Mobile Information Systems 6 (2010) 177–191
DOI 10.3233/MIS-2010-0098
IOS Press

Recovery of flash memories for reliable mobile storages

Daesung Moon[a], Byungkwan Park[b], Yongwha Chung[c,*] and Jin-Won Park[d]
[a]*Knowledge Information Security Research Department, ETRI, 161 Gajeong-dong, Yuseong-gu, Daejeon, 305–700, Korea*
[b]*Department of Computer and Information Science, Sunmoon University, Asan, Chungnam 336–708, Korea*
[c]*Department of Computer and Information Science, Korea University, Jochiwon, Chungnam 339–700, Korea*
[d]*School of Games, Hongik University, Jochiwon, ChungNam 339–701, Korea*

Abstract. As the mobile appliance is applied to many ubiquitous services and the importance of the information stored in it is increased, the security issue to protect the information becomes one of the major concerns. However, most previous researches focused only on the communication security, not the storage security. Especially, a flash memory whose operational characteristics are different from those of HDD is used increasingly as a storage device for the mobile appliance because of its resistance to physical shock and lower power requirement. In this paper, we propose a flash memory management scheme targeted for guaranteeing the data integrity of the mobile storage. By maintaining the old data specified during the recovery window, we can recover the old data when the mobile appliance is attacked. Also, to reduce the storage requirement for the recovery, we restrict the number of versions to be copied, called Degree of Integrity (DoI). Especially, we consider both the reclaim efficiency and the wear leveling which is a unique characteristic of the flash memory. Based on the performance evaluation, we confirm that the proposed scheme can be acceptable to many applications as a flash memory management scheme for improving data integrity.

Keywords: Mobile storage, flash memory, data integrity

1. Introduction

As the value and the importance of the information stored in a storage device are increased proportionally with the storage capacity, there has been a growing interest in protecting the stored information against external attackers. Especially, with USB flash drives replacing floppy disks at the time of greater concern for security, sneakernet bandwidth has gone up considerably. Add to this mix the rise of mobile banking and other fiscal uses of mobile devices with embedded storage, and the risks increase dramatically [1].

However, *protecting information stored in a storage device* is a challenging problem because of the following reasons: third-party management of the corporate storage, widespread use of the mobile storage, and vulnerability of these devices to loss/theft/capture. Thus, *protecting information stored in mobile appliances* becomes an emerging topic in the storage system area. However, most previous

*Corresponding author. Tel.: +82 41 860 1343; Fax: +82 41 864 0014; E-mail: ychungy@korea.ac.kr.

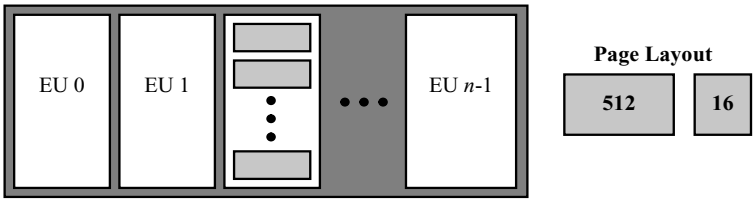

Fig. 1. The internal structure of a typical flash memory.

researches focused on either the mobile communication security issue [2–4] or the server-based storage security issue [5–8].

In this paper, we design and implement the mobile appliance storage for enhancing data integrity. Especially, we select a *flash memory* as our mobile appliance storage because of its resistance to physical shock and lower power requirement than Hard Disk Drive (HDD) [9–12]. We first propose the *Secure Flash Storage* by considering the characteristics of the flash memory and the requirements for data integrity, and then evaluate the performance of it with a trace-driven simulation.

The rest of the paper is structured as follows. Section 2 explains the overview of a flash memory and typical flash memory management schemes, and then describes the security issues in storage. Section 3 describes the proposed flash memory management scheme for improving data integrity, and the results of the performance evaluation are described in Section 4. Finally, conclusions are given in Section 5.

2. Background

A flash memory [12] is a type of Electrically Erasable Programmable Read Only Memory (EEPROM), and has the following characteristics over the HDD.

- Pros: small, light-weighted, robust; low power consumption; faster read access times.
- Cons: slower write access times; no in-place-update (needs an erase operation); limited lifetime ($<$ 100,000 times erasure); more expensive than disk (about 10 times); difficult to manage.

As shown in Fig. 1, the internal structure of a flash memory is defined by an *Erase Unit* (*EU*) which consists of 32$\sim$128 pages, and each page consists of a data area (512/1024/2048B) and a spare area (16/32/64B). In this paper, we consider 1*Gbit* NAND-type flash memories where the sizes of an EU, a page data area, a spare area are 16*KB* (32 pages), 512B, and 16B, respectively.

The basic operations defined over the flash memory are; "read a page" which takes less than 20*usec*, "write (or program) a page" which takes less than 200*usec*, and finally "erase an EU" which takes less than 2*msec*. Note that, the basic unit of the read/write operation is a page whereas an EU is the basic unit of the erase operation. Because the erase operation is the most time consuming one, we need to be careful to manage it. The research topics related with the erase operation can be summarized as follows;

- Performance: reclaim (or garbage collection) efficiency.
- Wear Leveling: a limited number (about 100,000) of erases for each EU; need to erase EUs evenly.

Typical flash memory management schemes can be classified into two classes; Flash-Specific File Systems which are based-on "log-structured file system", such as JFFS [13], YAFFS [14]. The other one is *Flash Translation Layer* (*FTL*) [15] which emulates a block device using a flash memory and its upper layer interface is the standard file system, such as FAT [16]. We add some "recovery utility" to FTL in order to improve the data integrity of the flash memory.

