Design of Embedded Database Based on Hybrid Storage of PRAM and NAND Flash Memory

Youngwoo Park, Sung Kyu Park, and Kyu Ho Park

Korea Advanced Institute of Science and Technology (KAIST),
305-701, Guseong-dong, Yuseong-gu, Daejeon, Korea
{ywpark,skpark,kpark}@core.kaist.ac.kr

Abstract. Andorid which is the popular smart phone OS uses a database system to manage its private data storage. Although the database system supports a powerful and lighteweight database engine, its performance is limited by a single storage media, NAND flash memory, and a single file system, YAFFS2. In this paper, we propose a new embedded database system based on hybrid storage of PRAM and NAND flash memory. Using the byte-level and in-place read/write capability of PRAM, we separately manage a journaling process of the database system. It increases the transaction speed and reduces the additional overhead caused by NAND flash memory. We implement our database system using SQLite and dual file systems (YAFFS2 and PRAMFS). Consequently, the proposed database system reduces the response time of the database transaction by 45% compared to the conventional database system. In addition, it mitigates the burden of NAND flash memory management. Moreover, previous database applications can be executed on the proposed system without any modification.

Keywords: Database, NAND flash memory, PRAM, SQLite.

1 Introduction

Recently, various non-volatile memories such as NAND flash memory [16] and PRAM (Phase-change RAM) [17] have been developed very quickly. These non-volatile memories gain acceptance as an alternative storage media.

NAND flash memories have become increasingly popular for the main data storage of the embedded system due to its shock-resistance, low power consumption, and high I/O speed. The characteristics of NAND flash memory are very different from those of traditional magnetic disks. It supports a page-level (512B or 2KB) read and write operation, but the write speed is much slower than the read speed. Once a page is written, the page should be erased first in order to update data on that page. The erase operation performs in terms of block (32KB or 256KB) and takes much longer than page write time. All pages in a block should be unnecessarily erased at the same time. As a result, NAND flash memory cannot be managed by in-place update scheme like a disk. It is very important to optimize the write and erase count for NAND flash memory based system.

J. Xu et al. (Eds.): DASFAA Workshops 2011, LNCS 6637, pp. 254–263, 2011.

Table 1. Characteristics of NAND Flash Memory and PRAM

	PRAM [7]	NAND Flash Memory [16]
Volatility	Non-volatile	Non-volatile
Interface	Byte-addreable	Page based I/O
Capacity	512Mbit	8 Gbit
Read Time	80ns/Word	60μs/Page$^+$
Write Time	1μs/Word	800μs/Page$^+$
Erase Time	N/A	1.5ms/Block*
Endurance	1M write cycles	5K erase cycles

+Page = 2KB, *Block=128KB.

On the other hand, PRAM has begun to gain acceptance as an alternative to the embedded storage because of its high density and performance. PRAM has been developed by key industry manufacturers such as Intel, Samsung, and IBM. Already, Samsung commercialize 32MB PRAM for mobile devices to replace NOR flash memory to store boot-up codes. Unlike NAND flash memory, PRAM supports byte-level read/write and in-place update. These characteristics can complement page-level read/write and out-of-place updates of NAND flash memory in the database system.

Table 1 summarizes the characteristics of NAND flash memory and PRAM. These special features of new storage media impose new challenges to the traditional database system. In order to make the database system adapt to those memories, it is necessary to reconsider the storage architecture and develop new database management algorithms.

Currently, Android, which is the popular smart phone OS releasd by Google, uses database to manage its private data storage[18]. Fig. 1 shows the database system of Android. Android uses SQLite library which is widely used for the embedded system [12]. Because SQLite is a file based database system and NAND flash memory is the main storage of smart phone, YAFFS2 [8] is used for database file management. The database system of Android supports a powerful and lightweight relational database engine available to all applications. However, the storage related management is compeletly assigned to the single file system. It is hard to adopt a new database architecture and develop a specific algorithm of the database system. The performance of Android's database system is limited by YAFFS2 and NAND flash memory.

