P-SQLITE: PRAM-Based Mobile DBMS for Write Performance Enhancement

Woong Choi, Sung Kyu Park, Seong Min Kim, Min Kyu Maeng, †Ki-Woong Park, and Kyu Ho Park
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology (KAIST)
†Daejeon University
Daejeon, Korea
Email: {wchoi, skpark, smkim, mkmaeng}@core.kaist.ac.kr, †woongbak@dju.kr, kpark@ee.kaist.ac.kr

***Abstract*—Recent advances in mobile services have led to ever-increasing demands for high performance mobile database due to its data-centric characteristics. However, NAND flash memory, which is the main storage medium of mobile device has drawbacks in write operation because of its erase-before-write characteristics. In an attempt to overcome the drawbacks of the NAND flash memory, we propose a new mobile database management system (MDBMS), called P-SQLITE. It is enhanced with the find-grained data management techniques for using phage change random access memory (PRAM) as a caching layer between main memory and NAND flash memory. P-SQLITE continuously monitors the write patterns to identify hot chunk which is fine-grained file access unit and migrates the hot chunks from NAND flash memory to the PRAM. The experimental results show that the P-SQLITE improves the write performance by 47% than the previous MDBMS.**

Keywords-MDBMS; NAND flash memory; PRAM

I. INTRODUCTION

In current mobile devices, NAND flash memory has led the mobile device market as a main storage system due to its portability, large capacity, and performance of read operation. Even though it has outstanding features for mobile environment, it has two serious limitations. One of the problem is read and write asymmetry where the write operation is almost 10 times slower than the read operation. If the application running on the mobile device is write-intensive, the performance of NAND flash memory is degraded itself. Also, garbage collection overhead is another drawback of it. Because NAND flash memory cannot execute in-place update which updates previously saved data in the same physical location of its cell, it invalidates the original data in a previous location and stores updated data in a new location. When the occupation of invalidated data in NAND flash memory is increased in some degree, garbage collection is performed to reclaim the invalidated data. Since the garbage collection causes additional write and erase operations, performance degradation occurs.

In order to overcome these drawbacks, non-volatile random access memories (NVRAM) such as phase change RAM (PRAM) [1], ferroelectric RAM (FRAM) [2], and magnetoresistive RAM (MRAM) [3] are widely employed as a candidate for the storage system of the mobile device to replace NAND flash memory. Among these memories, PRAM is the closest to being on the market. One of the attractive points of PRAM compared to NAND flash memory is its speed of write operation. Additionally, it has in-place update characteristics where NAND flash memory doesn't offer. However, PRAM has a capacity problem due to the limitation of the current technology. To overcome this problem, hybrid approach, which is the combination of NAND flash memory and PRAM is studied currently in the storage area [4], [5]. From this, it is possible to achieve the benefits of NAND flash memory (large capacity, fast read speed) and PRAM (fast write speed, in-place update).

Concurrently, mobile database management system (MDBMS) has been developed as a software data management system of the mobile device. Especially, SQLite [6] is widely used as a database to manage the data of mobile applications in the Android [7] platform which is the famous mobile Operating System (OS) released by Google. The performance of SQLite is bounded by that of storage because it stores the whole database in the host file system as a single file [8]. It means that the execution time of mobile application is closely related to the degradation of NAND flash memory performance due to the write performance.

In this paper, we propose a new mobile database management system, called P-SQLITE to enhance the write performance of MDBMS by applying hybrid architecture composed of NAND flash memory and PRAM. P-SQLITE detects the fine-grained hotness (write count) of MDBMS chunks which are fine-grained file access units and transfers them to PRAM. Also, it contains chunk migration considering the limited capacity of PRAM so that more recent written chunks are placed in the PRAM.

The remaining part of this paper is organized as follows. Section II introduces the previous work to enhance MDBMS with various methods. In Section III, overall architecture of P-SQLITE which is our proposed MDBMS are explained. Detailed descriptions of its modules and functionalities are discussed in Section IV. Evaluation for them are summarized in Section V while the conclusion and future work are given in Section VI.