The security issues in storage [17] can be summarized as follows;.

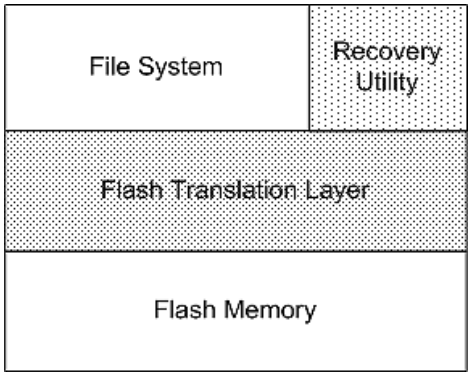

Fig. 2. Our Secure Flash Storage management scheme.

– Confidentiality: ensuring authorized users to access data, and can be achieved by encryption.
– Integrity: maintaining data consistency against accidental and malicious attacks to data through IDS or rollback.
– Performance: there is a tradeoff between the security level and the cost, and the most dominant cost is for encryption.

Several secure storage systems have been developed by considering the above security issues. For example, Network File System (NFS) [18] emphasizes the user authentication, Cryptographic File System (CFS) [19] provides the end-to-end security service by encrypting stored data, and Storage-based Intrusion Detection System (SIDS) [20] does not trust even the host machine operating system. Note that, all these approaches are "server-based" ones. To the best of our knowledge, the research result for securing "mobile storages" has not been reported yet.

3. Design of a secure flash storage

In this paper, we propose a new Flash Translation Layer (see Fig. 2) to improve the security of a mobile storage device for mobile appliances. That is, we apply some security concepts to the flash memory management scheme. Especially, we focus on the data integrity and propose some techniques such as roll-back after the attack, usage of timestamp, and page state managing. We assume that the data content can be protected by performing the user authentication first, rather than encrypting/decrypting repeatedly for all the data stored in the mobile storage. Of course, depending on the importance of the data, the data can be encrypted additionally. The issue of the user authentication is out of the scope of this research, but can be found in [21]. We also assume that the attack such as malware and denial-of-service is already detected using the security techniques [21]. Note that, regardless of whether we use the security techniques, we should have a recovery plan for incidents that will inevitably occur [22].

3.1. Data structures and state transition of page

For the purpose of explanation, we first describe the definition of the spare area for the NAND flash memory [23] in Fig. 3. Logical Sector Number (LSN) stores the information necessary for the mapping between the logical and the physical addresses. In this research, we utilize the reserved bytes (RSV) to design the Secure Flash Storage.

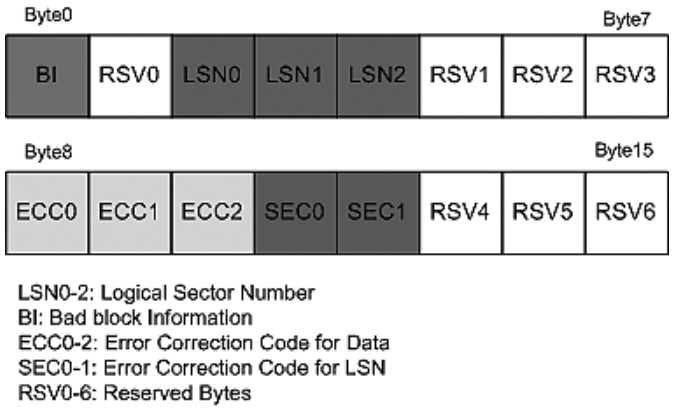

Fig. 3. Definition of the spare area for NAND flash memory [23].

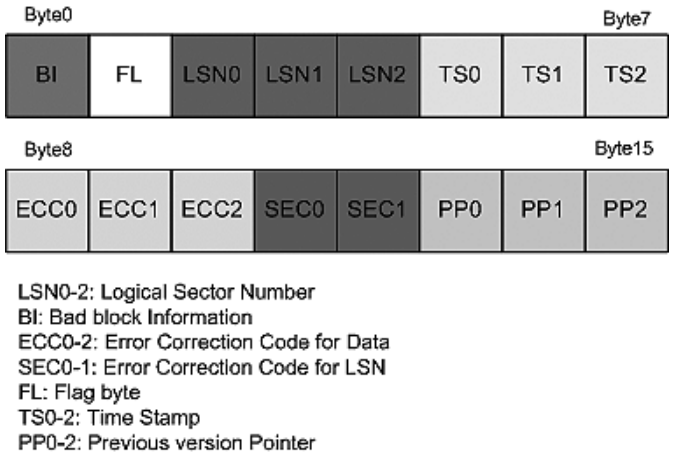

Fig. 4. New definition of the spare area for Secure Flash Storage.

Figure 4 shows the new definition of the spare area for our Secure Flash Storage. FL (Flag byte) represents the page state. In our design, we define a new state, called *Old* (O), in addition to *Free* (F), *Valid* (V), *Invalid* (I). Then, by maintaining the old data specified during the *Recovery Window*, we can recover the old data when the mobile appliance is attacked. TS (Time Stamp) in Fig. 4 represents the time for page write, and PP (Previous version Pointer) is for recovery. Especially, by following PP, we can reach the old data before the update.

According to the new definition of a state, we can represent the state transition of a page as Fig. 5. Note that, in order to save the extra space, the transition shown as "Update w/ Invalid" in Fig. 5 is defined for frequently updated data.