In this paper, we propose a hybrid storage architecture and transaction method for the embedded database system. In our hybrid storage architecture, NAND flash memory is used for the main database storage. At the same time, phase change random access memory (PRAM) accommodates the temporal database transaction. Because of byte-level read/write capability, PRAM ensures better

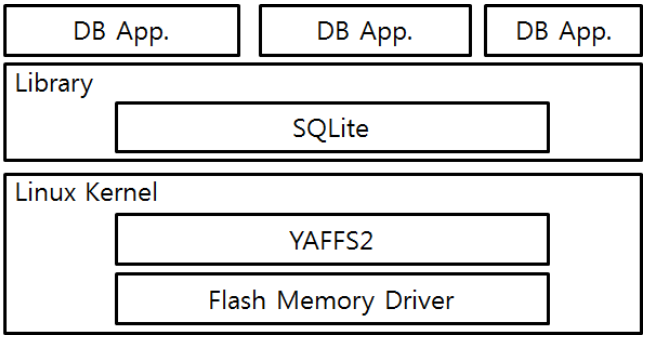


Fig. 1. Database system of Google's Android

performance for small size operations than NAND flash memory. Moreover, it can reduce many writes of NAND flash memory during transaction and decrease the database system overhead for storage management.

In our implementation, we modify SQLite database library to exploit PRAM as an alternative storage. SQLite does make use of temporary files during transactions. We separately store those temporary files to PRAM using a dual file system approach which is YAFFS2 for NAND flash memory and PRAMFS [9] for PRAM. The experimental results show that the proposed embedded database architecture reduces the response time of database transaction by 45% compared to thr traditional database system which consists of single NAND flash memory.

2 Related Work

Many researchers have developed new algorithms for flash-based DBMS. IPL [3] presents in-page logging approach in which the change made to a data page in memory are buffered on the per-page basis instead of writing the page entirely, thus avoiding high latency of write and erase operations. PDL [4] proposes a page-differential logging scheme with three design principles which are writing-difference-only, at-most-one-page writing, and at-most-two-page reading. These principles can guarantee good performance in both read and write operations and prolong the lifetime of flash memory by reducing the number of erase operations.

Some approaches have been tried to scale down DBMS. PicoDBMS [1] is designed for the resource-constrained smartcard. It proposes a new pointer-based storage model with a unique compact data structure and query execution with no RAM consumption, thus matching performance with the smartcard application's requirements. LGeDBMS [2] is designed for mobile systems. It applies the design principle of log-structured file system (LFS) [11]. It can reduce the number of I/Os by writing a log once rather than writing a log for each data change.

However, there is a limitation for designing DBMS only with flash memory. Although the average size of updates in DBMS is about dozens of bytes, the unit of the write operation in flash memory is over 2KB or 4KB. Flash memory also

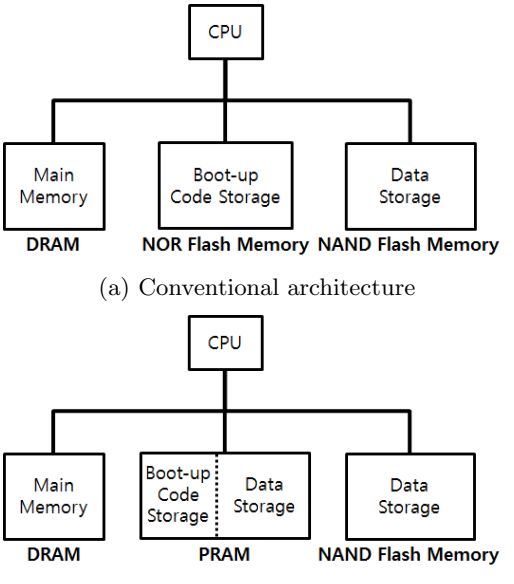
has long erase time, about 1.5ms, but small random requests in DBMS make a large number of erase operations.

