Flash Memory Shadow Paging Scheme for Portable Computers: Design and Performance Evaluation

Siwoo Byun, Seongyun Cho and Moonhaeng Huh

Deptartment of Digital Media, Anyang University
708-113, Anyang 5-dong, Manan-gu, Anyang-shi, Kyonggi-do 430-714, Republic of Korea
Tel: +82-31-467-0922, Fax: +82-31-467-0800
E-mail: {swbyun,scho,moonh}@anyang.ac.kr

Recently, a flash memory has become a major database storage in building portable information devices because of its non-volatile, shock-resistant, power-economic nature, and fast access time for read operations. We propose a new scheme called flash memory shadow paging *(FMSP) for efficient page management in a flash memory database environment. We improved traditional shadow paging schemes by reusing old data pages which are supposed to be disposed in the course of writing a new data page in the flash memory file systems. In order to reuse these data pages, we devised a deferred cleaning procedure and an operation interface which is geared to existing flash file systems.* FMSP *contributes to overcome the two drawbacks of traditional shadow paging schemes, additional space overhead and slow access caused by old page management. We also propose a simulation model to show the performance of FMSP. Based on the results of the performance evaluation, we conclude that* FMSP *scheme outperforms the traditional shadow paging schemes.*

ACM Classification: H.2.7 Database administration

1. INTRODUCTION

Advances in portable computing and internet service technologies (Byun and Hong, 2000; Pons, 2003) have resulted in extensive use of portable information devices such as PDAs (Personal Digital Assistants), HPCs (Hand-held PCs), and PPCs (Pocket PCs) in e-commerce environments. Each information device may run several applications, such as a small database, a personal information management system (Greer and Murtaza, 2003), and e-commerce software (Gyeung and Lee, 2003). Flash memory is one of the best candidates to support small information devices for data management in portable computing environments (Byun, 2006).

Recently, flash memory has become a critical component in building embedded systems or portable devices because it is non-volatile, shock-resistant, and uses little power. Its performance has improved to a level at which it can be used not only as the main storage for portable computers but also as mass storage for general computing systems (Chang and Kuo, 2005). Although flash memory is not as fast as RAM, it is hundreds of times faster than a hard disk in read operations. A performance comparison is shown in Table 1. These attractive features make flash memory one of the best choices for portable information systems (Yim and Koh, 2003; Yim, 2005).

Manuscript received: 17 Octoiber 2005
Communicating Editor: Gill Dobbie

Storage Media	Volatility	Read Time	Write Time	Erase Time
DRAM	Volatile	100 ns (1B)	100 ns (1B)	–
NOR Flash	Non-volatile	150 ns (1B)	200 ms (1B)	1 s (128KB)
NAND Flash	Non-volatile	36 ms (512B)	266 ms (512B)	2 ms (16KB)
Hard Disk	Non-volatile	12.4 ms (512B)	12.4 ms (512B)	–

Table 1: Performance Comparison of Storage Media

However, flash memory has two critical drawbacks. First, a segment, blocks of flash memory, need to be erased before they can be rewritten. This is because flash memory technology only allows individual bits to be toggled in one way for writes. The erase operation writes ones or zeros into all the bits in a segment. This erase operation takes much longer than a read or write operation. The second drawback is that the life of each memory block is limited to 1,000,000 writes. A flash management system should wear down all memory blocks as evenly as possible (Kim and Lee, 1999; Wu, Chang and Kuo, 2003).

Due to these disadvantages, traditional database technologies are not easy to apply directly to flash memory databases on portable devices. A database management system which is based on flash memory media must exploit the advantages of flash memory features while overcoming its constraints.

2. RELATED WORKS

In general database systems, transaction mechanisms are employed to maintain consistency and integrity in the presence of concurrent activities and failures. When a transaction is aborted, any modification to the database by the transaction has to be undone. And undoing the transaction typically involves relatively slow I/O's and large spaces for recovery (Kun-Lung and Kent, 1993).

Two commonly known recovery techniques are *update-in-place* approach (Vijay and Albert, 1992) and *shadow paging* approach (Jack and Hector, 1988). The update-in-place approach allows data modifications made by a transaction to be output to the database while the transaction is still in an active state. The shadow paging approach keeps changes in a separate area until a successful completion of the transaction is assured, at which time the modifications are applied to the database.