Then, we describe the necessary data structures for our Flash Translation Layer. *Direct Map* shown in Fig. 6(a) contains the logical-to-physical mapping information. It is built with Inverse Map in the mounting time and stored into RAM. On the contrary, *Inverse Map* shown in Fig. 6(b) contains the physical-to-logical mapping information in the spare area of the flash memory. The fields shown as bold ones are defined to implement our Secure Flash Translation Layer.

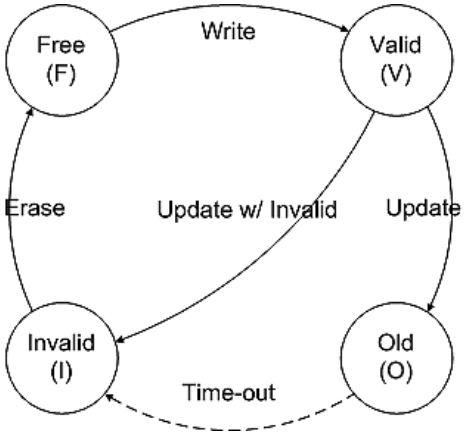

Fig. 5. State transition diagram of a page.

```
struct direct_map_t {
    unsigned char state : 1;     // logical block state
    unsigned int vnum : 7;       // number of versions
    unsigned short eun;          // physical page address   EU number
    unsigned char pgn;           // physical page address   Page number
} [LOGICAL_SPACE_SIZE];
```

(a)

```
struct page_t {
    char bi;                     // bad block information
    char fl;                     // flag: page state
    unsigned int lsn : 24;       // logical sector number
    unsigned int ts : 24;        // time stamp
    unsigned int ecc : 24;       // ECC for data
    unsigned short sec;          // ECC for LSN
    unsigned short pp_eun;       // previous version pointer   EU number
    unsigned char pp_pgn;        // previous version pointer   Page number
};
```

(b)

Fig. 6. (a) Direct map and (b) Inverse map.

3.2. Basic operations

We focus on the Page Update, the EU Reclaim, and the Space Thinning operations in this paper, although the Page Read and the Page Write operations are also the basic operations. And, the page allocation step in the Page Write and the Page Update is described in the EU Reclaim.

3.2.1. Page Read

To read a data, the logical block address needs to be translated to a corresponding physical block address (i.e., the EU number and the page number) through Direct Map stored in the main memory. Since the page read operation does not change any state information, it does not cause any update on Direct Map.

3.2.2. Page write

Page write operation writes a new value into an initially empty state (i.e., “E” in Direct Map), whereas page update operation writes a new value into an exist state (i.e., “X” in Direct Map). The page write procedure is as follows.

Step 1. Allocate new “F” state page.
Step 2. Write data on the page.

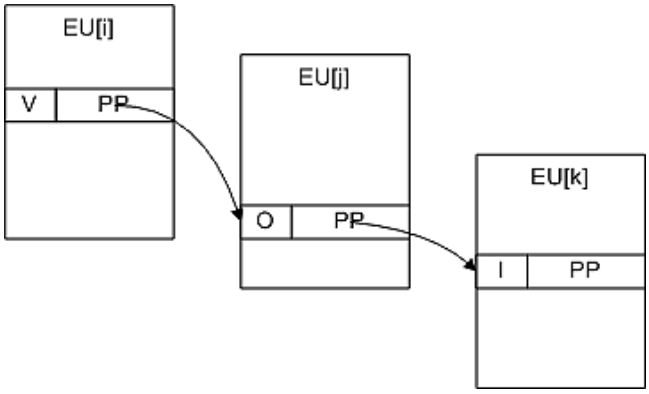

Fig. 7. Result of two page updates.

Step 3. Write management information on the spare area – change state "F" into "V" and timestamp. Since this operation write a new value into an empty state, previous version pointer (pp_eun, pp_pgn shown in Fig. 6(b)) is initialized with 1.
Step 4. Update the Direct Map (i.e., write the EU number and the page number, and change state "E" into "X").

3.2.3. Page update

The page update procedure is as follows, and the result of two updates is shown in Fig. 7. For example, after the first update, the states of the previous EU[k] and the current EU[j] become "O" and "V", respectively. After the second update, EU[k] may be changed as "I" state, which means the timestamp of EU[k] is beyond the Recovery Window.

Step 1. Allocate new "F" state page.
Step 2. Write data on the page.
Step 3. Write management information on the spare area – change state "F" into "V", timestamp, previous version pointer.
Step 4. Update previous version – change state "V" into "O".
Step 5. Update the Direct Map.

3.2.4. EU reclaim

With frequent updates, many "I" and "O" are created and "F" states are decreased. When we select an EU to reclaim, we should consider the reclaim efficiency and the wear leveling. In our scheme, we select an EU with the highest score computed in the following equation. In the following equation, we denote *valid* (j), *old* (j) and *invalid* (j) as the numbers of pages in EU (j) with the corresponding states respectively. We set $\sigma(j)$ be the sum of the number of pages for recent versions of "O" states. Also, we setλbe the parameter for wear leveling. That is, λ approaches to 1 if there happens a problem with wear leveling and this case makes the second term large, resulting in selecting the EU. Otherwise, λapproaches to 0 and we select the EU depending on the value of the first term in the following equation.

$$score(j) = (1-\lambda)\left(\frac{valid(j)+invalid(j)+old(j)}{valid(j)+\sigma(j)}\right)+\lambda\left(\frac{\max_i\{erasures(i)\}}{1+erasures(j)}\right) \quad (1)$$

where $0 < \lambda(\max_i\{erasures(i)\} - \min_i\{erasures(i)\}) < 1$.