 ISBN: 978-1-61208-272-1

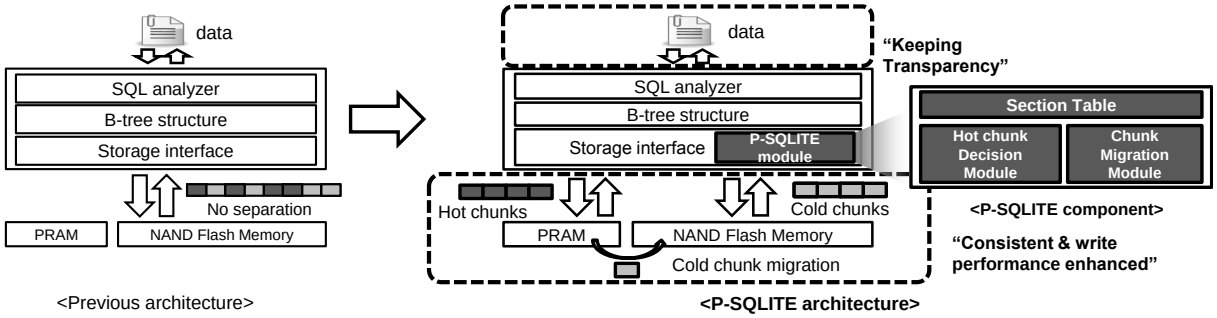


Figure 1: Overall Architecture of P-SQLITE compared with Previous MDBMS Architecture

II. Related Work

There are several researches to leverage the write performance of the file system equipped with NAND flash memory. As a recent study of software support, Yanfei Lv et al. [9] proposed operation-aware buffer management in the flash-based system to reduce the cost of read/write performance of NAND flash memory. They suggest Flash-based Operation-aware buffer Replacement (FOR) algorithm which considers both asymmetry of read/write speed and operation wise statistics to achieve high performance. However, it fails to support consistency of the dirty data in the buffer in case of unexpected power failure which brings the loss of valuable and necessary user data in the mobile devices.

PFFS2 [10] suggested PRAM as a metadata storage to reduce garbage collection overhead and write operations of NAND flash memory in the file system layer. Metadata is the data which indicates the fundamental information data of the file. In this research, they use virtual metadata storage which enables the migration when there is excess of metadata in PRAM. However, it is not appropriate to apply this method to improve the write performance of MDBMS since it handles a database as a single file.

Also, there were some efforts to leverage write performance on MDBMS. Y. park et al. [11] studied the effect of rollback journal files of MDBMS. Rollback journal files are back-up files to prevent unexpected loss of power during the modification. They decrease the write latency of the journal files and the number of garbage collections in NAND flash memory by storing them into the PRAM. Even if they increase the performance of MDBMS, it cannot handle the excess of the journal files due to various mobile databases since they assume the sufficient size of PRAM.

By analyzing the relationship between previous works and MDBMS, some important criteria should be guaranteed to improve its performance. First of all, gap between the hybrid storage architecture and current MDBMS have to be solved. Because current MDBMS is mismatched with the hybrid storage architecture in respect of data storing, this bottleneck should be solved to increase the write performance. Finally, the limited capacity of PRAM should be considered. Without the practical restriction of PRAM size, it is hard to assure improvement of write performance where PRAM is full.

III. P-SQLITE Overview

In this section, the overall design of P-SQLITE and its process sequence are described. It contains P-SQLITE module to leverage the write performance of the storage. Even when it needs additional operations compared to previous MDBMS to achieve fine-grained management of data, we improve the write performance of MDBMS through P-SQLITE.

Figure 1 shows the overall architecture of P-SQLITE compared with previous MDBMS architecture. In the case of previous MDBMS, data is firstly inserted as structured query language (SQL). After that, the SQL analyzer parses and figures out the SQL. Then, the data with the SQL is stored in the form of a node or nodes of B-tree and stored in a file through the storage interface. However, previous MDBMS doesn't use the database chunk as a unit of data management where it is a node of B-tree and fundamental structure of MDBMS. Thus, this architecture considers neither the write frequency of each chunk of the file nor hybrid storage system to enhance the write performance.