In update-in-place approaches, both undo and redo logs should be saved in a log file before the transaction commits in order to make failure recovery possible. Checkpointing is also needed to maintain an up-to-date copy of the database and thereby provides a starting point for log recovery. The recovery procedure only needs to process the undo and redo log records generated after the last complete checkpoint. The overall performance would be poor in those cases where frequent I/O activities occur by logging and checkpointing. On the other hand, update-in-place approach has a merit to require much less space than shadow paging approach (Choi, Yoon, Song, Kim and Jin, 2000).

In shadow paging approaches, a database maintains two images per page during the lifetime of a transaction: a current page and a shadow page. When a transaction starts, both pages are identical. The shadow page is never changed over the duration of the transaction. The current page may be changed when a transaction performs a write operation (Song, Kim and Ryu, 1999). To undo modification, it frees current page. To commit modification, it modifies all pointers to old (shadow) page to now point to new (current) page, and frees the shadow page. If the system fails, then shadow pages are used to recover a stable status of the system before the failure.

Although shadow paging has an advantage of fast and simple recovery, there are two drawbacks. First, it additionally requires a large space to maintain shadow pages. Second, the movement of a page to new versions destroys the original layout of the stable database. That is, when a page is updated, there may not be space for the new copy close to the original page's location. For example, if related data of a file are originally stored contiguously for efficient sequential access, they would eventually be spread into other locations thereby slowing down sequential access. This problem is common to many implementations of shadow paging (Bernstein, Hadzilacos and Goodman, 1987).

A *flash memory database management system* (FM-DBMS) is essentially an instance of a memory-based database system. Although FM-DBMS is a new research field, *main memory database management systems* (MM-DBMS), based on RAM memory, are a popular research topic, and many system models, such as MARS (Gruenwld and Eich, 1991), System M (Garcia-Molina and Salem, 1992) and Tachyon (Kim and Choi, 2002) have been proposed.

The MARS system includes a database processor and recovery processor, each of which can access a volatile main memory containing the database. A nonvolatile memory such as a flash memory is also available to both processors. The database processor is responsible for the execution of transactions up to the commit point. Update transactions do not modify the primary database copy until the updating transaction commits. Instead, the database processor records the update in the nonvolatile flash memory. If the updating transaction is aborted, the recorded update is simply discarded. To commit a transaction, the recovery processor copies its update records into the database from the nonvolatile flash memory. The records are also copied into a nonvolatile log buffer.

System M is implemented as a collection of cooperating servers on the Mach operating system. Message servers accept transaction requests and return transaction results to clients. Transaction servers execute requested transactions, modifying the database and generating transaction log data. Log servers move in-memory transaction log data to disk, and checkpoint servers keep the disk-resident backup database up to date. System M is capable of processing transactions concurrently.

Compared to MM-DBMS, there has been little research on FM-DBMS. And most of the work that has been reported focuses on the enhancement of the physical storage and file systems (Chang and Kuo, 2002; Kim and Lee, 1999), and not on database management such as recovery. But if we could exploit the advantages of flash memory and overcome its disadvantages effectively, flash memory could be expected to contribute to modern database systems, especially in portable computing systems. And so, we propose a new data recovery scheme which is based on shadow paging for flash memory database systems.

3. SHADOW PAGING SCHEME FOR FLASH MEMORY DATABASE

3.1 Flash Memory Database Model

Our FM-DBMS model, as shown in Figure 1, comprises three distinct components: a flash memory database manager (FMDM), a flash memory page manager (FMPM), and a flash memory segment manager (FMSM). FMDM manages user transactions from start to commitment by coordinating the transaction manager, query processor, concurrency manager, and data manager. FMPM manages the flash memory page mapping table and the access controller which handles the flash memory database. FMSM is made up of four distinct modules: an allocator, a cleaner, a cycle leveler, and a collector (Kim and Lee, 1999). The allocator is responsible for keeping a pool of free segments and decides which of these is to be assigned next. The cleaner reclaims expired segments to free space. The cycle leveler is responsible for even distribution of writes and erases over the flash segments. Finally, the collector identifies cold data on the segments so as to reduce the overhead of the cleaner.