Many researches have used non-volatile memories like NOR, FRAM, and PRAM [17] to handle small-sized data. Using the byte-level read/write capability of NOR flash memory, HFFS [10] synchronously stores data as a log in NOR flash memory, thus reducing the write count and providing a long lifespan of NAND flash memory. FRASH [5] proposes a novel file system for FRAM and flash memory, thus resolving long mount time issue. PFFS [6] presents a scalable and efficient flash file system using the combination of flash memory and PRAM. PFFS separates the metadata from the regular data in a file system and saves them into PRAM, thus solving the scalability problem and improving write and garbage collection performance.

We can solve the limitation only with NAND flash memory of the traditional database system by exploing hybrid storage of NAND flash memory and PRAM in this paper. It results in reducing the response time of database transactions and mitigating the burden of NAND flash memory management.

3 Hybrid Storage Architecture

For the embedded database system, we propose a hybrid storage architecture. Fig. 2 compares the proposed architecture with a conventional architecture. As shown in Fig. 2(a), the conventional embedded system uses NOR flash memory as boot-up code storage.



(a) Conventional architecture

(b) Proposed hybrid architecture

Fig. 2. Storage architecture comparison

On the other hand, the proposed hybrid architecture employs PRAM as boot-up code as shown in Fig. 2(b). PRAM supports the byte-level access and XIP (eXecution In Place). There is no problem to replace NOR flash memory to PRAM as boot-up code storage. In addition, the remaining space of PRAM is used as an alternative database storage while NAND flash memory is still main database storage. Because the size of boot-up codes is very small compared to commercialized PRAM size, PRAM can be utilized to enhance database performance. Using PRAM data storage, we can store a part of data generated during the database transaction and reduce unnecessary access to NAND flash memory.

4 Transaction on Hybrid Storage

In our hybrid storage architecture, the data storage of PRAM can be used to optimize several parts of the database system. In this paper, we target to enhance the transaction process. Fig. 3 shows the process of database transaction based on hybrid storage of PRAM and NAND flash memory.

The key of transaction process is to store journal for rollback operation. First, data to be updated are read to a database buffer from NAND flash memory. Then, these original data are updated in the database buffer. After updating data, journal data which are updated parts of data are written to PRAM instead of NAND flash memory. Originally, both data and journal data are written to NAND flash memory. It makes performance degradation because of two reasons. The first one is that the unit of write operation is page whose size is over 2KB,

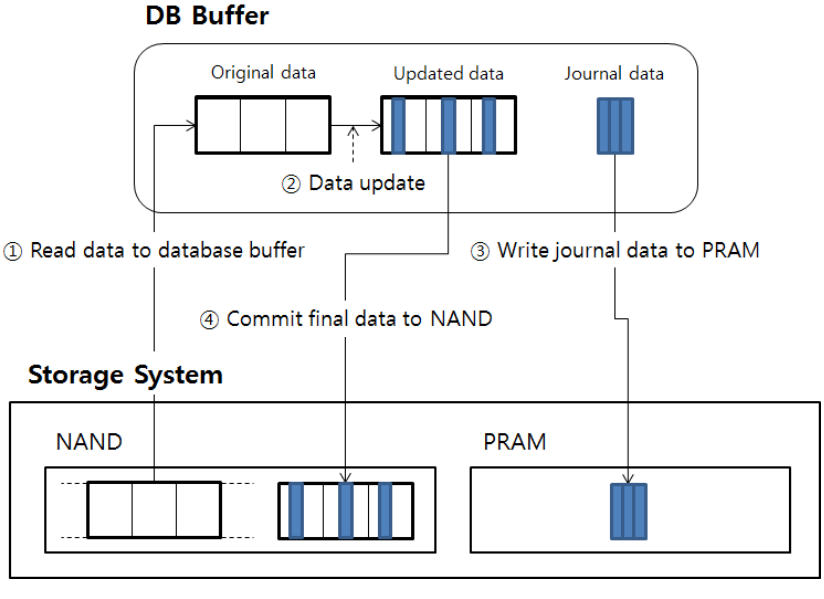


Fig. 3. Transaction Process on hybrid Storage

but the size of journal data is small. Another reason is that write operations in NAND flash memory make additional erase operations due to the characteristic, called out-of-place update, thus it can increase the operation time. Therefore, we can improve performance by writing journal data to PRAM. As whole updated data are written to NAND flash memory, the transaction process is finished.