On the other hand, P-SQLITE enables fine-grained hot-cold separation and migration of chunks on the hybrid architecture composed of PRAM and NAND flash memory. To do this, it contains new modules in the storage interface. The P-SQLITE module is composed of three main parts which are *section table, hot chunk decision module, and chunk migration module*. *Section table* records the write count of each chunk periodically. Based on this information, *hot chunk decision module* decides whether chunk is allocated in PRAM or not. The module puts marks for selected chunks on section table, not placing them at that moment. When actual writing is requested for the marked chunks again,the chunk actually is stored in NVRAM at that moment. This lazy policy reduces unnecessary migration of a chunk from NAND flash memory to NVRAM. Lastly, *chunk migration module* migrates relatively colder chunks from PRAM to NAND flash memory to cover the limited capacity of PRAM.

 ISBN: 978-1-61208-272-1

TABLE I. STRUCTURE OF SECTION TABLE

Chunk#	State	Device	WriteCount	EWMA
0	USED	PRAM	5	4.93
1	CHANGE	NAND	2	3.18
2	USED	NAND	4	2.09
3	USED	NAND	1	0.67
4	UNUSED	-	-	-
...	...	...	...	...

IV. MAIN MODULES OF P-SQLITE

In this section, main modules of P-SQLITE are presented. P-SQLITE consists of three components which are a section table, hot chunk decision module, and chunk migration module. We provide the detailed design and implementations of each component in following sub-sections.

A. Section Table

Section table is designed to handle the data with fine-grained chunk unit rather than file unit which are relatively coarse-grained. To manage the data with chunk unit, write count and location of each chunk should be recorded. Table I shows the structure of the section table. It has four indexes which are state, device location, write count in current transaction period and exponentially weighted moving average (EWMA) [12] for each chunk. The state column represents whether this chunk is used (USED), not used yet (UNUSED), or ready to change location into other devices (CHANGE). In this case, we added CHANGE state to apply the lazy policy which is explained in previous section. Secondly, the device column indicates whether each chunk is allocated in PRAM or NAND flash memory. The write count and EWMA columns are used to decide how corresponding chunks are frequently and recently written. Detail explanation for above two indexes are introduced in the next section. While implementing section table, it occupies two bytes in memory space per chunk which has size of two kilobytes. Since the space for section table is only a thousandth portion of original data, its overhead is negligible.

B. Hot Chunk Decision Module

The hot chunk decision process consists of new write count expectation based on write count history and total performance gain calculation based on write count expectation. First of all, defining the transaction period is needed to implement this decision module.

1) Transaction Period: Decision of period is important since it affects the reflection degree of recent tendency. To reflect recent tendency of write count, P-SQLITE updates write count of each chunk in every period. Since the database executes write operation in a transaction unit, we set the period as several execution of write transactions. We call this number of transactions as *transaction period*. If the transaction period is too short, high prediction miss is expected since number of sample data is too small. Furthermore, a short transaction period incurs frequent write count prediction process which causes performance overhead. On the other hand, performance is degraded by unnecessary migration because most of chunks are considered as recent data if the transaction period is too long.

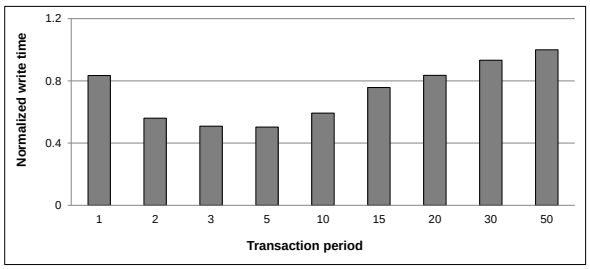


Figure 2: Normalized Write Time per Transaction Period

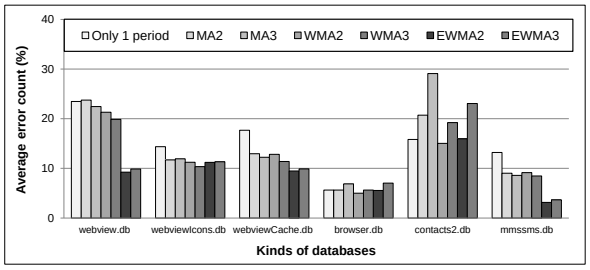


Figure 3: Avg. Error Rate during Various Transaction Periods

To verify the effect of transaction period and determine an appropriate value, we conduct experiment with six databases which are associated with web browser, message and phone book application with 1000 write transactions. Also, simple prediction policy which considers chunk as hot if write count of that chunk is over half of transaction count in a period are adopted to calculate migration penalty. As shown in Figure 2, there is relationship between transaction period and write performance as we mentioned. From the experimental result, we set the five transactions as a transaction period which gives the shortest write time.

