# IO characteristics of modern smartphone platform: Android vs. Tizen

Myungsik Kim, Hu-Ung Lee, and Youjip Won
Department of Computer Science Engineering, Hanyang University, Seoul, Korea.
{mskim77, oihtoto, yjwon}@hanyang.ac.kr

***Abstract*—In this work, we examine the IO characteristics of two smartphone platforms: Android and Tizen. We compare how the two platforms differ in using the filesystem, using the database, and accessing the storage. We collected the IOs from select seven different apps that are commonly available on both platforms. By analyzing the collected the IOs on both platforms, a dominant fraction of writes are 4 KB synchronous. On Android and Tizen, respectively, 65% and 45% of all write IO counts 4 KB; 47% and 75% of all writes accesses are SQLite related files such as database, rollback journal, WAL (Write Ahead Log), and etc.; 81% and 86% of all IOs are random accesses; and 90% and 52% of all writes counts are synchronous IO mode. According to our IO analysis, Android IO stack is more sophisticated then Tizen one: (i) disables *atime* updates, (ii) adopts *journal asynchronous commit*, and (iii) adopts `fdatasync()` in SQLite.**

***Index Terms*—Android, Tizen, Mobile Platform, Smartphone, IO Workload Characterization**

## I. Introduction

Storage IO is arguably the major performance bottleneck in smartphones [1]. It is found that Android apps generate an excessive amount of synchronous IOs most of which are from EXT4 journal writes [2]. In Android, misaligned interaction between SQLite rollback journal and EXT4 journaling layer creates excessive writes to filesystem journal [3]. Improper handling of platform IO requests can worsen the essential properties in mobile devices: it can decrease the devices' overall performance, can shorten NAND flash cell lifetime in eMMC, and can decrease energy efficiency which is essential in mobile devices that have limited battery power.

This study is motivated by the needs to compare the efficiencies of the IO stacks in the open source based smartphone platforms: Android and Tizen. While the IO characteristics of Android are relatively well known [1], [3], little is known about the details of Tizen platform [4]. We focus our effort in characterizing the IO behaviors of emerging platform by comparing one of the most widely used smartphone platforms currently available. We investigate IO characteristics of Tizen and Android and identify areas for improvement by comparing the two platforms.

To perform empirical study on the IO characteristics of two platforms, we establish seven categories for apps. From these seven app categories, we select fourteen workloads. For fair comparison, we select built-in basic application categories that are available on both platforms such as contacts, calendar, web browser, etc. We collect IOs while performing selected workloads and analyze the characteristics of the collected IOs.

978-1-4799-5344-8/15/$31.00 

In terms of IO size, 4 KB is the dominant, accounting for 65% and 45% of total IO count on Android and Tizen, respectively. The proportion of 4 KB IOs is 20% higher on Android than on Tizen. In terms of IO block type, metadata and journal account for 37.3% and 57% of the total IO size on Android and Tizen, respectively. Tizen has higher fraction of journal and metadata than Android has. In terms of file type, SQLite related files are common on both platforms. On Android and Tizen, SQLite related files account for 54% and 40% of all IO volume, respectively. Random write is dominant: 81% and 86% of all accesses are random in Android and Tizen, respectively. On both Android and Tizen, 78% of all IOs are sequential in terms of volume. On Tizen, 90% of all IOs are synchronous writes; Android shows less synchronous writes with 51.8%. We observe that Android use more buffered writes than Tizen.

The IO behavior of the two platforms are different due to their database structures and journaling options. The reasons for such write amplification are: (i) redundant data updates from platform components, (ii) unorganized database structure, and (iii) duplicated journaling by SQLite DB and EXT4 filesystem known as 'journaling of journal' anomaly [2], [3]. We observed that Android uses additional IO stack optimization options for filesystem and SQLite. Examples of the optimization options include EXT4 filesystem's *noatime* and *journal async commit* option. In addition, Android further reduces the number of IOs by using WAL journaling mode and `fdatasync()` for SQLite. On the other hand, Tizen does not use those IO stack optimization options. Tizen uses the default EXT4 mounting options and PERSIST rollback journal mode is used for SQLite. Adopting Android's tuning options on Tizen may improve its IO performance.