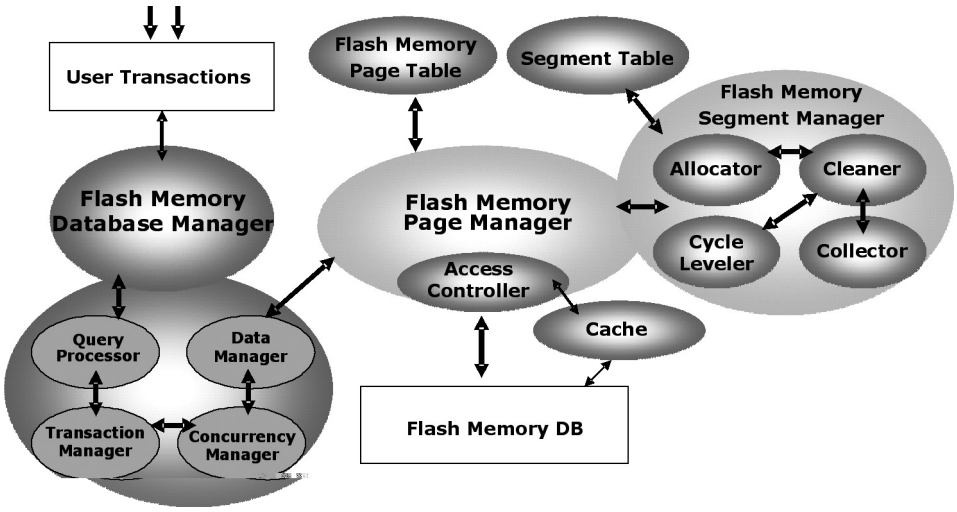

Figure 1: Flash Memory DBMS Model

3.2 File System Architecture for Flash Memory Storage

The flash memory storage differs from the existing hard disk in terms of limited number of writes for a flash memory unit. Thus, flash file systems should wear down all memory blocks as evenly as possible. In order to achieve wear leveling across flash memory pages, flash file systems are generally based on the Log-Structured File System (*LFS*) (Jeanna, Drew, Adam, Randolph and Wang, 1997).

In the LFS model, a file system is represented as an append-only log structure and a large number of data blocks are gathered in a cache before sequential writing in order to maximize throughput of collocated write operations. For example, Figure 2 shows a modified block written by LFS when creating two single-block files named dir1/file1 and dir2/flle2. LFS must write new data blocks and inodes for file1 and flle2, plus new data blocks and inodes for the containing directories. LFS performs the operations in a single large write.

The general flash file systems exploit the feature of sequential writes (append-only) in order to perform efficient wear leveling and to accelerate the performance of slow writes (Hui, Michael and

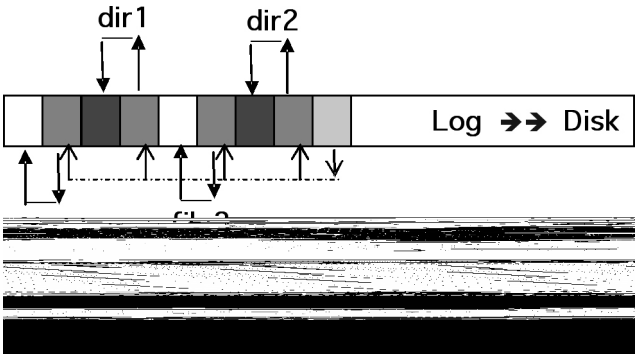

Figure 2: Log-Structured File System Model

Richard, 2004). In practice, most popular flash file systems such as JFFS (Journaling Flash File System) (jffs, 2005) and JFFS2 (jffs2, 2005) are essentially based on the LFS model. JFFS and JFFS2 are designed for use on flash-based PDA systems, e.g. the Hewlett Packard iPAQ. As in LFS, each write in JFFS2 creates a new data page in the log and then disuses the old data page by setting an invalid flag.

In the next section, we propose a new page management scheme which efficiently reuses these invalidated data pages in flash file system environments.

3.3 Flash Memory Shadow Paging Scheme Using Deferred Cleaning

Although traditional shadow paging schemes have an advantage of fast and simple transaction recovery, they essentially require large space to maintain shadow pages and destroy the original layout of data pages. In order to lessen this significant space overhead of shadow paging and to increase transaction performance, we propose *Flash Memory Shadow Paging* (FMSP) scheme which is suitable for portable devices which use flash memory as a major data storage.

While *FMSP* basically maintains a fast and simple recovery nature of traditional shadow paging schemes, *FMSP* exploits the characteristics of the flash file system for space efficiency. That is, *FMSP* reuses invalidated (expired) pages of the old version which are supposed to be disposed when a transaction writes a new version in the flash file systems such as JFFS and JFFS2. To reuse these invalidated pages efficiently, we need to devise a new shadow paging model (Figure 3) and an operation interface which is geared to generic flash file systems. FATM exploits these invalidated pages for use as shadow pages by a deferred cleaning mechanism, as follows.