2) Write Count Expectation: P-SQLITE predicts the write count in the future based on the previous write count of each chunk. Thus, prediction failure results in designating cold chunks as hot chunks and vice versa. In order to reduce an error rate in history based write count expectation, accurate expectation method is needed. Therefore, we evaluated various history-based prediction methods such as moving average (MA), weighted moving average (WMA), and exponentially weighted moving average (EWMA) with several periods.

To find out the best method which give least error, we did experiment with the three different methods and various periods. For this experiment, same applications which are

 ISBN: 978-1-61208-272-1

mentioned in the previous section are adopted. As shown in Figure 3, EWMA with two periods gives least error value which is 9% on average and 15% at most for the various applications. Therefore, we selected this EWMA method to find out the expectation value of a next transaction period.

$$n^*_{t+1} = \alpha n_t + (1-\alpha) n^*_t$$
$$(\alpha = 2/N+1)$$

,where (1)

n^*_{t+1}: EWMA of next period,
n_t: write count of current period,
n^*_t: EWMA of current period,
N: transaction period size.

The EWMA of next period is calculated by (1). P-SQLITE calculates the new EWMA for next period using current write count and EWMA values. The newly calculated EWMA value at the end of a transaction period is stored in the EWMA column for next calculation again.

3) Gain Calculation for Hot Chunk Decision: To select the set of the hot chunks which will bring the highest performance gain based on the calculated EWMA, P-SQLITE calculates the effective performance gain of each chunk. This calculation mainly consists of two parts: reordering the Section Table by new EWMA value of next period and calculating the gain by each row. Firstly, Reordering section table arranges the table in descending order to facilitate hot chunks selection. Then, P-SQLITE consider both performance gain by allocating the chunks in PRAM and performance loss by chunk migration from PRAM to NAND flash memory due to the PRAM capacity limitation.

As the first step of hot chunk decision, recorded section table is sorted by calculated EWMA value in descending order except for unused chunks as shown in Figure 4. In the figure, τ represents the number of hot chunks which are selected from the top of the table. P-SQLITE regard τ as the meaningful threshold of hot chunks.

$$TotalGain(\tau) = Gain_{PRAM}(\tau) - Loss_{Migration}(\tau)$$

, where

$Gain_{PRAM}(\tau)$: performance gain by alloting chunks in PRAM,
$Loss_{Migration}(\tau)$: performance loss by chunk migration from PRAM to NAND. (2)

After that, total performance gain is calculated as equation of τ. As shown in (2), the equation consists of two terms which are performance gain by allotting the numbers of chunks from the hottest order in PRAM and performance loss by chunks migrated from PRAM to NAND flash memory.

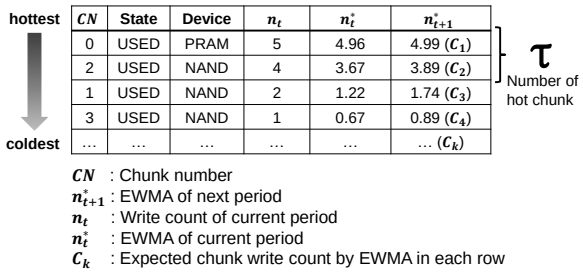


Figure 4: Reordered Section Table according to New EWMA

Finally, the detailed formula is provided in (3). First of all, the performance gain part is calculated by multiplication of the benefit by using PRAM per write count of a chunk and expected write count of the chunks set by τ. The benefit by using PRAM per a chunk write count is defined as the difference between write speed of PRAM and NAND flash memory. Expected write count is acquired by EWMA value in the table. As the number of chunk write increases, the performance gain also increases. Secondly, the performance loss is obtained by multiplication of the migration overhead per chunk and the number of chunk which will be migrated to NAND flash memory. If PRAM is full and new chunk is allocated in PRAM, relatively colder chunks are migrated to NAND flash memory in P-SQLITE. At this time, migration overhead is defined as the summation of read speed of PRAM and write speed of NAND flash memory. The more migrated chunks P-SQLITE has, the more migration loss is generated.