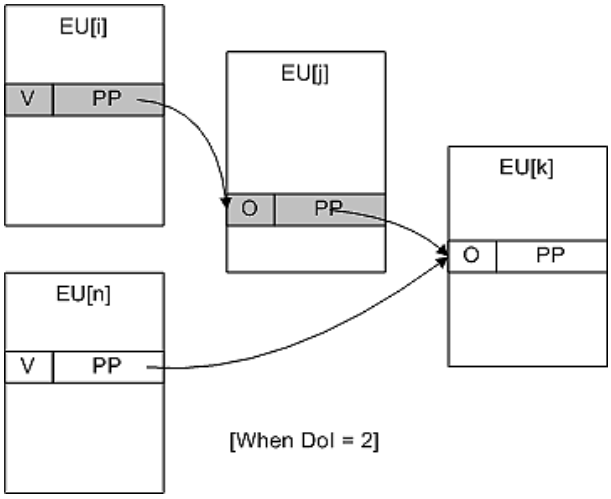

Fig. 8. Illustration of Space Thinning operation (DoI = 2).

```
struct eu_t {
        unsigned short erase_count;          // EU s erasure count
        float score;                         // score for reclaim
        struct pg_t {
                unsigned char state;         // pages state
                unsigned short pp_eun;       // previous pointer
                unsigned char pp_pgn;        // previous pointer
                unsigned int np_eun;         // next version pointer
                unsigned char np_pgn;        // next version pointer
                unsigned int ts ;            // timestamp
        } page[PG_NUM];
} EU [EU_NUM];
```

Fig. 9. EU state table stored in RAM.

3.2.5. Space Thinning

Increasing the Recovery Window size can enhance the data integrity with additional storage space. To balance the data integrity and the recovery space overhead, we introduce a notation of "*Degree of Integrity* (*DoI*)", which restricts the number of versions to be stored in order to make the versions to be evenly distributed in Recovery Window. DoI should be determined based on the characteristics of the applications and the size of the mobile storage device. The Space Thinning operation can be summarized as follows.

Step 1. When the number of versions reaches to ($2 \times DoI - 1$), start the following space thinning operation.
Step 2. Thinning out the even numbered versions from the tail of the list.
Step 3. The remained versions must be copied to new pages.

The illustration of the Space Thinning operation and the data structure required for reclaim and thinning are shown in Figs 8 and 9, respectively.

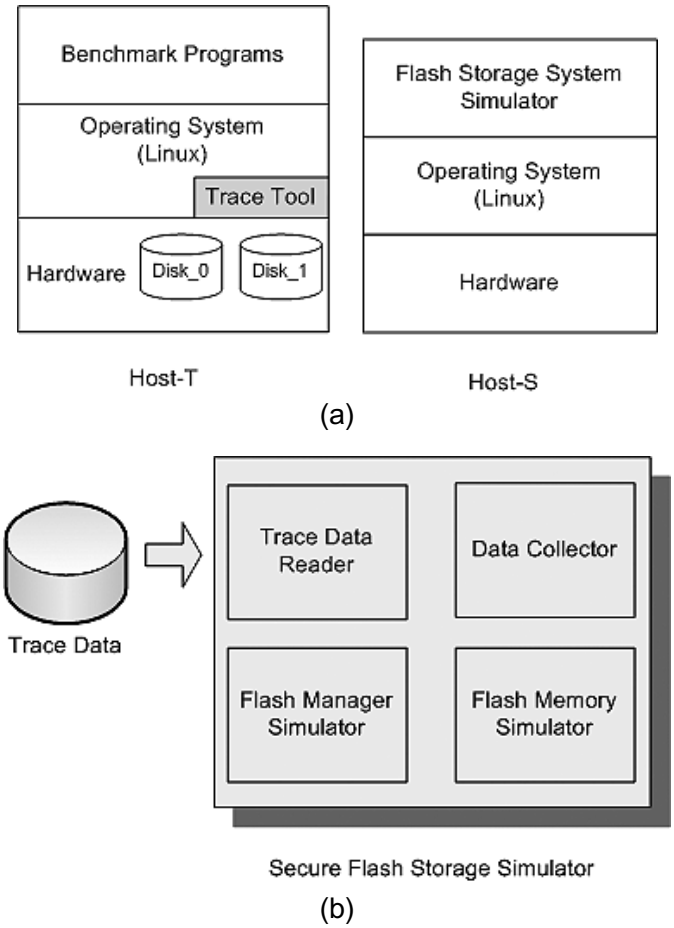

Fig. 10. (a) Experimental environment and (b) Structure of trace-driven simulator.

4. Performance evaluations

Since it is difficult to analyze the performance of the proposed Secure Flash Storage formally, we analyze it with a trace-driven simulation. Note that, most flash-based and/or disk-based previous researches also conducted a trace-driven simulation to verify their performance [24–26]. Especially, we want to verify that the caused overhead during normal operations is acceptable although it depends on DoI. The simulator for the Secure Flash Storage is driven by the trace data that were obtained by a patched Linux device driver to trace the device activities using the I/O benchmark program.

4.1. Simulation environment

We used one Linux-based server (Host-T) to obtain the trace data and another Linux-based server (Host-S) to develop and execute our simulator (see Fig. 10(a)). The trace tool for obtaining the trace data was implemented at the Linux device driver and we used the Disk Trace Tool developed at Bringham Young University [27]. Also, we used the benchmark program, called IOzone [28], to measure the I/O

```
struct trace_t {
        char type;              // access type: r read, w write
        char major;             // device s major number
        char minor;             // device s minor number
        char addr_type;         // L: LBA, C: CHS, S: SCSI
        unsigned int size;      // I/O request size
        unsigned int addr;      // IO Address
        unsigned int time;      // request time
};
```

Fig. 11. Trace data.

