Journal of Systems Architecture 55 (2009) 332–343

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A survey of Flash Translation Layer ☆

Tae-Sun Chung [a,*], Dong-Joo Park [b], Sangwon Park [c], Dong-Ho Lee [d], Sang-Won Lee [e], Ha-Joo Song [f]

[a] College of Information Technology, Ajou University, Suwon 443-749, Korea
[b] School of Computing, Soongsil University, Seoul 156-743, Korea
[c] Information Communication Engineering, Hankook University of Foreign Studies, Yongin 449-791, Korea
[d] Department of Computer Science, Hanyang University, Ansan 426-791, Korea
[e] School of Information and Communications Engineering, Sungkyunkwan University, Suwon 440-746, Korea
[f] Division of Computer, Pukyung National University, Busan 608-737, Korea

ARTICLE INFO

Article history:
Received 9 May 2007
Received in revised form 11 February 2009
Accepted 23 March 2009
Available online 17 April 2009

Keywords:
Flash memory
Embedded system
File system

ABSTRACT

Recently, flash memory is widely adopted in embedded applications as it has several strong points, including its non-volatility, fast access speed, shock resistance, and low power consumption. However, due to its hardware characteristics, specifically its "erase-before-write" feature, it requires a software layer known as FTL (Flash Translation Layer). This paper surveys the state-of-the-art FTL software for flash memory. It defines the problems, addresses algorithms to solve them, and discusses related research issues. In addition, the paper provides performance results based on our implementation of each FTL algorithm.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Flash memory has inherently strong points compared to a traditional hard disk. These points include its non-volatility, fast access speed, resistance to shocks, and low power consumption. Due to these advantages, it has been widely adopted in embedded applications such as MMC or CF card flash memory, mobile devices including cellular phones and mp3 players, and many others. However, due to its hardware characteristics, a flash memory system requires special software modules to read (write) data from (to) flash memory.

One basic hardware characteristics of flash memory is that it has an erase-before-write architecture [5]. That is, to update a location in flash memory, the location must first be erased before new data can be written to it. The memory portion for erasing differs in size from that for reading or writing [2], resulting in the major performance degradation of the overall flash memory system.

Therefore, a type of system software termed FTL (Flash Translation Layer) has been introduced [1,2,6,9,13,14]. At the core an FTL is using a logical-to-physical address mapping table. That is, if a physical address location mapped to a logical address contains previously written data, the input data is written to an empty physical location in which no data were previously written. The mapping table is then updated due to the newly changed logical/physical address mapping. This protects one block from being erased by an overwrite operation.

When applying the FTL algorithm to embedded applications, there are two major considerations: storage performance and RAM memory requirements. With respect to storage performance, as flash memory has the special hardware characteristics mentioned above, the overall system performance is mainly affected by the write performance. In particular, as the erase cost is much more expensive compared to the write or read cost, it is very important to minimize erase operations. Additionally, RAM memory required to maintain the mapping information is a valuable resource in embedded applications. Thus, if an FTL algorithm requires a large amount of RAM memory, the product cost will be increased.

This paper surveys the-state-of-the-art FTL algorithms. Gal and Toledo [7] also provided algorithms and data structures for flash memory systems. Compared to their work, the present study focuses on FTL algorithms and does not discuss file system issues [12,16,8]. Here, the problem is defined, FTL algorithms are discussed, and related research issues are addressed. Performance results based on our implementation of each of FTL algorithms are also provided.

This paper is organized as follows: In Section 2 the problem is defined. Section 3 shows how previous FTL algorithms can be classified, each of which is explained in depth in Section 4. Section 5 presents performance results. Finally, Section 6 concludes the paper.

☆ The preliminary version of the paper was presented at the 2006 IFIP international conference on embedded and ubiquitous computing (EUC 2006).

* Corresponding author.
E-mail addresses: tschung@ajou.ac.kr (T.-S. Chung), djpart@computing.ssu.ac.kr (D.-J. Park), swpark@hufs.ac.kr (S. Park), dhlee72@cse.hanyang.ac.kr (D.-H. Lee), swlee@skku.edu (S.-W. Lee), hajusong@pknu.ac.kr (H.-J. Song).

1383-7621/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2009.03.005

2. Problem definition & FTL functionalities

2.1. Problem definition

First, operation units in the flash memory system are defined as follows:

Definition 1. A sector is the smallest amount of data which is read or written at a time. That is, a sector is the unit of a read or a write operations.

