Dumb Design, Dumber Usage of Mobile Database

Myungsik Kim, Eunryoung Lim, Seongjin Lee, and Youjip Won
Department of Computer Science Engineering, Hanyang University, Seoul, Korea.
{mskim77|erlim|insight|yjwon}@hanyang.ac.kr

***Abstract*—In this work, we captured 36 days of I/O trace from NEXUS 5 and analyzed database usage of each application. We found that 70% of synchronous write I/Os are `snet_files_info.db`, `es0.db`, and `gmail.com.db`. There were 1533 records in `snet_files_info.db` that has to be updated for Google mobile service which creates write amplification of 6×10^6. Out of 62 tables in `es0.db`, only 2 tables were the most frequently used tables, but surprisingly the two tables share almost exact data. `mailstore.x@gmail.com.db` creates heavy synchronous writes during during checkpoint operation.**

***Index Terms*—Android, Mobile Platform, Smartphone, Mobile Database, Database I/O usage**

I. Introduction

Since the advent of the smartphone, usage of mobile devices have gradually increased its share in the computing market share, and to some extent it is considered as a representative personal computing device. A research shows that 85% of people feels that the mobile device is central to their everyday life and spends about 3.3 hours a day on their smartphones [1]. Although the number of everyday mobile device users are increasing, there are not many studies on common usage patterns of many mobile applications. It is important to understand the I/O behavior because I/O matters to life span of flash storage on mobile devices and performance of applications.

There are dozens of pre-installed user apps on a smartphone (e.g., NEXUS 5 has 29 Apps), and most of the other apps are provided by different 3^{rd} party developers. Also, it is well known fact that most of the data both user App and platform generated are stored to the storage using database [2], [3]. At first glance, we thought that applications would generate a lot of database related I/Os. As we have set out to explore the usage of database, surprisingly we found that it is not the applications that needs attention but Google related mobile services are highly inefficient in terms of database usage (especially, `es0.db` and `snet_files_info.db`).

II. Experiment Setup

We developed a real-time I/O collector to capture the I/O trace of Android based mobile devices, and captured the I/O trace on NEXUS 5 [4] starting from 2015-04-25 00:00 to 2015-05-31 24:00 (36 days). The developed I/O collector is extended version of MOST [2] with a server to store the captured data [3]. NEXUS 5 is equipped with 16 GByte of internal eMMC and runs Android Ver 4.4 KitKat on top of Linux Kernel Ver 3.4.0, and it also uses SQLite Ver 3.7.11 as default database. We used SQLite Analyzer [6], SQLiteman [7], and SQLite Database Browser [8] to analyzer SQLite databases structures and stored data. When we believe it is necessary, we captured the state of a database before and after a particular action of an application to have better understanding of what is going on with the application.

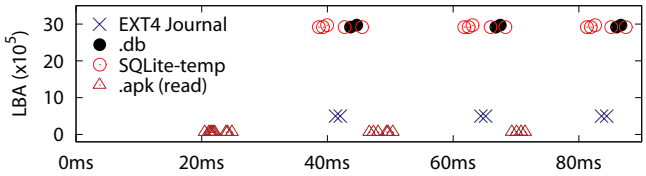

Fig. 1: I/O behavior of `snet_files_info.db` Update

III. Database I/O Analysis

We analyzed user I/O trace on NEXUS 5 that is captured during 2015-04-25 00:00 to 2015-05-31 24:00 (36 days). The total number and the sum of I/Os collected accounts to 7,196,131 and about 150 GByte of data, respectively. The number and the volume of synchronous write which we are interested in is 3,427,647 and about 39 GByte, respectively. If we removed metadata and file system journal writes from the synchronous write, there are 2,347,886 I/Os. We chose top three rank of databases that generated the most I/Os, which is shown in Table I. We analyze them to understand their usage pattern. Although there are a lot more to discuss about the tables, we point out few things only because the page is limited.

A. Case Study 1: `snet_files_info.db`

Purpose of `snet_files_info.db` is to keep track of list of files that needs to be used for Google mobile service, and the database file keeps list of 1,553 files and their updated time–`time_ms` field has size of 13 Byte. The part where database operates on each row of data to update a value is simple and straightforward, but the impact of such operation on storage can be easily overlooked. Fig. 1 shows the effect of three database I/O update pattern on `snet_files_info.db`. One database update creates 4 accesses to `.db`, 6 accesses to `.db-journal`, and 2 directory entry updates, and the volume it generates are 16 KByte, 28 KByte, and 16 KByte, respectively. Overall, update to 1553 rows of data results in 75 MByte that is write amplification of 6×10^6 to update $1553\times$ same 13 Byte. Note that, this database I/O accounts for 43% of all synchronous writes.