## II. Background

### A. Platform Architecture

Android and Tizen platforms are software stacks built on top of Linux kernel. The mobile platforms include various middleware components which consist of proven open-source solutions, such as WebKit, SQLite, SSL, libc, GStreamer, etc. Both platform frameworks are loaded with core applications, such as web browser, email, address book, image viewer, etc. They also provide an API to app developers.

TABLE I: Specifications on Tizen and Android Smartphones

| Platform | Android | Tizen |
|---|---|---|
| Model Name | Galaxy S3 | RD-PQ |
| Processor | Samsung Exynos 4412 | |
| CORE | Quad-core 1.4GHz Cortex-A9 | |
| RAM | 1 GB Mobile DDR2 | |
| Display | 4.8 inch HD AMOLED (1280x720) | |
| Sensors | Gyro, Proximity, Compass, Barometer | |
| Storage | eMMC 32 GB | eMMC 16 GB |
| Kernel | Linux 3.0.31 | Linux 3.0.15 |
| Platform | Android 4.1.2 | Tizen 2.2.1 |
| SQLite | 3.7.14 | 3.7.13 |

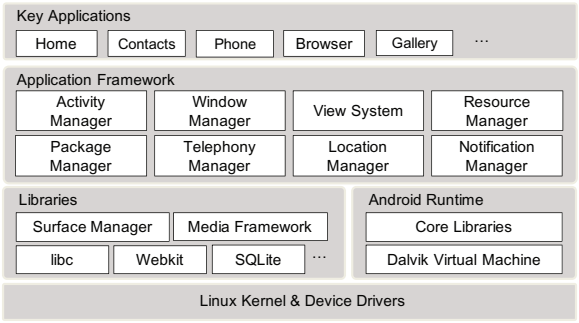
There is a difference in their application frameworks: Android uses specific Java runtime engine, Dalvik VM, for their application framework, whereas Tizen does not use a virtual machine. Instead, Tizen provides broader range of mobile app packaging technology. Tizen includes web-based framework and web SDK for web app which uses HTML5 [5], CSS, and Java script [6]. There is a tradeoffs between the two strategies. A virtual machine can provide enhanced system safety and security, but it may yield a larger runtime processing overheads. Web app has more flexibility based on inter-operable Web standard [7].

Tizen adopts device profile concept [8] which enables the platform to be used across multiple devices. Tizen is allows additional hardware components to be easily integrated. Through the cross-device platform concept, newer Tizen targets not only mobile devices but also in-vehicle infotainment (IVI) devices, smart cameras [9], wearable devices [10], and other consumer electronic devices. Android and Tizen have Webkit-based Chromium browser as their default browser [7]. Webkit is an open source web layout engine that is responsible for rendering web contents onto a screen. Android uses Webkit [11] and Tizen uses Webkit2 [12], [13] ).

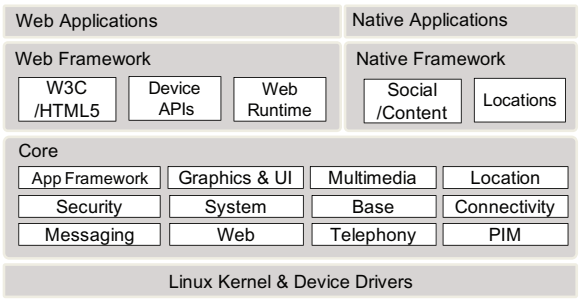
Tizen follows kernel mainline policy which is governed by the Linux Foundation. In contrast, Android patches the mainline Linux kernel especially to improve embedded devices. For example, Android has its own Inter-process communication option (Binder, Lightweight RPC to minimize memory copying overhead by referencing shared memory in kernel), memory management subsystem (Ashmem to less data has to be transferred), C library (bioniC, lightweight than glibc), debugging tools (logcat, Android Debug Bridge), etc.