$$TotalGain(\tau) = (NAND_{WS} - PRAM_{WS}) \sum_{i=1}^{\tau} C_i - (PRAM_{RS} + NAND_{WS}) \times Chunk_{MN}(\tau)$$

, where

$NAND_{WS}$: Write speed of NAND,
$PRAM_{WS}$: Write speed of PRAM,
C_i: Expected chunk write count by new EWMA in each row,
$PRAM_{RS}$: Read speed of PRAM,
$Chunk_{MN}$: Chunks which will migrate to NAND by τ. (3)

C. Chunk Migration Module

If PRAM is full and new chunks are about to migrate into PRAM, migration of relatively colder chunk need to occur to accept a new hotter chunk as shown in Figure 5. In this case, P-SQLITE selects least recently used (LRU) as victim selection algorithm. To apply the strategy, finding the recently used chunk is essential. P-SQLITE checks the section table which has write count and EWMA in the

 ISBN: 978-1-61208-272-1

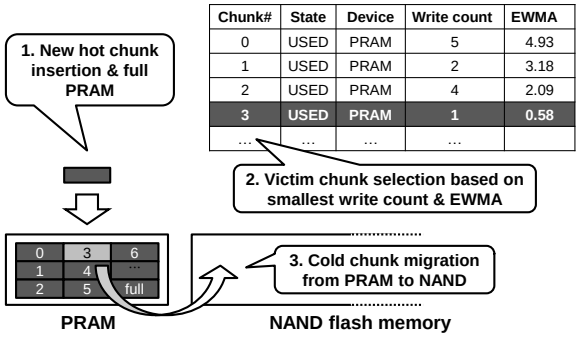


Figure 5: Sequence of Chunk Migration

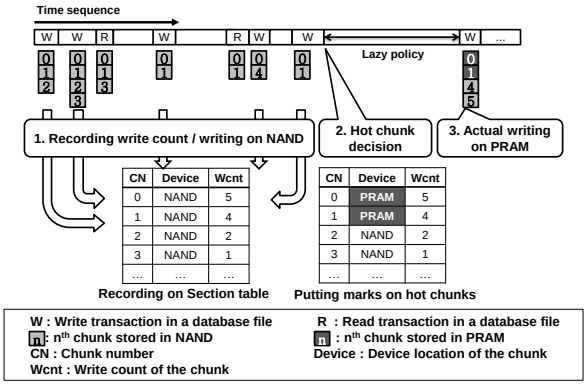


Figure 6: Process Sequence of P-SQLITE

current period to find out LRU chunk in PRAM. Based on these two values, it figures out the victim chunk in PRAM. Small write count and EWMA value of the chunk means that it is not recently used. P-SQLITE firstly selects the chunk which has smaller write count and secondly compares EWMA if there are chunks which have same write counts.

D. Overall Process Sequence of P-SQLITE

To find out hot chunks which are frequently updated in a database file, P-SQLITE firstly record the write count of each chunk per write transaction. Figure 6 shows the recording and assigning process of P-SQLITE. For every write transaction, write count of each chunk is written in the section table. After that, hot chunk decision module evaluates chunks which will be assigned in PRAM based on the section table. In this moment, it inserts mark for selected chunk in the section table, not placing them into PRAM immediately. Those chunks are stored in PRAM only after the actual writing command is requested for them again. This lazy policy reduces unnecessary migration of chunks from NAND flash memory to PRAM to prevent extravagant usage of PRAM capacity.

At the moment of actual writing, it is hard to expect the write performance enhancement by PRAM when it is full and recent hotter chunks are stored in NAND flash memory. To solve this problem, chunk migration module moves relatively colder chunks from PRAM to NAND flash memory.

V. EVALUATION

In this section, experimental environment and results are explained. Firstly, the specification of hardware components, software layers, and workload characteristics are introduced. After that, the experimental results for the effect of P-SQLITE compared to previous work are given.