5 Implementation Issue

We implement our database system using SQLite. SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, and transactional SQL database engine [12]. It is widely used for the embedded system because it is very compact library which can be less 300KB with all features enabled. Currently, Android uses SQLite as its built-in embedded database [13].

If we adopt SQLite library to our hybrid storage architecture, we need a method to access both PRAM and NAND flash memory. Basically, a database in SQLite is a single file. SQLite is implemented on the file system. One way to utilize hybrid storage is to develop a new file system and SQLite transaction algorithm for hybrid storage. However, it causes modification of all database applications and subordinates our database system to specific file system.

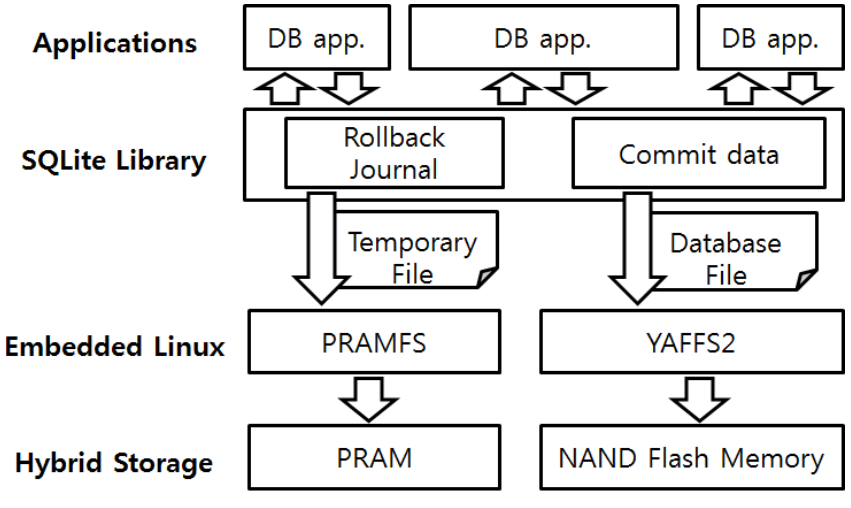


Fig. 4. Software architecture of proposed database system

Instead of file system modification, we use dual file systems and PRAM storage for storing temporal rollback information. As shown in Fig. 4, we use two separate file systems for single database management. YAFFS2, which is used for Googles Android, is NAND flash memory based file system [8]. PRAM is managed by PRAMFS [9] that is a simple file system designed for non-volatile memory. These two file systems are used to store files generated by SQLite database library.

YAFFS2 manages main database storage and PRAMFS stores only temporal file called rollback journal.

Currently, SQLite uses the rollback journal to atomic commit and rollback capabilities in SQLite. The rollback journal is frequently accessed and updated because it stores all information needed to restore the database back to its original state. In addition, the rollback journal is created when a transaction is first started and is deleted when a transaction commits or rolls back [12]. The size of rollback journal is small enough to store in small-sized PRAM storage. Because of non-volatility of PRAM, there is no problem to rollback transaction after sudden system power-off. Therefore, if the rollback journal is managed by PRAMFS, we can reduce a lot of data read/write of NAND flash memory during transaction and increase the database system performance.

Because the rollback journal is originally located in the same directory as the database file, we simply modify the SQLite library to designate the location of rollback journal. We never change a programming interface of SQLite. Previous database applications are compatibly executed on the proposed system. Moreover, there is no file system dependency in the proposed database system. We can use any two file systems to separate rollback journal from database. Even it identically works as conventional SQLite when using single file system.