### B. IO Stack

In both Android and Tizen platforms, SQLite is the way of managing the data in persistent manner. The SQLite uses `fsync()` to make the result of database transaction persistent [2], [3]. The performance of SQLite transaction relies heavily on the `fsync()` transaction. `fsync()` writes dirty file blocks to the respective location and then journals the respective metadata. Both Android and Tizen use EXT4 filesystem as their default filesystem. They use "Ordered Mode" journaling where only the metadata is journaled.




(a) Android Platform Architecture [14], [15]




(b) Tizen Platform Architecture [16]

Fig. 1: Platform Comparison between Android and Tizen

Different Android smartphone models adopts different block IO scheduler: Deadline (Google Nexus 5), BFQ (Budget Fair Queuing, Google Nexus One), and CFQ (Samsung Galaxy Nexus, Samsung Nexus S) [17]. Tizen platform offers three IO scheduling options: CFQ, Deadline, and Noop. Both Android and Tizen devices in this study use CFQ (Completely Fair Queuing) IO scheduler as the default option.

## III. Data Study

### A. Device

For this study, we use Tizen reference smartphone, RD-PQ, and Android smartphone, Galaxy S3. Table I summarizes specifications of the two systems. The two hardware platforms are identical. We use Galaxy S3 even though there are newer Android devices available because the two platforms share the same hardware platform. Tizen reference smartphone, RD-PQ, runs Linux Kernel 3.0. Tizen adopts traditional Linux kernel directory structure such as `/var` or `/opt`. Android uses modified Linux directory structure with added `/system` and `/data` directories. The program resources are generally located in `/usr/bin` in mainstream kernel, but in Android, they are in `/system/bin`. Both devices are initialized with stock OS image downloaded from the manufacturer's web site. While Android platform enjoys a plethora of prepackaged apps, Tizen phone yet comes with only limited number of apps. For this study, we install additional 38 apps to our Tizen device which includes in SDK package.

### B. Filesystem and Storage

Metadata update overhead constitutes a significant portion of IO traffic in Android based smartphones [2]. EXT4 inode has `atime` field. It is updated every time when the file is accessed. The updates in the `atime` make the inode dirty and makes it subject to filesystem journaling. EXT4 provides *noatime* option. When *noatime* is set, `atime` field is not updated when a file is in *read* operation. In default, Android platform deliberately disables the `atime` updates to reduce the IO traffic. In Tizen based smartphone in this study, `atime` field is updated every time when a file is accessed.

In order for EXT4 journal to function correctly, the order between the journal descriptor blocks and the journal commit block [18] needs to be preserved. EXT4 filesystem provides an option for *asynchronous journal commit*. In *asynchronous journal commit* option enabled, the `fsync()` call returns without waiting for the write completion of journal commit block. The journal commit block contains checksum field which is obtained from the journal descriptor block and the commit block itself to determine whether the commit block has been properly recorded, preserving the order between journal descriptor blocks and journal commit block. Android platform uses *journal asynchronous commit* option. Tizen does not use this option. According to previous study [19], with *journal asynchronous commit* option, the `fsync()` performance increases by up to 7%.

TABLE II: Comparison of EXT4 Filesystem Mount Option

| Android EXT4 mount option |
|---|
| /system rw,noatime,barrier=1<br>/cache nosuid,nodev,noatime,barrier=1,journal_async_commit<br>/data relatime,barrier=1 |

| Tizen EXT4 mount option |
|---|
| /opt relatime,user_xattr,barrier=1<br>/var relatime,user_xattr,barrier=1<br>/opt/usr relatime,user_xattr,barrier=1 |

### C. Workloads

We select seven app categories from Android and Tizen. Selected categories are contacts, web browser (visiting 3 different web sites), mail, camera (with video recorder), multimedia (playing music and video), gallery, web streaming (playing a video clip on youtube), and HTML5 (benchmark site). We like to understand the IO behaviors of these apps in the practical settings. For this reason, instead of using benchmark, we perform human-based experiment. From seven categories, we select total of fourteen applications in this study. For each of fourteen usage scenarios, the user runs the app for one minute. Mobile devices use Dynamic Voltage and Frequency Scaling to minimize the power consumption. As the CPU clock cycle may affects the IO performance, we fix the operating frequency to the maximum level. We use MOST (Mobile Storage Analyzer) [2], [20] to collect IOs. The experiment is repeated three times to warrant correctness of the study. Table III summarizes the workloads and user application scenarios for tracing IOs.

