



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#) 

Master's Thesis

Selective Journaling for Resolving Journaling of
Journal anomaly

Byungeun Park

Department of Computer Science and Engineering

Graduate School of UNIST

2017

Selective Journaling for Resolving Journaling of Journal anomaly

Byungeun Park

Department of Computer Science and Engineering

Graduate School of UNIST

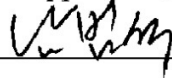
Selective Journaling for Resolving Journaling of Journal anomaly

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of science

Byungeun Park

12/ 14/ 2016

Approved by



Advisor

Beomseok Nam

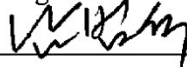
Selective Journaling for Resolving Journaling of Journal anomaly

Byungeun Park

This certifies that the thesis/dissertation of Byungeun Park is
approved.

12/14/2016

signature



Advisor: Beomseok Nam

signature



typed name: YOUNG-RI CHOI

signature



typed name: Woungki Baek

signature

typed name:

signature

typed name:

Abstraction

The mobile device low performance comes from the storage device. Decreasing I/O traffic compensate this defect. Latest researchers found that the relationship between SQLite database and filesystem is a cause of needless I/O traffic. This paper suggest Selective Journaling. Selective Journaling use transaction features to decide if SQLite use journal in file system or in SQLite. Test result shows that Selective Journaling is faster than original SQLite.

Contents

Chapter 1. Introduction	1
Chapter 2. Journaling of Journal	2
Chapter 3. Background Knowledge	3
3.1 SQLite	3
3.2 EXT4 File System.....	4
Chapter 4. Selective Journaling	5
4.1 Build in SQLite	5
4.2 Build in File System.....	6
Chapter 5. Evaluation	7
Chapter 6. Conclusion and future work	9

LIST OF FIGURES

Figure 1 Journaling of Journal.....	2
Figure 2 Selective Journaling	5
Figure 3 The performance result from 1000 transactions in each Insert, Update, and Delete query	7

Chapter 1. Introduction

As various mobile device become more common, many people prefer to use mobile device instead of PC. Therefore, many researches focus on mobile device performance. Mobile device had a bottleneck comes from the data transmission via network. However, better network performance remove this problem. Instead, storage performance is the hot issue for mobile device performance [1]. Storage performance is lower than any other component in PC. Mobile operating system makes this storage issue worse. Android OS control application I/O via SQLite for easy data management [2]. On the other hand, SQLite makes high traffic because of unnecessary writes. Many methods were suggested to solve this problem [4 - 10]. Most of these solutions solve the problem by changing file system data structure or using other hardware performance.

This research suggest Selective Journaling for reducing needless writes from the conflict between storage and SQLite. The performance is better than the original SQLite.

Chapter 2. Journaling of Journal

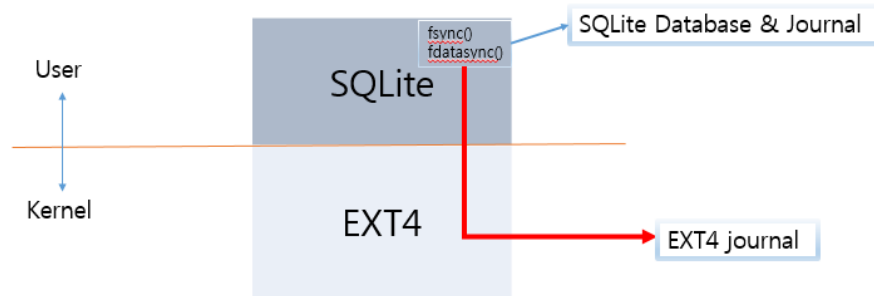


Figure 1 Journaling of Journal

Android mobile OS has a unique I/O stack. Traditional system use the system call write() or user level function. Instead, most of Android applications use SQLite for saving data. Database atomicity and consistency are maintained with journal or log. When journal and log created, file system create their recovery file because file system recognize them as a new files. This is called Journaling of journal(Figure 1) [3]. Journaling of journal build file system journal for SQLite recovery journal. Previous research shows that 100byte string saving requires at most 80 KB I/O. Journaling of Journal takes 40 KB in this I/O[6]. The Journaling of journal's unnecessary I/O traffic decrease the mobile device performance.

Chapter 3. Background Knowledge

3.1 SQLite

SQLite widely used because it is no server SQL with compact library. The embedded SQL database engine makes possible not to apply sever. Instead of this, SQLite reads and writes directly to the disk files.