- Upon the receipt of a write operation message issued by a write transaction in FM-DBMS, *FMSP* accesses the *Page Table* which maintains a pointer to the related data object in the Flash File System.
- Instead of overwriting the old (invalidated) page which contains the data object, *FMSP* creates a new page to write the current version of the data object. The new page is sequentially allocated by the *Allocator*.
- The *Deferred Cleaning List* maintains the invalidated page which contains the last committed version of the data object. *FMSP* delays the actual cleaning process of the invalidated page until the write transaction reaches to commitment state. The invalidated page is reused as a shadow page for fast recovery in case of a transaction abort.

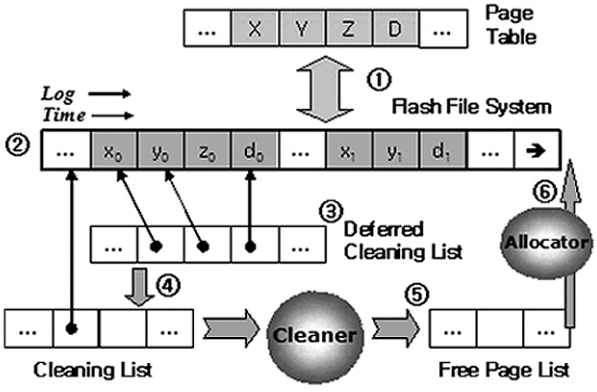

Figure 3: A New Shadow Paging Model for Flash File Systems

- If the write transaction has reached to commitment state, *FMSP* moves the shadow page into the *Cleaning List* for actual cleaning.
- The *Cleaner* periodically cleans the expired shadow page and then moves the cleaned page into the *Free Page List*.
- Upon the receipt of a page allocation request message from the Flash File System, the *Allocator* returns the pointer of a cleaned page in the *Free Page List*.

Since *FMSP* does not suffer from logging and check-pointing overhead in update-in-place approaches, *FMSP* could achieve high transaction throughput and fast response time. Furthermore, *FMSP* removes the drawback of requiring large backup space in traditional shadow paging approaches by reusing invalidated pages in flash file systems. And *FMSP* uses even less space than update-in-place schemes because *FMSP* does not maintain transaction logs.

In traditional shadow paging schemes, the page movement caused by writing a new version destroys the original page layout thereby slowing down sequential data access. *FMSP* also experiences this sort of page distribution. However, *FMSP* does not suffer from this problem of slowness which is common to traditional shadow paging schemes. This is because *FMSP* is based on flash memory in which the data location is not important to the data access speed like RAM.

Therefore, *FMSP* could contribute to overcome the two drawbacks of traditional shadow paging schemes, additional space overhead and slow access caused by page distribution. On the other hand, *FMSP* requires a new structure for a deferred cleaning list (Table 2) and an operation interface which is geared to the flash file system. But the additional overhead associated with this enhancement is outweighed by the positive effect of *FMSP*.

3.4 Operation Interface for FMSP

We now describe an operation interface for our flash memory shadow paging system. In order to handle flash operations, *Page Manager for Flash Memory* (PM/F) exchanges physical I/O operation/result messages with *Data Manager* (DM/F). *DM/F* is responsible for logical data operations. *PM/F* is responsible for accessing data/shadow pages of the requested I/O operations and returning I/O result messages back to *DM/F*. *Transaction Manager* (TM/F) is responsible for scheduling various operations associated with a user transaction issued at a portable device, and sends data access requests to *DM/F*.

Four types of operation messages, *read request*, *write request*, *commit request* and *abort request*, denoted by *OP_WRITE*, *OP_READ*, *OP_COMMIT* and *OP_ABORT*, respectively, are transferred from *TM/F* to *PM/F* via *DM/F*.

```
typedef struct Deferred_Cleaning_List {
    struct Deferred_Cleanning_List *prev_ptr; // double linked list , 4 bytes
    struct Deferred_Cleanning_List *next_ptr; // double linked list , 4 bytes
    struct Shadow_Page *shadow_page_ptr; // pointer to shadow page in Flash File System, 4 bytes
    unsigned long  data_object_id; // data object id , 4 bytes
    unsigned long  time_stamp; // save creation time for cleaning, 4 bytes
} deferred_cleaning_list; // total 20 bytes
```

Table 2: A Simple Structure of Deferred Cleaning List

- ***OP_WRITE*(*Tr, x, val*) Message:**
  This message is used to send a *write* operation request from transaction *Tr* to *PM/F*.
  1. Write *val* as a new value of data object *x* into an unused location in flash memory storage. The shadow page of old value of data object *x* is still maintained in the *deferred cleaning list*.
  2. Record this location's address as a new reference of data object *x*.
  3. Acknowledge to *TM/F* the processing of *OP_WRITE*.