6 Experiment

6.1 Experimental Environment

In order to evaluate our database system, we use an evaluation board shown in Fig. 5. This evaluation board has a 266MHz ARM processor and 64MB of SDRAM. It can include most of the storage devices that are currently used for embedded systems: NOR flash memory, SLC and MLC NAND flash memory,

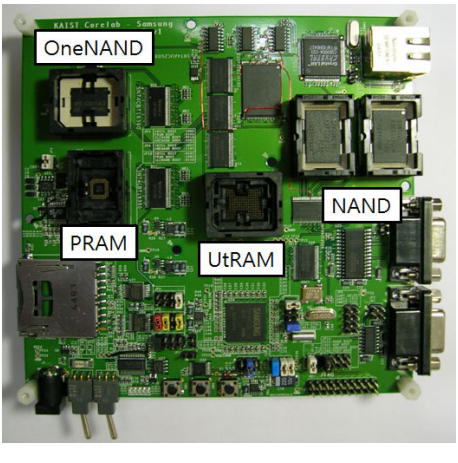


Fig. 5. The evaluation board for experiment

OneNAND, UtRAM, and PRAM. We can organize the proposed hybrid storage architecture on that board. However, PRAM is not currently available, thus the performance evaluation was actually carried out with an emulated PRAM by using UtRAM. We analyze the read/write time of PRAM and emulate that by power backed UtRAM with software delay [6][15]. Our emulation method is enough to evaluate our hybrid storage architecture because PRAM read/write time is deterministic and the data in power backed UtRAM was not volatilized.

6.2 Experimental Result

We implement our database system in a Linux OS with a kernel 2.6.19 version. Latest version of YAFFS2 and PRAMFS are used for file system of NAND flash memory and PRAM. We modify SQLite 3.7.3 vesion to select rollback journal location optionally. The proposed database system is compared to the conventional embedded database system that uses YAFFS2 file system with single NAND flash storage. To evaluate the performance of database, we use DBT-2 benchmark which is a fair usage implementation of the TPC-C benchmark specification [14]. Our results are measured by taking the average of consecutive five tests of DBT-2 benchmark.

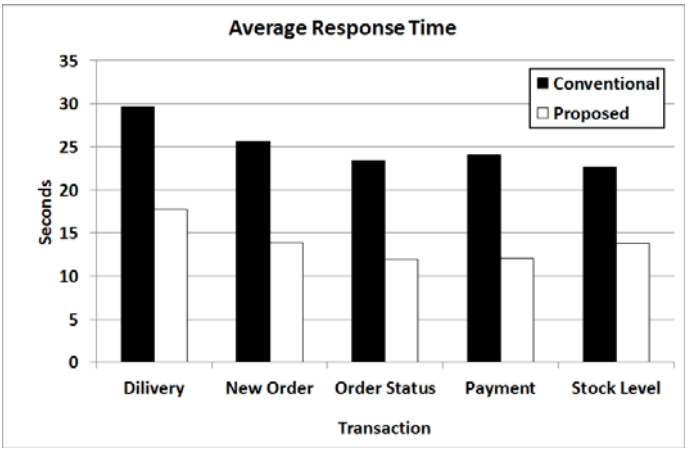


Fig. 6. Average response time of DBT-2 benchmark

Fig. 6 summarizes the average response time of TPC-C transaction. The results show that our proposed database system reduces the response time of all kinds of TPC-C transactions by 45%. The average NOTPM (New Order Transaction Per Minute) value is increased to 0.334 in the proposed database system, while the conventional database system shows only 0.172 NOTPM. This is mainly because the proposed system decreases the overhead of rollback journal. The proposed storage makes it possible to read/write rollback journal by single byte. We can reduce rollback journal update latency.

Table 2. Comparison of Write and Erase Count in NAND Flash Memory between Conventional and Proposed Database

	Conventional Database with Only NAND Flash Memory	Proposed Database with NAND Flash Memory and PRAM
Avg. Write Count	13020	5178
Avg. Erase Count	77	12

In addition, we can exploit in-place update in the proposed system. Table 2 shows the average write and erase count of NAND flash memory made during DBT-2 benchmark execution. Because proposed system updates rollback journal data in PRAM, it reduces many write and erase operations of NAND flash memory and increases the performance of our database system. Moreover, if the write and erase count is reduced, the overhead of NAND flash memory management like garbage collection and wear-leveling may be also decreased significantly.