The SQLite library size so small that it can be applied in many small gadgets such as smartphone, PDAs, and MP3 players. Full optioned SQLite is smaller than 500KB, and minimal SQLite size is 300KB.

SQLite has journal modes such as DELETE, TRUNCATE, MEMORY, PERSIST, and NONE are the options that decide the SQLite performance and safe transaction. Because NONE mode does not use any journal mode, it has the fastest performance but the absent recovery journal does not guaranteed the safe transaction. User can decide to use this but this mode is not recommended for reliable transaction in SQLite. The other unsafety journal mode is MEMORY mode which saves journal in volatile RAM. This saves disk I/O but there is no safety when application error happened. If the application using SQLite crashes in the middle of a transaction when the journaling mode is MEMORY, then the database file cannot be recovered.

Except none and memory modes, others have safe transaction. The DELETE journaling mode, which is the SQLite default mode, deleted the rollback journal at the end of each transaction. TRUNCATE journaling mode commits transactions by truncating the rollback journal to zero size instead of deleting it. The PERSIST journaling mode remain the recovery journal at the end of each transaction. PERSIST mode make journal header to zeros with overwritten. This will prevent other database connections from rolling the journal back. The PERSIST journaling mode is useful as an optimization on platforms where deleting or truncating a file is much more expensive than overwriting the first block of a file with zeros.

WAL mode, which use write-ahead log instead of rollback journal, has the best speed with safe transaction. The previous traditional rollback journal writes a copy of the original database content into rollback journal file. After the transaction is finished, writing add or modify data in the database file. When the corruption from the accident like power disconnected, the original content saved in the rollback journal recover the database file. Database files come back to its original state. The commit occurs when the rollback journal is deleted. The WAL mode inverts this traditional journaling process. The original content is remain in the database file and the changes are appended into a WAL file. A commit occurs when a special record indicating a commit is appended to the WAL. Thus a commit can happen without ever writing to the original database, which allows readers to continue operating from the original unaltered database while changes are simultaneously being committed into the WAL [12].

3.2 EXT4 File System

Filesystem controls how data is stored and retrieved. File systems can be used in any kind of storage device. Today's common storage device is a hard disk drive.

In this paper, the mount option about fourth extended filesystem (EXT4), which is a journaling file system for Linux, is shown to support Selective Journaling knowledge. EXT3 shows up the first file system journaling. Each of file system journal mode, Writeback, Orderd, Journal has their own tradeoff between performance and data corruption risk. Writeback has the best performance and the highest risk. Writeback mode create metadata journaling. The data will be written before or after the journal is updated. This lead file corruption when files modified right before a crash. Journal mode make metadata journaling and data journaling before being committed to the main file system. Because the journal is relatively continuous on disk, this can improve performance, if the journal has enough space. However, if the contents require the space more than the journal, performance gets worse. The shortage journal space cause of data write in the journal and the main part of the filesystem. Even though this performance problem, Journal mode ensure the data recovery against corruption.

Ordered mode performance and recovery data is in the middle of Writeback and Journal mode. It also create metadata journaling like Writeback mode, but it's guaranteed that file contents are written to disk before associated metadata is marked as committed in the journal. If there is a power shutdown or kernel error while a file is updated or written, the journal will indicate that the new file or appended data has not been committed, so it will be purged by the cleanup process. This allows new files have the same level of integrity protection as the journaled level. However, overwritten files can be corrupted because the original data is not stored [13].