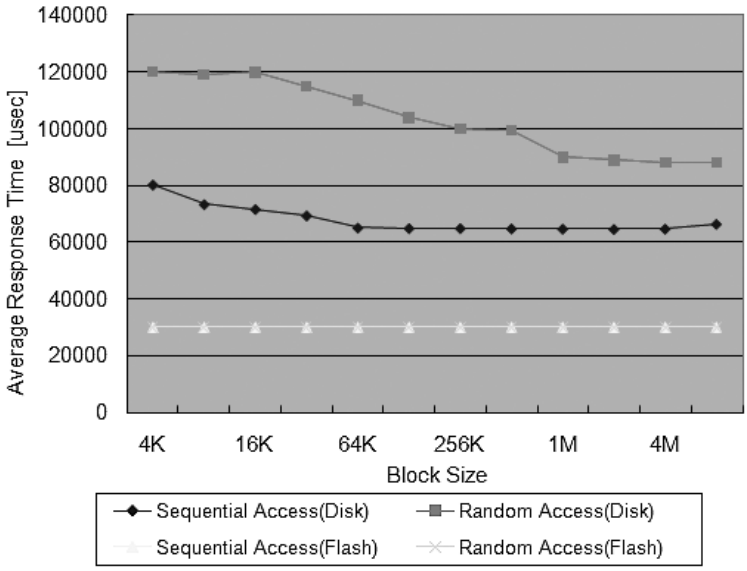

Fig. 12. Characteristics of flash memory and disk.

performance. Our simulator was implemented with C, and has the structure shown in Fig. 10(b). The data structure for the trace data is shown in Fig. 11.

4.2. Performance evaluation results

Figure 12 shows the difference of the average response times between the flash memory and the disk. Because the disk seek time is relatively large, the performance for sequential access and larger block is superior to that of random access and smaller block, respectively. On the contrary, the performance of the flash memory is almost independent of the access pattern (i.e., sequential vs. random) and the access size (i.e., large vs. small). The critical performance factor in the flash memory is the delay caused by the EU Reclaim. Also, the decision factor to the EU Reclaim execution is the *write bandwidth* which means the total data size to be written to the flash memory.

Figure 13 shows the characteristics of the Page Write operation. The 200 *usec* response time means the first write to a page, and the response time increases with the following page updates due to the required invalidate and write operations. If the write operations are executed increasingly, the “F” state pages decreases and the EU reclaim is needed. Once the EU Reclaim operation is initiated, however, the

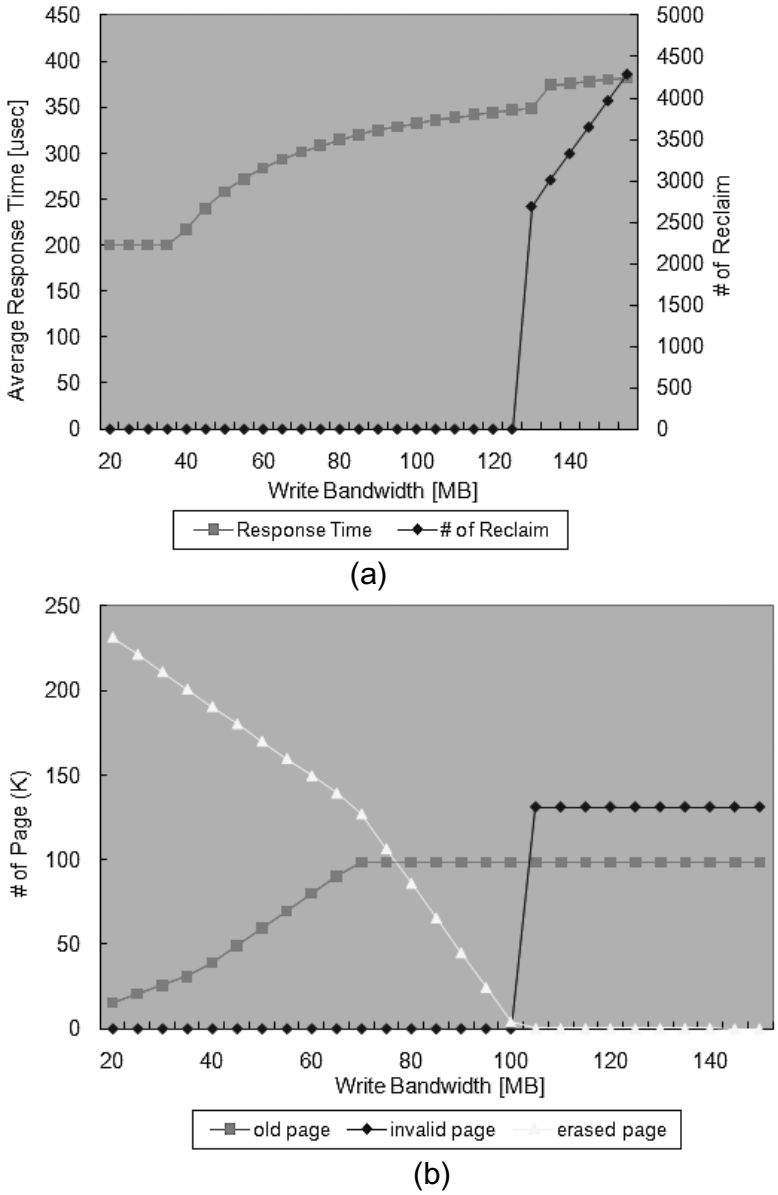

Fig. 13. Characteristics of page write (DoI = 3, sequential access pattern).

response time increases significantly. This is because the required time for the EU Reclaim is quite long (2*msec*) and the flash memory cannot do other operations during this EU Reclaim. The page state change in Fig. 11 also shows that maintaining "O" state pages increases the response time and requires more storage space. However, when we set DoI, the number of "O" state pages does not increase exponentially but approaches asymptotically to a certain level. This is due to the Space Thinning operation, and the number of "I" state pages increases. Also, when the number of "F" state pages is reduced to a certain level, the EU Reclaim operation is initiated.