In this study, we collect IOs and analyze six IO attributes: IO sizes, IO type (read or write), spatial locality (random vs. sequential), filesystem block type (data block, metadata block, journal block), and synchronization mode (buffered vs. synchronous IO). We categorize IOs in the filesystem perspective into three blocks types: data block, metadata blocks (e.g., inodes, bitmaps, directories, group descriptors), and journal block (EXT4 journal). We categorize the files into six types by file extension name: executables (`.apk` (Android application package), `.odex` (Optimized Dalvik Executable), `.tpk` (Tizen native package), `.wgt` (Tizen web app package), `.exe`, `.so`), SQLite database table (`.db`), SQLite temporary files (`.db-journal`, `.db-wal`, `.db-shm`, `.db-mjx`), multimedia (`mp4`, `.jpg`, `.mp3`, `.png`, `.jpeg`, `.png`, `.3gp`), resource (`.dat`, `.xml`, `.js`, `.cache`, `.info`, `.dat`, `.tmp`), and others including directory entry.

## IV. Primitive IO

### A. IO Size

On both platforms, 4 KByte IOs are dominant. In Android and Tizen, 4 KByte IOs accounts for 65% and 45% of all write count, respectively. The fraction of 4 KByte IOs is higher on Android platform than on Tizen. We categorize IOs into five groups based on their size ( $\leq$4 KB, $\leq$ 16 KB, $\leq$ 64 KB, $\leq$ 256 KB, and more than 256 KB.) and identify the fraction of each group. Fig. 2 illustrates the result. Fig. 2(a) and Fig. 2(b) show the IO size distribution with respect to IO counts for Android and Tizen, respectively.

Android shows particularly high number of write IOs while executing web application compared to other workloads. The IOs from web application are generated by storing caching data. On Tizen, mail workload generates the highest number of IO writes with 1,667 in Fig. 2(b). The reason for this excessive number of IOs is the write requests from the database. On both platforms, media contents are saved using a small number of large size buffered writes. Multimedia contents are stored through a buffered write with large IO extents. In Camcorder (Cc) and Camera (C) scenarios, 512 KB write IOs constitute about 80% and 60% of the total incurred sector size in Android and Tizen, respectively.

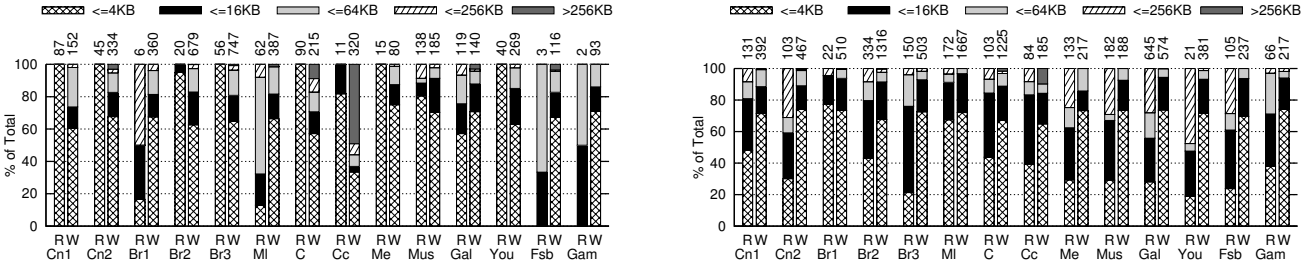
### B. Block Type

We analyze the IO access patterns in terms of filesystem block types. We categorized the IOs into three block types; metadata, journal, and data. Fig.3 illustrates the result. In Android and Tizen, metadata IO and journal IO combined together accounts for 45% and 66% of the total write IO counts, respectively. Filesystem journaling and metadata updates are managerial operations.