7 Conclusion

Because of advance of the non-volatile memory technology, we should redesign database system. Each non-volatile memory has very different characteristics. It is important to consider the features of non-volatile memories. In our proposed design, we developed the hybrid storage architecture to use the advantage of PRAM and NAND flash memory. We prove that the embedded database system performance can be increased by simple modification. Our implementation is compatibly used for previous SQLite application. Our idea is further applicable to storage architecture using other non-volatile memories.

8 Future Work

As the further work, we'll separate other temporary files such as master journals, statement journals, TEMP databases, materializations of views and subqueries, transient indices, and transient databases used by VACUUM as well as the rollback journals for writing to PRAM. We can also exploit PRAM for storing frequently updated data for reducing write operations in NAND flash memory. Because PRAM has limited size compared to NAND flash memory, we need to develop allocation, swapping, and buffering schemes in SQLite layer or file system layer according to the data size or type.

References

1. Pucheral, P., Bouganim, L., Valdureiz, P., Bobineau, C.: PicoDBMS: Scaling Down Database Techniques for the Smartcard. In: Proc. the 26th International Conference on Very Large Data Bases (VLDB 2000), pp. 11–20 (2000)

2. Kim, G.-J., Baek, S.-C., Lee, H.-S., Lee, H.-D., Joe, M.J.: LGeDBMS: A Small DBMS for Embedded System with Flash Memory. In: Proc. the 32nd International Conference on Very Large Data Bases (VLDB 2006), pp. 1255–1258 (2006)
3. Lee, S.-W., Moon, B.: Design of Flash-Based DBMS: An In-Page Logging Approach. In: Proc. the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD 2007), pp. 55–66 (2007)
4. Kim, Y.-R., Whang, K.-Y., Song, I.-Y.: Page-Differential Logging: An Efficient and DBMS-independent Approach for Storing Data into Flash Memory. In: Proc. the 2010 International Conference on Management of Data (SIGMOD 2010), pp. 363–374 (2010)
5. Kim, E.-K., Shin, H., Jeon, B.-G., Han, S., Jung, J., Won, Y.: FRASH: Hierarchical File System for FRAM and Flash. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007, Part I. LNCS, vol. 4705, pp. 238–251. Springer, Heidelberg (2007)
6. Park, Y., Lim, S.-H., Lee, C., Park, K.H.: PFFS: A Scalable Flash Memory File System for the Hybrid Architecture of Phase-Change RAM. In: Proc. the 2008 ACM Symposium on Applied Computing (SAC 2008), pp. 1498–1503 (2008)
7. Qureshi, M.K., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., Abali, B.: Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap with Start-Gap Wear Leveling. In: Proc. the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42), pp. 14–23 (2010)
8. Aleph One Ltd., Yet Another Flash File System, YAFFS (2002), `http://www.yaffs.net`
9. Protected and Persistent RAM Filesystem, `http://pramfs.sourceforge.net`
10. Lee, C., Baek, S.H., Park, K.H.: A Hybrid Flash File System Based on NOR and NAND Flash Memories for Embedded Devices. IEEE Transactions on Computers 57(7), 1002–1008 (2008)
11. Rosenblum, M., Ousterhout, J.K.: The Design and Implementation of a Log-Structured File System. In: Proc. the 13th ACM Symposium on Operating Systems Principles (1992)
12. SQLite, `http://www.sqlite.org`
13. Wikipedia, SQLite, `http://en.wikipedia.org/wiki/SQLite`
14. Database Test 2 (DBT − 2^{TM}), `http://osdldbt.sourceforge.net`
15. K1S5616BCM Data Sheet, `http://www.samsung.com`
16. K9F2G08U0A Data Sheet, `http://www.samsung.com`
17. Wikipedia, Phase-change memory, `http://en.wikipedia.org/wiki/Phase-change_memory`
18. What is Android?, `http://developer.android.com/guide/basics/what-is-android.html`