- ***OP_READ*(*Tr, xl*) Message:**
  This message is used to send a *read* operation request from transaction *Tr* to *PM/F*.
  1. If *Tr* has previously written into data object *x*, return to *TM/F* the value stored in the current reference address.
  2. Otherwise, return to *TM/F* the value stored in the shadow page in the *deferred cleaning list*.

- ***OP_COMMIT*(*Tr*) Message:**
  This message is used to send a *commit* operation request from transaction *Tr* to *PM/F*.
  1. For each data object *x* updated by *Tr*, change the stable address of *x* from the shadow version page to the current version page.
  2. Remove the shadow pages of *x* from the *deferred cleaning list*, and invalidate them, and insert them into the *cleaning list*. The *Cleaner* cleans these invalidated pages to produce free pages.
  3. Acknowledge to *TM/F* the processing of *OP_COMMIT*.

- ***OP_ABORT*(*Tr*) Message:**
  This message is used to send an *abort* operation request from transaction *Tr* to *PM/F*.
  1. Invalidate the current pages used by data object *x*, and insert them into the *cleaning list*. The *Cleaner* cleans these invalidated pages to produce free pages.
  2. Remove the shadow pages of *x* from the *deferred cleaning list*, and set the stable address of *x* to the shadow page.
  3. Acknowledge to *TM/F* the processing of *OP_ABORT*.

4. COMPUTER SIMULATION

4.1 Simulation Model

We compared the performance of *FMSP* to the general shadow paging scheme, *general shadow paging* (GSP) scheme, by means of computer simulation. We used a *closed queuing model* for a flash memory database system. This model is closed in the sense that the system maintains its workload count-out by multiprogramming so long as the average number of active transactions stays the same.

The system model used for the simulation consists of five distinct components: the user transaction generator (UTG), the flash transaction manager (TM/F), the flash memory data manager (DM/F), the flash page manager (PM/F), and the flash segment manager (SM/F). The *UTG* is responsible for generating user transactions, modeled as a sequence of read and write operations. *TM/F* manages a user transaction from start to commitment, analyzes the transactions, and sends it to a data request queue of *DM/F*. *DM/F* is responsible for the logical data access, and sends physical I/O operations to *PM/F*. *PM/F* accepts the I/O requests one by one and processes each one using flash memory mapping with shadow pages and deferred cleaning list. The *SM/F* manages the allocator, cleaner, cycle leveler, and collector.

The number of issued transactions, *num_TRs*, effectively controls the workload level of concurrency in the system. Varying the value of *num_TRs* provides wide variation in the amount of

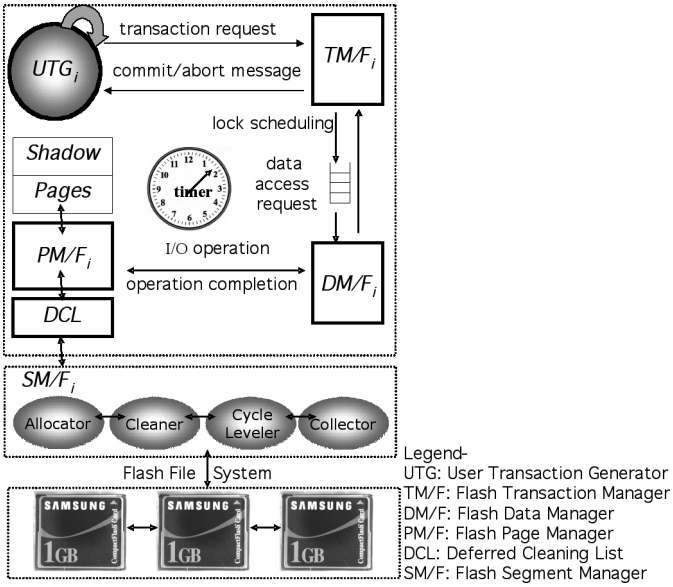

Figure 4: Queuing System Model for Simulation

data contention. The closed queuing model shown in Figure 4, is central to our simulation. This model was programmed using CSIM (Schwetman, 1992) discrete-event simulation software, and our experiments were carried out on a Pentium IV PC server running under the Microsoft Windows 2000 Server operating system.

