

# Reused Page Management for Log-Structured Flash Storage Systems<sup>1</sup>

Changbae Roh, Siwoo Byun  
 Dept. of Digital Media Anyang University  
 708-113, Anyang 5-dong, Manan-gu, Anyang-shi,  
 Kyonggi-do 430-714, Republic of Korea  
 {cbroh, swbyun}@anyang.ac.kr

## Abstract

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) scheme 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 mechanism 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 page distribution.

## 1. Introduction

Advances in portable computing and internet service technologies (Byun, 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 & Murtaza, 2003; JFFS, 2005), and e-commerce software (Gyeong & Lee, 2003; Richard, 1997). Flash memory is one of the best candidates to support small information devices for data management in portable computing environments. (Byun, 2004)

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 & Kuo, 2002). 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 (Yim & Koh, 2003). These attractive features make flash memory one of the best choices for portable information systems.

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 a 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 & Lee, 1999).

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.

Table 1. Performance Comparison of Storage Media

Media	Read Operation	Write Operation	Erase Operation
RAM	2.6 $\mu$ s (512B)	2.6 $\mu$ s (512B)	-
NOR Flash Memory	15 $\mu$ s (512B)	3.5 ms (512B)	1.2 s (128KB)
NAND Flash Memory	36 $\mu$ s (512B)	266 $\mu$ s (512B)	2 ms (16KB)
Hard Disk	12.4 ms (512B)	12.4 ms (512B)	-

<sup>1</sup> This work was supported from Korea Sanhak Foundation (2006).

## 2. Motivation

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 & Kent, 1993).

Two commonly known recovery techniques are *update-in-place* approach (Vijay & Albert, 1992) and *shadow paging* approach (Matthew & John, 1983; Jack & 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 modification 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 et al, 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, & 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 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 large space to maintain shadow pages. Second, the movement of 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 implementation of shadow paging.(Bernstein, Hadzilacos, & 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 & Eich, 1991), System M(Garcia-Molina & Salem, 1992) and Tachyon(Kim, Choi, & Kim, 2002) have been proposed. 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 & Kuo, 2002; Kim & 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 data manager (FMDM), a flash memory media manager(FMMM), 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 recovery manager. FMMM manages the flash memory 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 & 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.

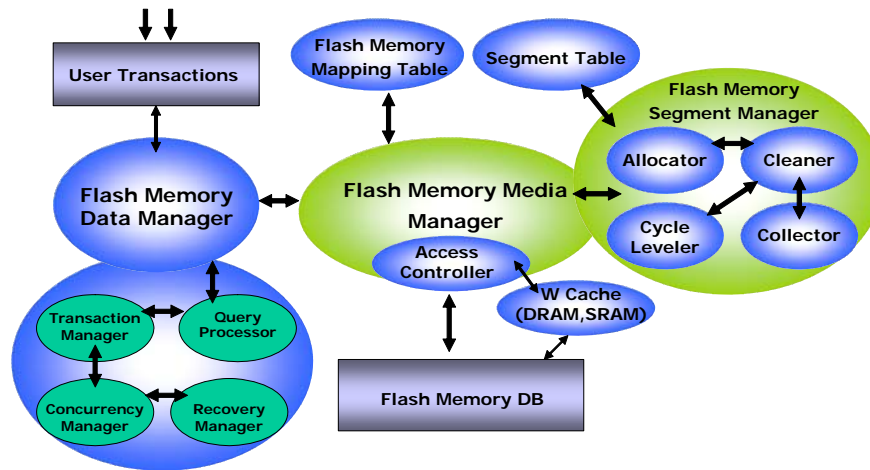


Figure 1. Flash Memory DBMS Model

### 3.2 File System Architecture for Flash Memory

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 Log-Structured File System (LFS) (Jeanna et al., 1997; Mendel & John, 1992).

In 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 modified blocks written by LFS when LFS creating two single-block files named dir1/file1 and dir2/file2. LFS must write new data blocks and inodes for file1 and file2, plus new data blocks and inodes for the containing directories. LFS performs the operations in a single large write.

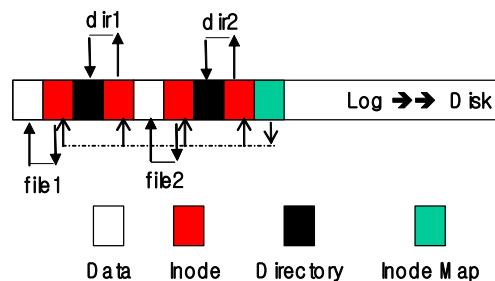


Figure 2. Log-Structured File System Model

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, & 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 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 space-efficient page management scheme which 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 *FMSP* (Flash

Memory Shadow Paging) 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 flash file system for space efficiency. That is, FMSP reuses invalidated (expired) pages of 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 deferred cleaning mechanism, as follows.

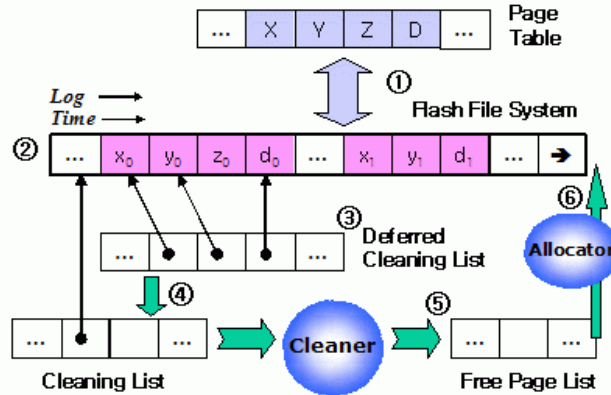


Figure 3. A New Shadow Paging Model for Flash File Systems

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*.