Tizen creates more IO overheads than Android does. In performing an identical tasks, Tizen apps yields much than Android does. The 80% of writes are metadata and journal writes on Tizen when executing gallery app workload, compared to 47% on Android. The amount of metadata and journal writes depends on the synchronous write requests from an application. We observe that Tizen does not employ options

TABLE III: Scenarios

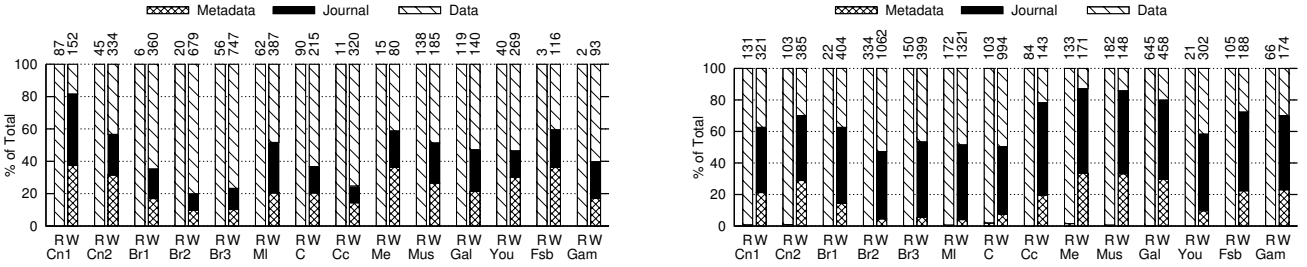
| Category | Workload | Scenario |
|---|---|---|
| Contacts | contact1(Cn1), contact2(Cn2) | Add a name and a phone number in the address book. |
| Web | google(Br1), daum(Br2), naver(Br3) | Access web sites and portal news service using web browser |
| Mail | mail(Ml) | Check e-mail on mail app, write e-mails, and send e-mails. |
| Camera | camera(C), camcorder(Cc) | Taking a picture and Recording a video. |
| Multimedia | media(Me), music(Mus), gallery(Gal) | Play videos, play music files, view pictures. |
| Web streaming | youtube(You) | Play videos on YouTube using web browser |
| HTML5 | fishbowl(Fsb), game(Gam) | Access two HTML5 benchmark sites. |




(a) IO Count: Android

(b) IO Count: Tizen

Fig. 2: IO Distribution




(a) IO Count: Android

(b) IO Count: Tizen

Fig. 3: Filesystem Block Type

that can potentially reduce a synchronous IOs such as SQLite *fdatasync* option or WAL journaling mode. Due to all these causes, in the gallery app of Tizen, journal writes and metadata writes combined together account for 80% of entire writes.

### C. File Type

We categorize the files into six groups depending on their file extensions: executable, SQLite, SQLite-temp, multimedia, resources, and others as mentioned in Section III-C. SQLite-temp files are temporary files that SQLite generates in order to implement atomic commit and rollback capabilities. These temporary files have `.db-journal` or `.db-wal` extensions. Fig. 4 illustrates file type distribution in filesystem layer by IO count and sector count. On Android, SQLite writes account for 47% of the total IO counts, which are lower than the portions on Tizen. On Tizen, SQLite and SQLite-temp, combined, accounts for 75% of all IO counts (Fig. 4(b)). The portion of the two file types is lower in terms of the total size than the total IO count because most of SQLite IOs are 4 KB in size (about 73% of all SQLite IOs are 4 KB). Particularly, with camera workload, over 80% of all IOs are induced by SQLite. In case of mail workload, most IOs are DB file accesses on both platforms. In performing an identical tasks Tizen Application generates much larger fraction of SQLite related IO than Android application does.