A. Experimental Setup

We developed P-SQLITE in HBE-EMPOS3 SV210 board [13] made by Hanback electronics. It has 800Mhz ARM Conrtex-A8 CPU, 512Mbyte of DDR2 SDRAM and 256Mbyte SLC NAND flash memory. In case of software base, Android 2.2 proyo [7] and Linux Kernel 2.6.32 were used as the main operating systems in the P-SQLITE. Our P-SQLITE is implemented based on the fundamental MDBMS, SQLITE [6] which is basically installed in the Android platform to manage mobile applications. We evaluated P-SQLITE with databases of four practical applications which are web browser (webview.db, webviewcache.db, webviewIcons.db), text message (mmssms.db), phone book (contacts2.db), and alarm (alarms.db).

We selected PRAM as an NVRAM of P-SQLITE where it is widely discussed in the storage research area and it is the closest RAM to being on sale. Because PRAM and its specifications are not released yet, we used the specification of [14]. Also, we emulated the PRAM with SDRAM by inserting additional delays in the file system level since the target board only contains SDRAM. Additionally, we filled up PRAM with sufficiently large database files in each application to consider the capacity limitation problem of PRAM. Table II shows the parameters for NAND flash memory and PRAM.

B. Write Performance Improvement of P-SQLITE

Firstly, we measure the read and write performance of P-SQLITE to analyze overall performance. As shown in the Table II, read latency of PRAM is similar to that with NAND flash memory and PRAM has much higher write latency than read latency. Thus, write performance is critical factor of P-SQLITE performance. In Figure 7, the write

TABLE II. PARAMETERS FOR NAND FLASH MEMORY AND PRAM

	NAND Flash	PRAM
Device capacity	256MB	8MB
Page Size	2KB	-
Block Size	128KB	-
Read time	100μs/page	20ns/byte(40μs/2KB)
Write time	800μs/page	100ns/byte(200μs/2KB)
Erase time	1.5ms/block	-

 ISBN: 978-1-61208-272-1

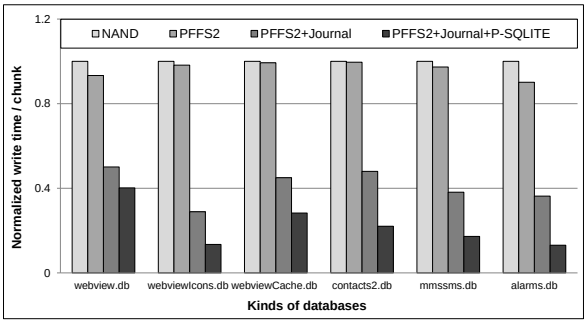
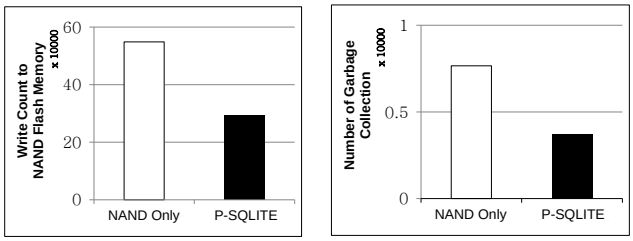


Figure 7: Normalized Write Time of Various Systems



(a) Write Count (b) Garbage Collection

Figure 8: Write Count and Garbage Collection in NAND Flash Memory

performance of each database with four different approaches including P-SQLITE are evaluated. We combined P-SQLITE with previous works and estimated the write performance of them which were referred in Section II. "PFFS2" [10] stores metadata in PRAM at the file system level and "Journal" [11] places the rollback journal in PRAM at database software level. By adopting PFFS2, it gives only 7% write enhancement compared to NAND flash memory. Because MDBMS has only one file to handle the database, the effect of saving metadata in PRAM is comparatively small. Case of rollback journal file, it gives performance enhancement by almost 50% on average compared to NAND flash memory since it reduces the garbage collection. Finally, the average write performance improvement by P-SQLITE is about 47% compared to previous scheme where both previous works are applied. Since P-SQLITE is able to combine with previous works independently, the scheme adopting whole three techniques has 77% write performance improvement compared to only NAND flash memory.