4.2 Simulation Parameters

The simulation parameters are shown in Table 3. Some of the actual values used for simulation parameters were taken from previously reported work (Yim and Koh, 2003) where they are fully justified. Other values were chosen to investigate performance differences between *FMSP* and *GSP*, while keeping simulation times reasonable.

System Parameters	Description	Value
num_TRs	Number of transactions per second	500 ~ 3500 in steps of 500
cpu_delay	CPU time for accessing an data object	5 msecs
Flash_read_delay	I/O time for reading an object in flash memory	36 msecs
Flash_write_delay	I/O time for writing an object in flash memory	266 msecs
Flash_erase_delay	I/O time for erasing an object in flash memory	2 msecs
Flash_page_size	Page size in flash memory	512 Bytes
DC_object_size	Object size in a deferred cleaning list	20 Bytes
update_ratio	Probability of an update operation	50 percent

Table 3: Major Simulation Parameters

The primary performance metrics used in this study are the *transaction throughput* and *average response time*. Transactions throughput is defined to be the number of transactions that successfully completed per second. Average response time is defined to be the difference of points in time when a transaction is submitted and when it is successfully completed. An additional performance-related metric, *shadow bandwidth*, is used in analyzing the results of our experiments. Shadow bandwidth is defined to be the average transfer rate per second (Mbytes/sec) to save and manage shadow pages for a write transaction. A lower shadow bandwidth corresponds to a lower overhead for shadow page writes and management.

4.3 Results and Interpretation

We now analyze the results of simulations performed using the two shadow page management schemes, *FMSP* and *GSP*. We investigated the workload effect of concurrency level on the performance of transaction management schemes, by varying the number of transactions per second, *num_TRs*.

The overall transaction throughput as a function of *num_TRs* is presented in Figure 5, the corresponding average response times are depicted in Figure 6. In Figure 5, the x axis means the input workload which is the number of submitted transactions. Thus, varying the number of issued transactions per second means the variation of the system workload. The y axis means the output performance which is the number of completed transactions.

As *num_TRs* increases, the transaction throughput gradually grows with increasing concurrency. In this experiment, the highest throughput was achieved by *FMSP*, followed by *GSP*. This means that our *FMSP* is able to serve more transactions than *GSP* under the same workload condition. In Figure 6 the average response time gradually grows with increasing concurrency, which is largely due to the accompanying growth in data contention.

Figure 5 shows that the transaction throughput of each scheme levels off or starts to decline for values of *num_TRs* that are greater than 2500. Although *num_TRs* has been increased beyond 2500, the number of active transactions currently executing in the system appears to decrease slightly. This implies that adding more transactions simply increases data contention, without yielding additional concurrency and throughput. From this, it seems that the transaction performance is mainly limited by data contention.