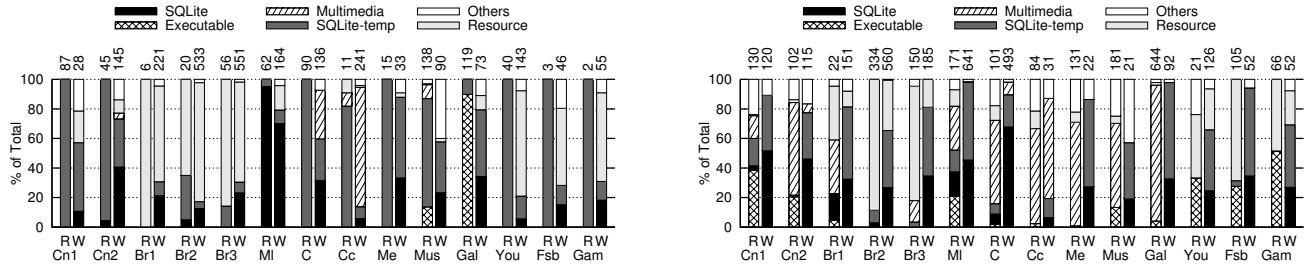
### D. Buffered vs. Synchronous Write

We categorize the IOs into synchronization mode (buffered vs. synchronous IO). Fig. 5 shows that Tizen platform generates much larger fraction of synchronous writes than Android platform does. In Android and Tizen, synchronous writes accounts for 51.8% and 90.3% of the total writes IO counts, respectively. Tizen do not use buffered IO sufficiently, it only exist 9.7% of total write IO counts.

Synchronous write operation is a heavy operation. It blocks the process until the requested data blocks reach the storage device. For the performance's sake, the application developers use synchronous writes only when there is no other resort. According to this analysis, the Tizen IO stack leaves much room for further improvement to reduce the fraction of synchronous writes.

TABLE IV: IO Characterization Comparison Summary

| IO Semantics | Sub-Type | Android | Tizen |
|---|---|---|---|
| IO Size | IO Size 4 KB | 65% | 45% |
| | Block type Size | Data> Journal >Meta | Data> Journal >Meta |
| Block type | Meta and Journal | 45.1%(c) 37.3%(s) | 66.4%(c) 57%(s) |
| File type | SQLite and SQLite temp | 47.6%(c) 40.2%(s) | 75%(c)54%(s) |
| Sequentially | Random Count / Sequential Size | 81.4% / 78% | 86% / 78% |
| Synchronous/Buffered IO | Synchronous Write Count | 51.8% | 90.3% |




(a) IO Count: Android

(b) IO Count: Tizen

Fig. 4: File Type Analysis

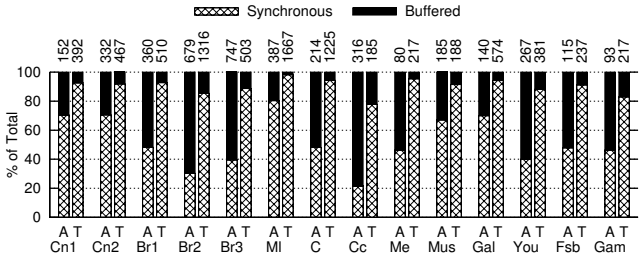



Fig. 5: Process IO Distribution (A:Android, T:Tizen)

## V. Related Work

There have been researches on optimizing performance of smartphones. Research areas include IO workload analysis and optimization of mobile devices. Kim et al. [1] presented IO characteristics of Android smartphones and pointed out that storage IO behavior can affect the application performance. Wi-Fi network speed can change the storage performance by up to three times, depending on the choice of applications. Ouarnoughi et al. analyzed the IO performance on embedded DB which focused on flash memory filesystem [21]. Lee et al. [2] showed that 70% of all writes are random on the Android platform and that SQLite and EXT4 generate excessive journaling IOs.

