number of IOs of each block types. The workloads are classified into three categories (*Data*, *Metadata,* and *Journal*) in block device layer. In read operation, most of block types are in *Data* category. On average, 48% of write IOs are issued by *Journal*. *Metadata* and *Journal* generates most of the write IOs; number and size of both categories sums up to 66.4 % and 57 %, respectively. Excepting media Data access in Camera and multimedia applications, more than two thirds are for storing *Metadata* and *Journal* in block device.

Figure 5 illustrates file type distribution in filesystem layer with respect to IO and sector count. The result provides insight to how applications utilize the files and to understand the correlation between file types. We categorize files into six groups depending on their file extensions, which are described

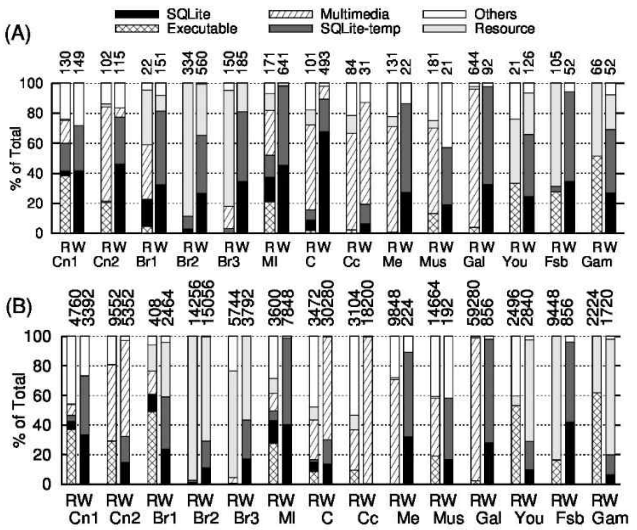

Fig. 5. File type distribution by IO count (A), sector count (B). Note that the number does not count the IOs for storing metadata and journals of filesystem layer.

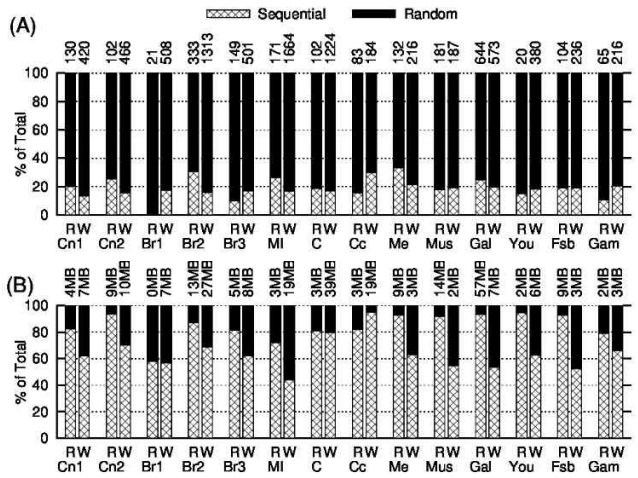

Fig. 6. Randomness by IO count (A), sector count (B)

in Table 4. File type categories are as follows: *Executable, SQLite, SQLite-temp, multimedia, resources*, and *others*.

In terms of the IO count of file types, SQLite and SQLite-temp combined is dominant with average of 75%. The volume of IO of SQLite and SQLite-temp is 54% on the average (Figure 5 (B)). Note that volume of SQLite is not as large as one can expect from the result of IO count because most of SQLite IOs are 4KB in size (about 73% of all SQLite IOs are 4KB in size).

Particularly, Camera scenario shows that over 80% of all IOs are induced by SQLite. We find that most of the requests are for updating `media.db` file. Database generates seven to twelve IOs to update the .db file.

Figure 6 describes the ratio of random and sequential IO with respect to IO count and sectors (extent). The result shows that 86% of all IO counts are random access IOs and 78% of all sector counts are for sequential accesses. There is no remarkable difference between read and write IOs. The result shows that random access is dominant in terms of IO count, and sequential access is a dominant in terms of sector count.