B. Case Study 2: `es0.db`

Social network service, Google+ uses `es0.db` to save various background metadata about user activities. `es0.db` consists of 62 tables–32 tables on the `.db` file did not receive any updates, and only 2 tables, `ALL_TILES` and `ALL_PHOTOS`, holds most of the data (68.2% and 27.1%, respectively). `ALL_TILES` and `ALL_PHOTOS` has 13 fields

TABLE I: Count and Size of I/Os of Frequently Updated Databases in Smartphone (×1000, Count/Size)

Database	Total I/O	4KB Write	Read I/O	Write I/O	Sync. Write	.db	.db-j or .wal
snet	1,015 / 4,547	903 / 3,613	1 / 9	0.04 / 0.18	1014 / 4538	401 / 1651	608 / 2828
es0	597 / 4,078	470 / 1,882	56 / 837	0.25 / 154	540 / 3,087	303 / 1,259	236 / 1,980
mail	121 / 1,444	87 / 349	26 / 272	1 / 15	94 / 1,157	75 / 353	13 / 714
All	7,196 / 150,720	3,595 / 14,381	3,047 / 72,244	720 / 39,414	3,427 / 39,062	943 / 4,990	1,145 / 6,804

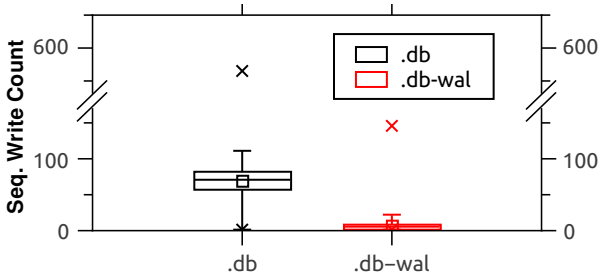

Fig. 2: I/O Count in `mailstore.x@gmail.com.db`

for storing metadata of pictures along with photo album indexes and 30 fields for storing another set of metadata of pictures, respectively. We observed 91 consecutive database updates at the most and on average there were 4 updates on the database file. We found it interesting to see that `ALL_TILES` and `ALL_PHOTOS` shares same information on most of the fields, but saves the exact same information on two different tables, which suggests that tables are not normalized properly.

C. Case Study 3: *mailstore.x@gmail.com.db*

`mailstore.x@gmail.com.db` is a database to manage received emails, which uses WAL SQLite journal mode. WAL file is known as the least I/O generating journal mode, but it potentially suffers from writing a lot of data when it is time to checkpoint the WAL file. As a result, `.db` file receives a lot more I/Os than `.db-wal` file. Our analysis shows that there are six time more I/Os observed on `.db` file (75,798) than `.db-wal` file (13,403). Fig. 2 shows a box plot of number of I/Os observed on two files at the time of SQLite checkpoint. It shows that `.db` file exhibits 14 times more I/Os than `.db-wal` file. On one occasion, `.db` file received 550 consecutive I/Os on checkpoint. Note that each writes are synchronously written to the device which might cause the device to lag to complete the checkpoint.

IV. Related work

Recent studies suggest that I/O stack of mobile devices are not optimized and needs significant attention [2], [3], [9]. Lee *et al.* [2] is the first to point out the problem of journaling of journal and Jeong *et al.* [3] and Kim *et al.* [9] pioneered to solve the problem by modifying the SQLite journaling mechanism. In this work, we further analyze the effect of database structure and application access behavior on the I/O subsystem.

V. Conclusion

In this paper, we implemented a real time I/O collector to capture 36 days of I/O trace on NEXUS 5. We found that only three databases account for about 70% of all synchronous write I/Os. We analyzed the structure and the usage of the databases and found number of things. First, `snet_files_info.db` updates 1533 records for Google mobile service which results in write amplification of six million. Second, `es0.db` keeps number of tables but only 2 tables are most frequently accessed, which generates significant number of I/Os. Third, when checkpoint occurs on `mailstore.x@gmail.com.db` as much as 550 I/Os can be generated to flush out the actual data user stored.

VI. Acknowledgment

This work was sponsored by IT R&D program MKE/KEIT (No.10041608, Embedded system Software for New-memory based Smart Device). This work was supported by the ICT R&D program of MSIP/IITP. [R0601-15-1063, Software Platform for ICT Equipments]

References

[1] salesforce.com. (2014) 2014 mobile behavior report. [Online]. Available: http://www.exacttarget.com/sites/exacttarget/files/deliverables/etmc-2014mobilebehaviorreport.pdf

[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] Google. (2015) Nexus 5. made for what matters. [Online]. Available: http://www.google.com/nexus/5/

[5] K. Kim, E. Lim, S. Lee, and Y. Won, "Real-time io trace analysis of smartphone users," in *In Proceedings of 2nd International Conference on Advances on Computer, Electrical and Electronic Engineering (ICACEEE 2014)*. International Journal of Information Technology Computer Science (IJITCS), 2014, pp. 7–8.

[6] S. Consortium. (2012) SQLite download page : A program to analyze how space is allocated inside an SQLite database file. [Online]. Available: https://www.sqlite.org/download.html

[7] P. Vaněk. (2015) sqliteman:SQLite3 admin and devel tool - sourceforge. [Online]. Available: http://sourceforge.net/projects/sqliteman/

[8] M. Piacentini. (2015) DB browser for SQLite. [Online]. Available: http://sqlitebrowser.org/

[9] 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