SQLite creates journal files that are used in case of data crash and these journal files are journaled in the filesystem, incurring additional IOs. Decreasing this journaling IO overhead in SQLite DB has been researched. Jeong et al. [3] analyzed the journaling IO overhead of Android and also proposed to optimize the IO stack by using `fdatasync()` mode, WAL journal mode, polling based I/O, and alternative filesystem (flash-friendly filesystem, F2FS [22]).

Other studies attempted to decrease IO overhead by processing transactional atomicity in the storage's flash translation layer (FTL) [23] or converting the structure of SQLite's index tree from B-tree to LS-MVBT (Lazy Split Multi Version B-Tree) [24]. LS-MVBT saves recovery information in the DB file itself, not in a journal file, and uses the version information in case of system failure. This eliminates unnecessary synchronization IOs that are related to recovery.

There have been numerous research works on Android IO [1]–[3] but only limited introduction research has been done on emerging Tizen platform. The existing research on Tizen IO [25] only points out that the platform presents the same journaling problems as Android. To the best of our knowledge, there are no comparative analysis for two competing platforms, Android and Tizen, regarding IO characteristics. Our study substantially different from the existing research in that we find possibilities of the improvement and optimization by executing platform comparison.

## VI. Discussion and Summary

In this work, we analyze the IO characteristics on Android and Tizen platforms. Of the total number of IOs, 65% and 45% are 4 KB in size; 81.4% and 86% are random IOs; and 51.8% and 90% are synchronous writes on Android and Tizen, respectively. In both platforms, the amount of filesystem data blocks written to the storage device is smaller than the amount of filesystem journal blocks and filesystem metadata blocks. The filesystem journal and metadata account for 57% and 66.4% of the total writes on Android and Tizen, respectively. SQLite related writes account for dominant fraction of storage writes. On Android and Tizen, 54% and 75% of IOs, respectively, are SQLite DB related. Although SQLite is known to be lightweight, it generates excessive amount of

synchronous write IOs to update DB journal files and EXT4 journal. Table IV shows the summary of the IO characteristics in two platforms.

Android IO stack, as it currently stands, is more mature than Tizen IO stack. We confirmed that Tizen IO stack is not mature than Android, through comparison of IO attributes. On Tizen, only PERSIST mode is used for SQLite journaling. `fdatasync()` calls for selectively synchronization represent only 2% of the total number of `fsync()` calls in Tizen. To achieve better IO on Tizen, Tizen should also reflect the Android's IO stack optimization points. The IO performance of Tizen can be improved by replacing `fsync()` with `fdatasync()`, optimizing EXT4 filesystem mounting options, and utilizing other SQLite journaling modes, including WAL mode. From this comparative study, IO characteristics identified in the findings can provide directions of improvement IO stack of emerging mobile platforms.

## VII. Acknowledgment

This work was sponsored by IT R&D program MKE/KEIT (No.10041608, Embedded system Software for New-memory based Smart Device). This research was supported by the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015- H8501-15-1006) supervised by the IITP (Institute for Information&communications Technology Promotion)

## References

[1] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *Trans. Storage*, vol. 8, no. 4, pp. 14:1–14:25, Dec. 2012. [Online]. Available: http://doi.acm.org/10.1145/2385603.2385607

[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*, ser. EMSOFT '12. New York, NY, USA: ACM, 2012, pp. 23–32. [Online]. Available: http://doi.acm.org/10.1145/2380356.2380367

[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*, ser. USENIX ATC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 309–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2535461.2535499

[4] J. Yap. (2013) Tizen OS needs differentiation, clear roadmap. [Online]. Available: http://www.zdnet.com/article/tizen-os-needs-differentiation-clear-roadmap/

[5] M. B. Hoy, "HTML5: a new standard for the Web," *Medical reference services quarterly*, vol. 30, no. 1, pp. 50–55, 2011.

[6] O. Gadyatskaya, F. Massacci, and Y. Zhauniarovich, "Security in the firefox OS and tizen mobile platforms," *Computer*, vol. 47, no. 6, pp. 57–63, June 2014.