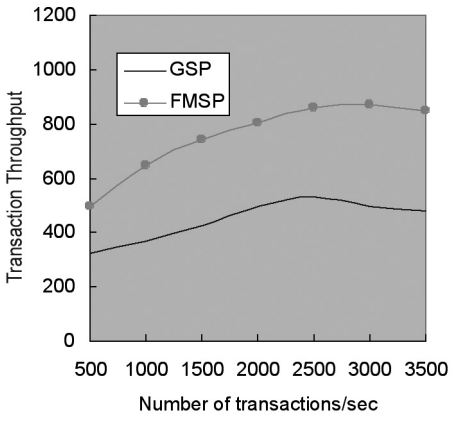

Figure 5: Transaction Throughput

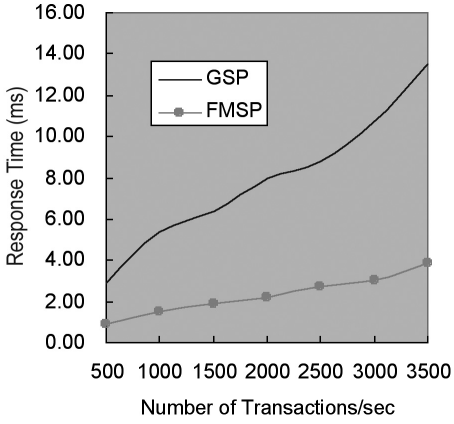

Figure 6: Average Response Time

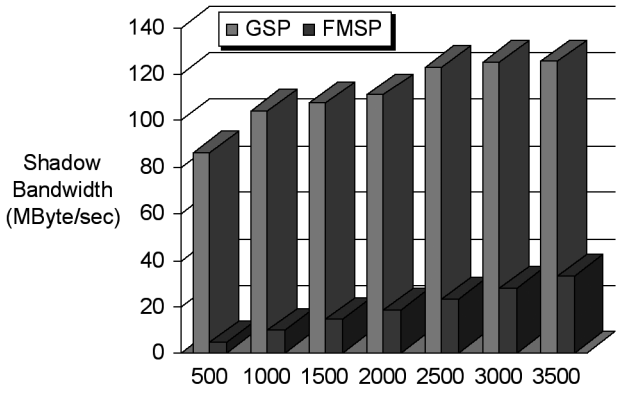

Figure 7: Shadow Bandwidth

From Figure 5, we can see that *FMSP* achieves higher transaction performance than *GSP* at all concurrency levels. Figure 6 shows that the response time of *FMSP* is superior to that of *GSP*. This is because *FMSP* reduces the space and time overhead for writing shadow pages by reusing invalidated pages efficiently in slow flash file systems. And this could lead to achieving high transaction throughput and fast response time.

This assertion is confirmed by shadow bandwidth in Figure 7, which shows that write rate for shadow pages is always lower using *FMSP* than using *GSP*. *FMSP* shows 82 percent lower shadow bandwidth than *GSP*. This is due to the positive effect of page reutilization by the deferred cleaning procedure in *FMSP*.

5. CONCLUSIONS

Currently, flash memory is the most popular storage media for information management in portable computing systems. We proposed *Flash Memory Shadow Paging* (FMSP) which is a new page management scheme for the flash memory database storage. *FMSP* removes additional storage overhead for keeping shadow pages by reusing invalidated data pages. We also devised a deferred cleaning procedure and its operation interface in flash memory file systems. Unlike previous shadow paging schemes, *FMSP* could contribute to overcome additional space overhead and slow access speed. We also proposed a simulation model based on a closed queuing system to show the performance of *FMSP*. Our simulation results show that FMSP outperforms the traditional shadow paging scheme in terms of response time and transaction throughput, especially in a high data contention environment.

For future research, in order to improve I/O performance of flash memory database systems, we shall study a new transaction control scheme which allows previous version reads and efficiently handles slow write and erase operations in lock management processes. In addition, a new index structure that reduces the number of write operations and a new index control scheme for handling the characteristics of flash memory efficiently need to be studied further since they could significantly affect the performance of flash memory storage systems.

REFERENCES

BERNSTEIN, P., HADZILACOS, V. and GOODMAN, N. (1987): *Concurrency control and recovery in database systems*, Addison-Wesley.