Chapter 4. Selective Journaling

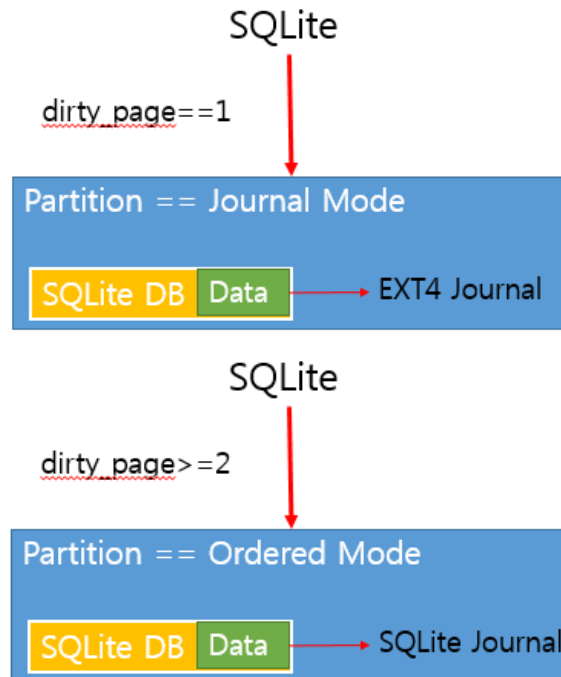


Figure 2 Selective Journaling

This paper suggest Selective Journaling to reduce the performance degradation. Selective Journaling choose either file system journal or SQLite journal. It depends on the number of changing pages. Selective Journaling use EXT4 file system because most of Android OS use EXT4. Selective Journaling gets ordered and journal mode from this file system. These modes are EXT4 journaling mode for consistency. Ordered mode care the metadata consistency in file system. However, journal mode assures data and metadata consistency. Selective Journaling use file system journal mode and turn off the SQLite journaling if only one page modified. Even though the file system checkpoint does not running, partial database update don't need to be considered because only one page need to be registered. If the page changes more than one, there is no guarantee that I/O computing is included in the database transaction. This lead to partial computing that could be happened while checkpoint is working. To prevent this, file system data journal is replaced to SQLite journal. EX4 data mode changes to ordered mode(Figure 2).

Selective Journaling does not have any performance advantage if the page modify more than one. However, Selective Journaling get less effect from this reason because the previous research [4] shows that SQLite in most Android OS I/O care only one pages.

4.1 Build in SQLite

In SQLite journal mode, at least two pages are modified because of database metadata changes. In

this case, metadata embedding [4] reduce I/O comes from the metadata changes. In WAL mode, which makes log, additional I/O comes from the wal frame header, the small data structure register for saving page information. Header embedding [6] helps to reduce I/O.

4.2 Build in File System

Current file system mount options can change journal mode. File system data mode can be switched by mount command line. The journal mode changes happen in every fsync()/fdatasync() function calls.

Chapter 5. Evaluation

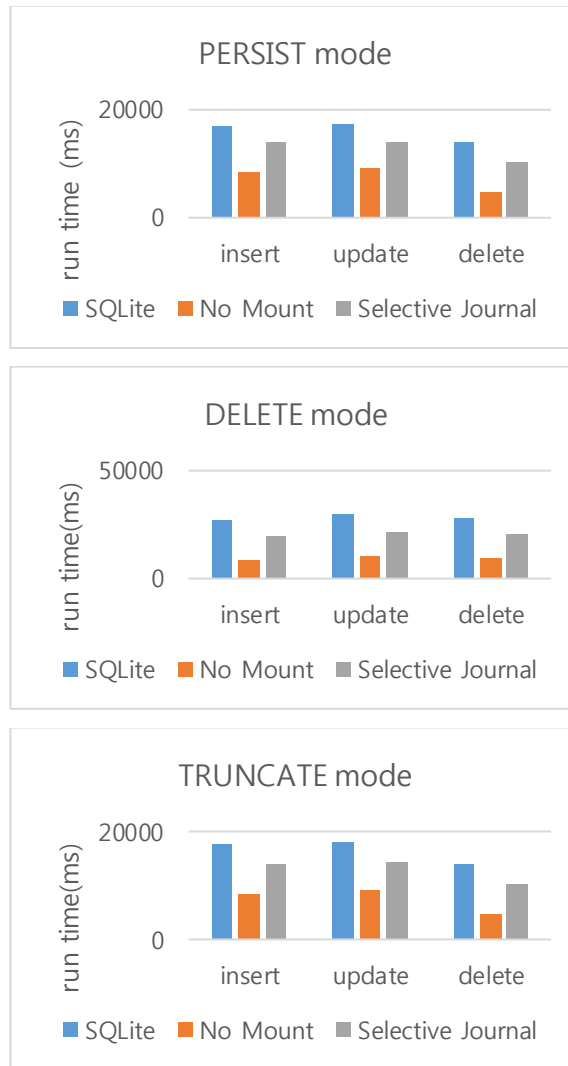


Figure 3 The performance result from 1000 transactions in each Insert, Update, and Delete query

This section evaluate the Selective Journaling performance with following environment set up. Selective Journaling is modified based on SQLite version 3.14. Selective Journaling run in Samsung SSD 840 PRO Series.