In Fig. 14, the EU Reclaim operation causes the long response time when the write bandwidth reaches

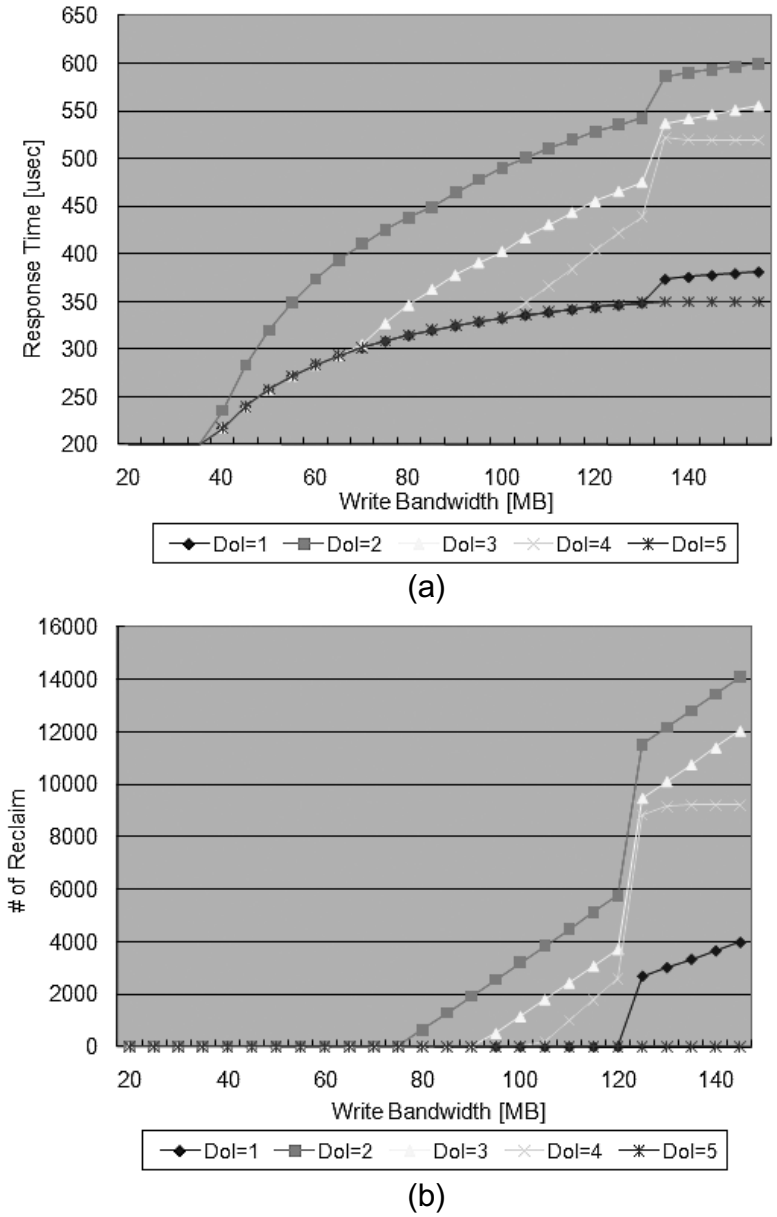

Fig. 14. (a) Page write time, (b) Reclaim frequency.

to 40*MB*/*sec* and the number of "F" state pages is reduced. DoI=2 needs more EU Reclaim operations than larger DoIs, and the jumps near 128*MB*/*sec* were from the target size (128*MB*) of our Secure Flash Storage.Also, the number of EU Reclaim operations is proportional to the write bandwidth.

Figure 15 shows the characteristics of the Space Thinning operation. When DoI is 2, the Space Thinning is initiated with lower write bandwidths and is increased proportionally with the write bandwidth. When we choose larger DoI, the first occurrence of the Space Thinning is delayed and the number of Space Thinning operations is reduced.

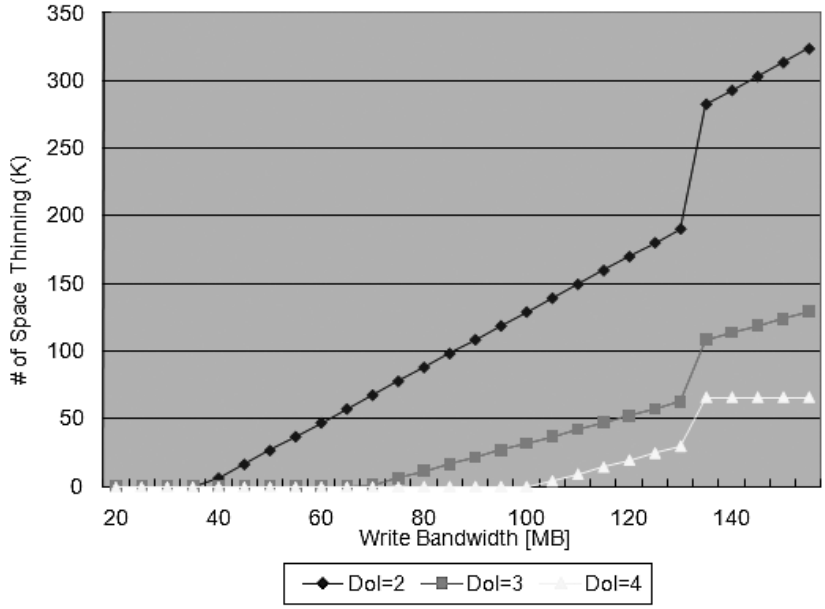

Fig. 15. Characteristics of Space Thinning.