② Instead of overwriting the old 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.

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 as 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 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.

### Operation Interface for FMSP

We now describe an operation interface for our flash memory shadow paging system. In order to handle flash operations, *Shadow Paging Manager (SPM/F)* exchanges operation/result messages with *Transaction Manager(TM/F)*. *TM/F* is responsible for scheduling various operations associated with a user transaction issued at a portable device. *SPM/F* is responsible for accessing data object of the requested operations and returning result messages back to *TM/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 *SPM/F*.

- ***OP\_WRITE*(*Tr*, *x*, *val*) Message:**

This message is used to send a *write* operation request from transaction *Tr* to *SPM/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*, *x*) Message:**

This message is used to send a *read* operation request from transaction *Tr* to *SPM/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 *SPM/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 a *abort* operation request from transaction *Tr* to *SPM/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. 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 storage. *FMSP* removes additional storage overhead for keeping shadow pages by reusing invalidated data pages. We also devised a deferred cleaning mechanism and an operation interface for flash file systems. Unlike previous paging schemes, *FMSP* could contribute to overcome additional space overhead and slow access speed.

Since *FMSP* has a generic functionality of efficient storage management for flash memory file systems, it can be widely employed in portable and embedded computing devices. 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 effect the performance of flash memory database systems.

## References

- [1] Bernstein, P., Hadzilacos V., & Goodman N. (1987). Concurrency control and recovery in database systems: Addison-Wesley.
- [2] Byun S., & Hong E. (2000). Increasing Data Availability for Unreliable Mobile Computers. *Journal of Research and Practice in Information Technology*. 32(3). 181-199.
- [3] Byun S. (2004). A Technical Trend of Flash Memory Based Data Management for Portable Computers. *Proceedings of the IIEK 2004*. 27(1). Korea, Seoul. 823-826.
- [4] Byun S. (2004). Framework for Flash Memory Data Processing. *Proceedings of the KSII Fall*. 5(2). Korea, Daejeon. 207-210.
- [5] Chang L., & Kuo T. (2002). An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems. *Proceedings of the 8th IEEE Real-Time and Embedded Technology Symposium*. California, San Jose. 187-196.
- [6] Choi M., Yoon H., Song E., Kim Y., Jin S et al. (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.
- [7] Garcia-Molina H., & Salem K. (1992). Main Memory Database Systems: An Overview. IEEE Transactions on Knowledge and Data Engineering 4(6). 509-516.
  - [8] Greer H., & 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.
  - [9] Gruenwld L., & Eich M. H. (1991). MMDB reload algorithms. Proceedings of the ACM SIGMOD. Denver. 397-405.
  - [10] Gyeung K., & Lee G. (2003). E-Catalog Evaluation Criteria and Their Relative Importance. Journal of Computer Information Systems. 43(4). 55-62.
  - [11] Hui D., Michael N., & 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.
  - [12] Jack K., & Hector G., (1988). Optimizing Shadow Recovery Algorithms. IEEE Transactions on Software Engineering. 14(2). 155-168.
  - [13] Jeanna N. M., Drew R., Adam M. C., Randolph Y., & 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.
  - [14] JFFS. (2005). Retrieved from <http://developer.axis.com/software/jffs/>
  - [15] JFFS2. (2005). Retrieved from <http://source.redhat.com/jffs2/>.
  - [16] Kim H., & Lee S. (1999) A New Flash Memory Management for Flash Storage System. Proceedings of the 23rd Annual International Computer Software and Applications Conference. Arizona, Phoenix. 284-289.
  - [17] Kim S., Choi W., & 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.
  - [18] Kuanchin C., Rea J., & Alan L. (2004). Protecting Personal Information Online: A Survey Of User Privacy Concerns and Control Techniques. Journal of Computer Information Systems. 44(4). 85-92.
  - [19] Kun-Lung W., & Kent F. (1993). Rapid Transaction-Undo Recovery Using Twin-Page Storage Management. IEEE Transactions on Software Engineering. 19(2). 155-164.
  - [20] Matthew S. H., & John D. G. (1983). Shadowed Management of Free Disk Pages with a Linked List. ACM Transactions on Database Systems. 8(4). 503-514.
  - [21] Mendel R., & John K. O. (1992). The design and implementation of a log-structured file system, ACM Transactions on Computer Systems. 10(1). 26-52.
  - [22] Pons A. P. (2003). Enhancing The Quality-Of-Service For Application Service Providers. Journal of Computer Information Systems. 44(1). 3-8.
  - [23] Richard G. V., Carl S. G., & Michacel T. V. (1997). Electronic Commerce on the WWW/Internet. 38(1). 20-25.
  - [24] Song E. M., Kim Y. K., & 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.
  - [25] Vijay K., & 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.
  - [26] Yim K. & Koh K. (2003). A Study on Flash Memory Based Storage Systems Depending on Design Techniques. Proceedings of the Information Science Conference. 30(2-1). 274-276.

**Changbae Roh** received his B.S. degree in Computer engineering from Daejeon University in 2001 and earned his M.S. in Computer education from Hannam University in 2003 and in the process of getting doctor's degree Radio engineering from KyungHee University.

**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 Korea Advanced Institute of Science and Technology (KAIST) in 1991 and 1999. He is now with Anyang University Digital Media Department as associate professor.