Selective Journaling, original SQLite and No mount are compared to check performance. To change the mount option in Selective Journaling, mount option change command is included. No Mount remove this command line to measure the command line overhead. The file system default mount option of No Mount is ordered mode. No Mount cannot be used as a SQLite because every one page modification does not save recovery journal; there is no file system journaling and database journaling.

Selective Journaling, SQLite Journaling and No Mount are tested with insert, update, and delete

queries. Truncate, persist, and delete modes are applied in this test. Insert, update, and delete queries run with integer key and string value. The size of key and value is 30 ~34 bytes per a transaction. Each query tested with 1000 transactions.

WAL mode is excluded in this comparison. Although WAL mode performance is better than journaling based recovery, most of mobile apps use journal mode because WAL mode does not guarantee the multiple transaction's atomicity [11].

Figure 3 shows that Selective Journaling performance is better than the original SQLite. Most of SQLite transactions modify one page. Thus, file system journaling for recovery is applied most of transactions in Selective journaling. From this result, the journaling of journal is minimized.

The run time difference between No Mount and Selective Journaling is the mount option change command line overhead which is the reason of Selective Journaling low performance.

Chapter 6. Conclusion and future work

This paper suggest Selective Journaling for solving Journaling of journal without any hardware and data structure supports. The experiments use mount command to switch file system data mode, and check the amount of performance enhancement. The results show that PERSIST, DELETE, and TRUNCATE mode in Selective Journaling performance are better than the SQLite original journal modes.

The ideal Selective Journaling performance is similar with No Mount one. Based on the No Mount performance, mount option change command is the overhead of current Selective Journaling. Next plan is set up syscalls can manipulate file system mount option in SQLite application. This will reduce the mount option change command overhead.

REFERENCES

1. H. Kim, N. Agrawal, and C. Ungureanu, *Revisiting storage for smartphones*, in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. 2012, USENIX Association: San Jose, CA. p. 17-17.
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*. 2012. ACM Tampere, Finland.
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*. 2013. USENIX Association San Jose, CA.
4. W.-H. 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*. 2014, USENIX Association: Santa Clara, CA. p. 273-285.
5. W.-H. Kim, J. Kim, W. Baek, B. Nam, and Y. Won, *NVWAL: Exploiting NVRAM in Write-Ahead Logging*, in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. 2016, ACM: Atlanta, Georgia, USA. p. 385-398.
6. W. Lee, K. Lee, H. Sun, W.-H. Kim, B. Nam, and Y. Won, *WALDIO: eliminating the filesystem journaling in resolving the journaling of journal anomaly*, in *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*. 2015, USENIX Association: Santa Clara, CA. p. 235-247.
7. H. Luo, L. Tian, and H. Jiang, *qNVRAM: quasi non-volatile RAM for low overhead persistency enforcement in smartphones*, in *Proceedings of the 6th USENIX conference on Hot Topics in Storage and File Systems*. 2014, USENIX Association: Philadelphia, PA. p. 4-4.
8. W.-H. Kang, S.-W. Lee, B. Moon, G.-H. Oh, and C. Min. *X-FTL: transactional FTL for SQLite databases*. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013. ACM New York, NY.
9. G. Oh, S. Kim, S.-W. Lee, and B. Moon, *SQLite optimization with phase change memory for mobile applications*. Proc. VLDB Endow., 2015. **8**(12): p. 1454-1465.
10. K. Shen, S. Park, and M. Zhu, *Journaling of journal is (almost) free*, in *Proceedings of the 12th USENIX conference on File and Storage Technologies*. 2014, USENIX Association: Santa Clara, CA. p. 287-293.
11. K. Lee and Y. Won. "SQLite WAL mode multiple transaction." The Institute of Electronics and Information Engineers, (2015.6): 1331-1334. Print.

12. SQLite web site. <https://sqlite.org/>
13. EXT3 web site. <https://en.wikipedia.org/wiki/Ext3>

Acknowledgement

I entered UNIST in 2015 as a graduate student. So far, I've seen the development of UNIST and have grown with it together. I wish to express deep affections and attachment to my school, UNIST for supporting me both materially and spiritually as well as providing a lot of opportunities.

I would like to show my gratitude Prof. Beomseok Nam for giving me a good guideline for this paper. He also provides advices and feedbacks. During the master course, I could learn how to study new materials and think good ideas.

Specially, I am grateful for all the professors of the department of Computer Engineering. They give me not only professional knowledge but also worthy advices to do the research. Also, I owe many thanks to my colleagues for their assistance for this thesis.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Again, thank you for all the people who have supported and inspired me.