C. Reduction of Write Count and Garbage Collection

We also estimated the total write count and number of garbage collection in NAND flash memory during 5000 write transactions of phone book application. Figure 8 shows that the P-SQLITE brings 46% write count in NAND flash memory and 51% reduction of garbage collection compared to NAND flash memory. It means our work concentrates the frequent write count in PRAM efficiently where it increases the write performance of MDBMS on the hybrid storage system.

D. Discussion

In our P-SQLITE system, additional cost is needed where the space overhead occurs due to the Section Table. However, additional two bytes per 2K chunk for Section Table are negligible compared to total performance.. Also, it causes additional overhead for migration. If write count pattern is flat, performance overhead can become smaller.

VI. Conclusion

In this paper, we propose P-SQLITE which enhances the write performance of the storage system with fine-grained data management. It contains section table which records the information of each chunk of the file. We also design hot data decision module to select the chunks which will be allocated to PRAM dynamically. Finally, P-SQLITE has chunk migration module which migrates colder chunk from PRAM to NAND flash memory to consider the limited size of PRAM. Through this implementation, we improve the write performance of the previous MDBMS in the hybrid storage architecture by 47% on average.

For the further work, we will focus on the write endurance problem of PRAM. Limited lifetime of PRAM is also challenging issue and proper wear-leveling technique for the system should be considered. We are planning to clarify the appropriate wear-leveling policy for mobile database management system.

Acknowledgment

The work presented in this paper was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. 10035231-2010-01.

References

[1] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam, "Phase-change random access memory: A scalable technology", IBM Journal of Research and Development, vol. 52. no. 4, July. 2008, pp.465-479.

[2] A. Sheikholeslami and P. G. Gulak, "A survey of circuit innovations in ferroelectric random-access memories", Proceedings of the IEEE, vol. 88, no. 5, May. 2000, pp. 667-689.

[3] S. Tehrani, J. Slaughter, E. Chen, M. Durlam, J. Shi, and M. DeHerren, "Progress and outlook for mram technology", IEEE Transactions on Magnetics, vol. 35, no. 5, Sep. 1999, pp. 2814-2819.

[4] S. Lee, S. Jung, and Y. Song, "An efficient use of PRAM for an enhancement in the performance and durability of NAND storage systems", IEEE Transactions on Consumer Electronics, vol. 58, no. 3, Feb. 2000, pp. 825-833.

[5] H. Lee, "High-performance NAND and PRAM hybrid storage design for consumer electronics", IEEE Transactions on Consumer Electronics, vol. 56, no. 1, Feb. 2000, pp. 112-118.

 ISBN: 978-1-61208-272-1

[6] Sqlite official webpage, http://www.sqlite.org/, retrieved: March, 2013.

[7] Android, Software stack for mobile devices that includes an operating system, http://www.android.com/, retrieved: March, 2013.

[8] J.-M. Kim and J.-S. Kim, "AndroBench: Benchmarking the storage performance of android-based mobile devices", Frontiers in Computer Education AISC, 2012, pp. 667-674.

[9] Y. Lv, B. Cui, B. He, and X. Chen, "Operation-aware bufferr management in flash-based systems", Proc. The 2011 International Conference on Management of Data (SIGMOD'11), 2011, pp. 13-24.

[10] Y. Park and K. H. Park, "High-performance scalable flash file system using virtual metadata storage with phase-change ram", IEEE Transactions on Computers, vol. 60, no. 3, Mar. 2011, pp. 321-334.

[11] Y. Park, S. K. Park, and K. H. Park, "Design of embedded database based on hybrid storage of pram and nand flash memory", Proc. The 16th International Conference on Database Systems for Advanced Applications (DASFAA'11), 2011, pp. 254-264.

[12] N. Ye, Q. Chen, and C. Borror, "Ewma forecast of normal system activity for computer intrusion" detection. IEEE Transactions on Reliability, vol. 53, no. 4, Dec. 2004, pp. 557-566.

[13] Hbe-empos3 sv210 board, http://www.hanbak.cn/en/, retrieved: March, 2013.

[14] M. Kryder and C. S. Kim, "After hard drives: what comes next?", IEEE Transactions on Magnetics, vol. 45, no. 10, Oct. 2009, pp. 3406-3413.

 ISBN: 978-1-61208-272-1