Figure 7 shows percentage of buffered IO and synchronous IO modes of write IO requests. The result only shows the fraction of IO modes of *data* block types only because volume of *metadata* is insignificant and *journal* uses synchronous mode. More than 90% of write are treated with synchronous mode, and only 9.7% are processed with buffered IO mode. It is because filesystem regards SQLite journal as data.

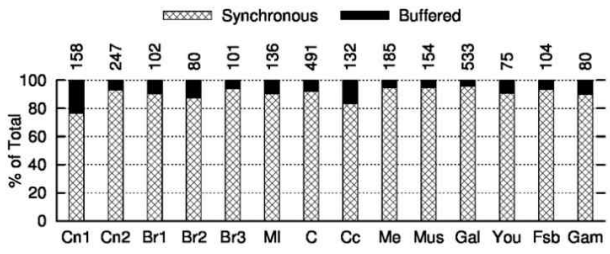

Fig. 7. Buffered vs. Synchronous IO mode distribution

VI. DISSCUISION

We find that most of the write IOs are generated by Journaling and metadata. Since mobile devices are prone to power failures, it seems appropriate to exploit synchronous IO mode. To guarantee the integrity of the data, Tizen uses SQLite as default database and EXT4 journaling filesystem. Studies have shown that such combination leads to performance degradation because of inefficiency caused by journaling of journal phenomenon

Tizen has a manager called RUA (Recently Used Application), which is a system application that is called at the launch of an application. After analyzing the write IO counts of SQLite, we observe that accesses to `rua.db` file accounts about 8% of total SQLite write IO count. At launch of each application, RUA manager generates two accesses to .db and four accesses to .db-journal. Note that filesystem writes journal data of each access to .db and .db-journal, which sums to 18 writes in total. We believe that DB management in RUA management needs much attention.

VII. CONCLUSION

In this paper, we analyze IO characteristics of several scenarios on Tizen platform. We find that IO behavior of Tizen is similar to that of Android platform, and Tizen also falls short on journaling of journal problem that generates excessive metadata and journal IOs— 66.4% of total write IOs are for metadata and journal. SQLite related file types constitute on the average of 75% of all file types. 85% of all IOs are random accesses, and 90% of all the writes are in synchronous mode. We also observe that 45% of all write IOs have 4KB in size.

REFERENCES

[1] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting Storage for Smartphones," Trans. Storage, vol. 8, pp. 14:1--14:25, 2012.

[2] K. Lee and Y. Won, "Smart Layers and Dumb Result: IO Characterization of an Android-based Smartphone," in Proceedings of the Tenth ACM International Conference on Embedded Software, ed. New York, NY, USA: ACM, 2012, pp. 23-32.

[3] S. Jeong, K. Lee, S. Lee, S. Son, and Y. Won, "I/O Stack Optimization for Smartphones," in Proceedings of the 2013 USENIX Conference on Annual Technical Conference, ed. Berkeley, CA, USA: USENIX Association, 2013, pp. 309-320.

[4] M. S. v. Bennewitz, "Chances for a tizen smartphone entry," Linux Journal, vol. 2013, p. 5, 2013.

[5] H. Jaygarl, C. Luo, Y. Kim, E. Choi, K. Bradwick, and others, "Professional Tizen Application Development," p. 526, 2014.

[6] Y. Park, "An Overview of Tizen Application Core Framework," in TIZEN Developer Conference, ed, 2012.

[7] Apple. (2014, May). About the iOS Technologies. Available: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

[8] Creative Commons Attribution. (2014, May). Introduction to Android. Available: https://developer.android.com/guide/index.html

[9] mobiThinking. (2014, May). Global mobile statistics 2014 Part A: Mobile subscribers; handset market share; mobile operators. Available: http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#mobiletablet

[10] A. D. Brunelle, "Blktrace user guide," ed: USA, 2007.