[7] E. A. Hernandez, "War of the mobile browsers," *IEEE Pervasive computing*, vol. 8, no. 1, p. 0082, 2009.

[8] N. Willis. (2014) Tizen common and open hardware. [Online]. Available: http://lwn.net/Articles/617543/

[9] M. Brian. (2013) Samsung's NX300M smart camera is its first to run tizen OS. [Online]. Available: http://www.engadget.com/2013/11/11/samsungs-nx300m-mirrorless-camera

[10] V. Prabhakaran, "Samsung announced tizen-based gear 2 and gear 2 neo," 2014.

[11] J. J. Sánchez, "Webkit and blink: Open development powering the HTML5 revolution," in *LinuxCon 2013*. The Linux Foundation, 2013. [Online]. Available: http://www.slideshare.net/igalia/webkitblinklinuxcon2013

[12] J. Teixeira and T. Lin, "Collaboration in the open-source arena: The webkit case," in *Proceedings of the 52Nd ACM Conference on Computers and People Research*, ser. SIGSIM-CPR '14. New York, NY, USA: ACM, 2014, pp. 121–129. [Online]. Available: http://doi.acm.org/10.1145/2599990.2600009

[13] Y. Zhu and V. J. Reddi, "Webcore: Architectural support for mobileweb browsing," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 541–552, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2678373.2665749

[14] J. Liu and J. Yu, "Research on development of android applications," in *Intelligent Networks and Intelligent Systems (ICINIS), 2011 4th International Conference on*, Nov 2011, pp. 69–72.

[15] M. Butler, "Android: Changing the mobile landscape," *Pervasive Computing, IEEE*, vol. 10, no. 1, pp. 4–7, Jan 2011.

[16] Tizen.org. (2014, Nov) Reference Device-PQ. https://wiki.tizen.org/wiki/Reference_Device-PQ.

[17] D. T. Nguyen, G. Zhou, X. Qi, G. Peng, J. Zhao, T. Nguyen, and D. Le, "Storage-aware smartphone energy savings," in *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '13. New York, NY, USA: ACM, 2013, pp. 677–686. [Online]. Available: http://doi.acm.org/10.1145/2493432.2493505

[18] V. Chidambaram, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Optimistic Crash Consistency," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Farmington, PA, November 2013.

[19] H. Kim and J. Kim, "Tuning the ext4 filesystem performance for android-based smartphones," in *Frontiers in Computer Education*, ser. Advances in Intelligent and Soft Computing, S. Sambath and E. Zhu, Eds. Springer Berlin Heidelberg, 2012, vol. 133, pp. 745–752. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-27552-4_98

[20] S. Jeong, K. Lee, J. Hwang, S. Lee, and Y. Won, "Androstep: Android storage performance analysis tool." in *Software Engineering (Workshops)'13*, 2013, pp. 327–340.

[21] H. Ouarnoughi, J. Boukhobza, P. Olivier, L. Plassart, and L. Bellatreche, "Performance analysis and modeling of SQLite embedded databases on flash file systems," *Design Automation for Embedded Systems*, pp. 1–36, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10617-014-9149-2

[22] J. Kim. (2012) F2FS: introduce flash-friendly file system. http://lwn.net/Articles/518718/.

[23] W. Kang, S. Lee, B. Moon, G. Oh, and C. Min, "X-FTL: Transactional FTL for SQLite Databases," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 97–108. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465326

[24] W. Kim, B. Nam, D. Park, and Y. Won, "Resolving journaling of journal anomaly in android I/O: Multi-version B-tree with lazy split," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, ser. FAST'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 273–285. [Online]. Available: http://dl.acm.org/citation.cfm?id=2591305.2591332

[25] M. Kim, S. Lee, and Y. Won, "IO workload characterization comparison of Tizen based consumer electronics," in *Proceedings of IEEE International Symposium on Consumer Electronics, 2014*, 2014, pp. 11 822–6.