BYUN, S. and HONG, E. (2000): Increasing data availability for unreliable mobile computers. *Journal of Research and Practice in Information Technology* 32(3):181–199.
BYUN, S., HUR, M. and HWANG, H. (2006): Flash memory lock management for portable information systems, *International Journal of Cooperative Information Systems* 15(3): 461–479.
CHANG, L. and KUO, T. (2002): An adaptive striping architecture for flash memory storage systems of embedded systems, *Proc. 8th IEEE Real-Time and Embedded Technology Symposium*, California, San Jose, 187–196.
CHOI, M., YOON, H., SONG, E., KIM, Y-Keol, KIM, Y-Kuk, JIN, S., HAN, M. and CHOI, W. (2000): Two-step backup mechanism for real-time main memory database recovery. *Proceedings of the Seventh Intl Conference on Real-Time Systems and Applications* (*RTCSA'00*). Korea, Cheju Island, 453–457.
GARCIA-MOLINA, H. and SALEM, K. (1992): Main memory database systems: An overview, *IEEE Trans. Knowl. Data Eng.*, 4(6):509–516.
GREER, H. and MURTAZA, B. (2003): Web personalization: The impact of perceived innovation characteristics on the intention to use personalization. *Journal of Computer Information Systems*. 43(3):50–55.
GRUENWLD, L. and EICH, M. H. (May 1991): MMDB reload algorithms, *Proc. ACM SIGMOD Conference*, Denver, 397–405.
GYEUNG, K. and LEE, G. (2003): E-Catalog evaluation criteria and their relative importance. *Journal of Computer Information Systems*. 43(4):55–62.
HUI, D., MICHAEL, N. and RICHARD, H. (2004): ELF: An efficient log-structured flash file system for micro sensor nodes. *Proceedings of the 2nd international conference on Embedded networked sensor systems*. 176–187.
JACK, K. and HECTOR, G. (1988): Optimizing shadow recovery algorithms. *IEEE Transactions on Software Engineering*. 14(2):155–168.
JEANNA, N. M., DREW, R., ADAM, M. C., RANDOLPH, Y. and WANG, T. E. (1997): Improving the performance of log-structured file systems with adaptive methods. *Proceedings of the sixteenth ACM symposium on Operating systems principles*, Saint Malo, France, 238–251.
JFFS. (2005): Retrieved from http://developer.axis.com/software/jffs/
JFFS2. (2005): Retrieved from http://source,redhat.com/jffs2/
KIM, H. and LEE, S. (1999): A new flash memory management for flash storage system, *Proc. 23rd Annual International Computer Software and Applications Conference*, Arizona, Phoenix, 284–289.
KIM, S., CHOI, W. and KIM, B. H. (2002): Design and implementation of the concurrency control manager in the main-memory DBMS tachyon. *Proceedings of the 26th Annual International Computer Software and Applications Conference*. Oxford, England, 635–641.
KUN-LUNG, W. and KENT, F. (1993): Rapid transaction-undo recovery using twin-page storage management. *IEEE Transactions on Software Engineering*. 19(2):155–164.
MATTHEW, S. H. and JOHN, D. G. (1983): Shadowed management of free disk pages with a linked list. *ACM Transactions on Database Systems*. 8(4):503–514.
MENDEL, R. and JOHN, K. O. (1992): The design and implementation of a log-structured file system, *ACM Transactions on Computer Systems*. 10(1):26–52.
PONS, A. P. (2003): Enhancing the quality-of-service for application service providers. *Journal of Computer Information Systems*. 44(1):3–8.
SCHWETMAN, H. (1992): CSIM user's guide for use with CSIM revision 16, *Microelectronics and Comput. Technology Corporation*.
SONG, E. M., KIM, Y. K. and RYU, C. H. (1999): No-log recovery mechanism using stable memory for real-time main memory database systems. *Proceedings of the RTCSA'99*, 428–431.
VIJAY, K. and ALBERT, B. (1992): Performance measurement of main memory database recovery algorithms based on update-in-place and shadow approaches. *IEEE Transactions on Knowledge and Data Engineering*. 4(6):567–571.
WU, C., CHANG, L., and KUO, T. (2003): An efficient B-Tree layer for flash-memory storage systems, *Proc. RTCSA*, Taiwan, 409–430.
YIM, K. and KOH, K. (2003): A study on flash memory based storage systems depending on design techniques, *Proc. Information Science Conference*, 30(2–1) 274–276.
YIM, K. (2005): A novel memory hierarchy for flash memory based storage systems, *Journal of Semiconductor Technology and Science*, 5(4) 262–269.

BIOGRAPHICAL NOTES

Siwoo Byun received his B.S. degree in computer science from Yonsei University in 1989 and earned his M.S. and Ph.D. in computer science from the Korea Advanced Institute of Science and Technology (KAIST) in 1991 and 1999. Currently, he is teaching and researching in the area of distributed database systems, embedded systems, mobile computing, and fault-tolerant systems at Anyang University.

Siwoo Byun

Seongyun Cho

Seongyun Cho received his B.S. and M.S. degrees in electronic engineering from Hanyang University in 1987 and 1989 at Seoul, Korea. The Ph.D. degree was received in computer systems engineering from Wales University, Cardiff, U.K. in 1999. During 1989–1994, he stayed in the Korea Telecom Research Centre (KTRC), to study the Digital Exchange System Program, B-ISDN administration and management. He is now with Anyang University Digital Media Department as associate professor.

Moonhaeng Huh

Moonhaeng Huh received the B.S. degree from Soongsil University and the M.S. degree from Yonsei University in 1979 and 1989 respectively. He received the Ph.D. degree in computer science from Chungbuk National University, Korea, in 2003. He served as a chief researcher of computer communication and software engineering at Korea Telecom (KT) Research Centre from 1984 to 2000, and as the head of digital contents business in Korea IT Industry Promotion Agency (KIPA) from 2001 to 2004. In 2004, he joined the department of Digital Media, at Anyang University.