Figure 16 shows the space overhead with different size of the Recovery Window (DoI = 10), with different trace data (DoI = 3), and with different DoI (recovery=20%). The Recovery Window shown as Fig. 16(a) was set as the relative value (i.e., 10%, 20%, . . . , 100%) to the total trace time. The size of additional storage space is proportional to the increased size of the Recovery Window, but we know that we can control the size of the storage space depending on the application program and the DoI value. Also, the fact that the space overhead is not increased even with larger write bandwidths (see Fig. 16(c)) was due to the Space Thinning operation.

The results from the trace drvien simulation may not show the exact performance metrics for the Secure Flash Storage. However, the simulation results show the availability and superior performance of the Secure Flash Storage over hard disks for a typical workload.

5. Conclusions

As the mobile appliance is applied to many ubiquitous services and the importance of the information stored in it is increased, the security issue to protect the information needs to be considered. Although many researches reported the mobile communication security issue, the mobile storage security issue, especially with the flash memory whose operational characteristics are different from those of HDD, has not been reported.

In this paper, we proposed a flash memory management scheme, called *Secure Flash Storage*, targeted for guaranteeing the data integrity of the mobile storage. After defining some data structures and the necessary operations, two performance parameters (i.e., *Recovery Window*, *Degree of Integrity*) were derived to balance the security and the overhead by considering both the reclaim efficiency and the wear leveling. Finally, we implemented the Secure Flash Storage, and evaluated the Secure Flash Storage quantitatively with the performance parameters. Based on the trace-driven simulation, we confirm that

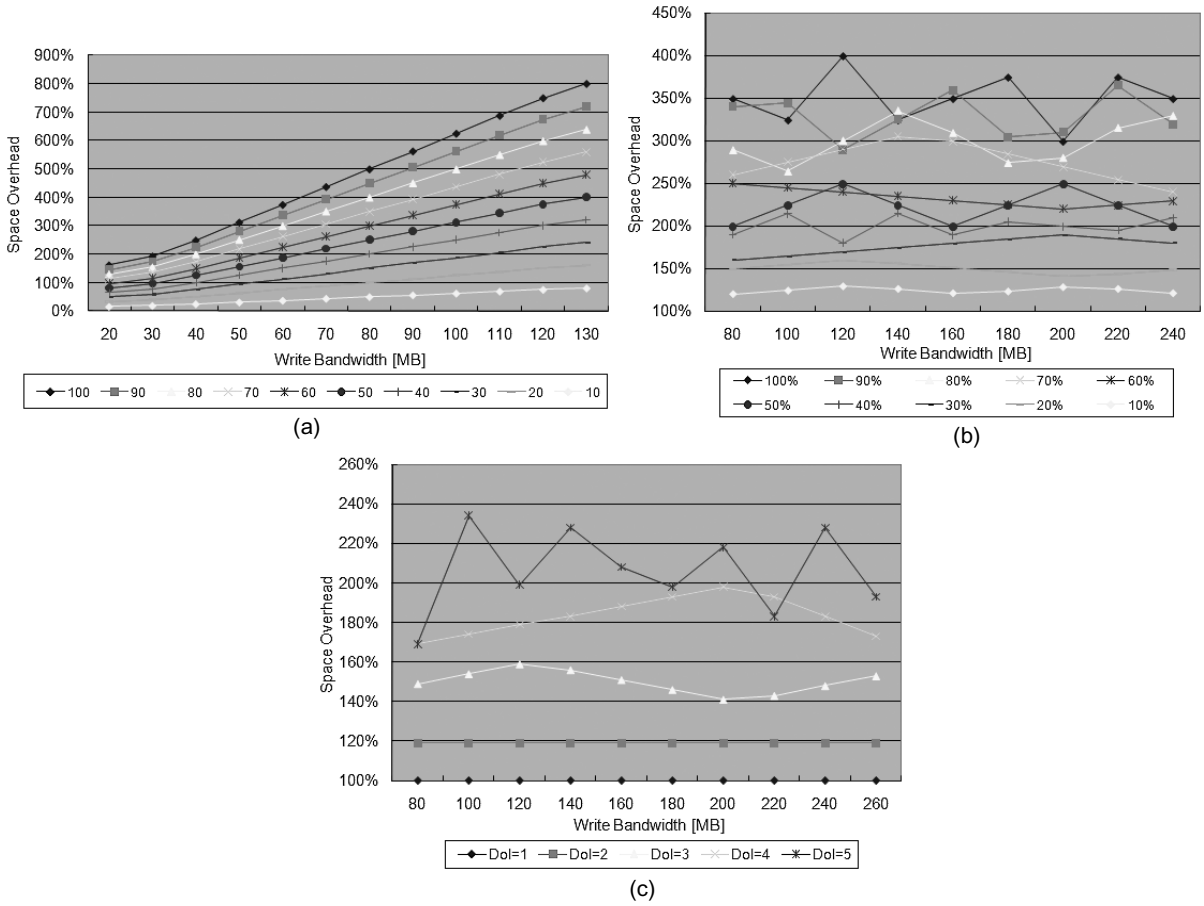

Fig. 16. Space overhead with different parameters; (a) Recovery window, (b) Trace data, (c) DoI.

the proposed scheme can improve the data integrity with an acceptable overhead by controlling the performance parameters.

Acknowledgements

This research was partially financially supported by the Ministry of Education, Science Technology (MEST) and Korea Industrial Technology Foundation (KOTEF) through the Human Resource Training Project for Regional Innovation and partially by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency)

References

[1] J. Cole, Security in Storage: A Call for Participation, *IEEE Computer* **38**(9) (2005), 103–105.
[2] A. Aikebaier, T. Enokido and M. Takizawa, Design and Evaluation of Reliable Data Transmission Protocol in Wireless Sensor Networks, *Mobile Information Systems* **4**(3) (2008), 225–237.

[3] A. Durresi, M. Durresi and L. Barolli, Secure Authentication in Heterogeneous Wireless Networks, *Mobile Information Systems* **4**(2) (2008), 119–130.
[4] D. Venugopal and G. Hu, Efficient Signature based Malware Detection on Mobile Devices, *Mobile Information Systems* **4**(1) (2008), 33–49.
[5] *Proc. of Security in Storage Workshop*, IEEE, 2007.
[6] *Proc. of Security in Storage Workshop*, IEEE, 2005.
[7] *Proc. of Security in Storage Workshop*, IEEE, 2003.
[8] *Proc. of Security in Storage Workshop*, IEEE, 2002.
[9] G. Lawton, Improved Flash Memory Grows in Popularity, *IEEE Computer* **39**(1) (2006), 16–18.
[10] S. Nath and P.B. Gibbons, Online maintenance of very large random samples on flash storage, *Proc. of the 34th conference on Very Large Data Bases* (*VLDB'08*) (2008), 970–983.
[11] K. Park et al., Anticipatory I/O Management for Clustered Flash Translation Layer in NAND Flash Memory, *ETRI Journal* **30**(6) (2008), 790–798.
[12] Samsung, Samsung Solid-State Disk Data Sheet, 2006.
[13] D. Woodhouse, JFFS: The Journaling Flash File System, *Proc. of Ottawa Linux Symposium*, Available at http://sources.redhat.com/ jffs2/jffs2.pdf.
[14] Aleph One, *YAFFS: Yet Another Flash Filing System*, Cambridge, UK, Available at http://www.aleph1.co.uk/yaffs/index.html, 2002.
[15] A. Ban, *Flash File System*, US patent 5,404,485.
[16] Silberschatz et al., *Operating System Concepts*, Wiley, 2003.
[17] E. Riedel, M. Kallahalla and R. Swaminathan, A Framework for Evaluating Storage System Security, *Proc. of the 1st Conference on File and Storage Technologies*, 2002.
[18] B. Pawlowski et al., *The NFS Version 4 Protocol*, SANE, 2000.
[19] M. Blaze, A Cryptographic File System for UNIX, *Proc. of the 1st ACM Conference on Communications and Computing Security*, 1993.
[20] A. Pennington et al., Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior, *Proc. of the 12th USENIX Security Symposium*, 2003.
[21] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Prentice Hall, 2006.
[22] S. Farrell, Portable Storage and Data Loss, *IEEE Internet Computing* **12**(3) (2008), 90–93.
[23] Samsung, *NAND Flash Spare Area Assignment Standard*, 2005.
[24] L. Chang and T. Kuo, Real-time Garbage Collection for Flash Memory Storage Systems of Real-Time Embedded Systems, *ACM Transactions on Embedded Computing Systems* **3** (2004), 837–863.
[25] C. Park et al., Cost-Efficient Memory Architecture Design of NAND Flash Memory Embedded Systems, *Proceedings of the 21st International Conference on Computer Design* (*ICCD'03*) (2003), 474–480.
[26] J. HSIEH, T. KUO and L. CHANG, Efficient Identification of Hot Data for Flash Memory Storage Systems, *ACM Transactions on Storage* **2**(1) (2006), 22–40.
[27] F. Sorenson et al., A System-Assisted Disk I/O Simulation Technique, *Proc. of the 7th International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems* (1999), 296–304.
[28] D. William and D. Capps, *IOzone File System Benchmark*, http://www.iozone.org/, 2006.

Daesung Moon received the MS degree from Busan National University, Korea, in 2001. He received the PhD degree from the Korea University, Korea in 2007. He joined the Electronics and Telecommunications Research Institute (ETRI), Korea, in 2000, where he is currently a Senior Member of the engineering staff in the Human Recognition Technology Research Team. His research areas are biometrics, image processing, and security.

Yongwha Chung received the BS and MS degrees from Hanyang University, Korea, in 1984 and 1986. He received the PhD degree from the University of Southern California, USA in 1997. He worked for ETRI from 1986 to 2003 as a Team Leader. Currently, he is an Associate Professor in the Department of Computer Information, Korea University. His research interests include biometrics, security, and performance optimization.

Byungkwan Park received the BS degree in Electronic Engineering from Hanyang University, and MS degree in Computer Science from Korea Advanced Institute of Science and Technology. He received the PhD degree in Computer Science from Korea University. He is currently a professor in Division of Computer Science and Engineering at Sunmoon University. He previously worked as senior engineer in ETRI. His research interests include computer architecture, embedded systems and storage technologies.

Jin-Won Park graduated from Seoul National University in Korea. He received PhD degree from The Ohio State University in USA in 1987, majoring in industrial and systems engineering. He had been working at Electronics and Telecommunication Research Institute(ETRI) in Korea (1988–1999). He is currently teaching at Hongik University in Korea. His research interest is in the area of the performance evaluation of computer systems.