Definition 2. A block is the unit of an erase operation in flash memory. The size of a block is some multiples of the size of a sector.

Fig. 1 shows the software architecture of the flash file system. This section focuses on the FTL layer shown in Fig. 1. The file system layer issues a series of read or write commands each with a logical sector number, to read data from, or write data to, specific addresses in flash memory. The logical sector number is converted to a real physical sector number of flash memory by some mapping algorithm in the FTL layer.

Thus, the problem definition of FTL is as follows: It is assumed that flash memory is composed of n physical sectors. The file system – the upper layer – considers a flash memory as a block-I/O device that consists of m logical sectors. Given that a logical sector must be mapped to at least one physical sector, the number m is less than or equal to n.

Definition 3. Flash memory is composed of a number of blocks, and each block is composed of multiple sectors. Flash memory has the following characteristics: If a physical sector location in flash memory contains previously written data, it must be erased in units of blocks before new data overwrites the existing data. The FTL algorithm produces a physical sector number in flash memory from the logical sector number given by the file system.

2.2. FTL functionalities

An FTL algorithm should provide the following functionalities:

- Logical-to-physical address mapping: The main functionality of an FTL algorithm is to convert logical addresses from the file system to physical addresses in flash memory.
- Power-off recovery: Even when a sudden power-off event occurs during FTL operations, FTL data structures should be preserved and data consistency should be guaranteed.
- Wear-leveling: FTL should include a wear-leveling function to wear down memory blocks as evenly as possible.

3. A taxonomy for FTL algorithms

In this section, a taxonomy for FTL algorithms is suggested according to features that include addressing mapping, mapping information management, and the size of the RAM table.

3.1. Addressing mapping

3.1.1. Sector mapping

A naive and intuitive FTL algorithm is the sector mapping algorithm [1]. In sector mapping, every logical sector is mapped to a corresponding physical sector. Therefore, if there are m logical sectors recognized by the file system, the row size of the logical-to-physical mapping table is m.

Fig. 2 shows an example of sector mapping. In the example, it is assumed that a block is composed of four pages, resulting in 16 physical pages in total, where each page is organized into a data sector and a spare area. Additionally, if it is assumed that 16 logical sectors exist, the row size of the mapping table is 16. When the file system issues the command – "write(9,A): write data 'A' to lsn (logical sector number) 9", the FTL algorithm writes the data 'A' to psn (physical sector number) 3 according to the mapping table if psn 3 has not had data written to it at an earlier time.

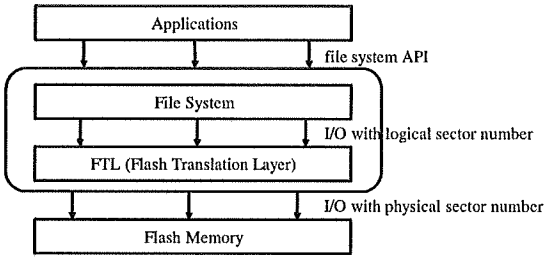


Fig. 1. Overall architecture of flash memory system.

However, in another case, the FTL algorithm determines the location of an empty physical sector, writes data to it, and adjusts the mapping table. If an empty sector does not exist, the FTL algorithm will select a victim block from flash memory, copy the valid data in the victim block to the spare free block, and update the mapping table. Finally, it will erase the victim block, which will become the spare block.

To rebuild the mapping table after a power outage, the FTL algorithm either stores the mapping table to flash memory or records the logical sector number in the spare area upon each write operation to the sector area.

3.1.2. Block mapping

As the sector mapping algorithm requires a large amount of memory space (RAM), it is hardly feasible for small embedded systems. For this reason, block mapping-based FTL algorithms [2,6,13] are proposed. Some detailed algorithms will be presented in Section 4. The basic idea of block mapping is that the logical sector offset within a logical block is identical to the physical sector offset within the physical block.

In the block mapping scheme, if there are m logical blocks detected by the file system, the row size of the logical-to-physical mapping table is m. Fig. 3 shows an example of the block mapping algorithm. Assuming that there are four logical blocks, the row size of the mapping table is four. If the file system issues the command "write(9,A)", the FTL algorithm calculates logical block number 2(=9/4) and sector offset 1(=9%4), and then locates physical block number 1 using the mapping table. As the physical sector offset equals the logical sector offset in the block mapping algorithm, the physical sector location can be easily determined.

It is clear that the block mapping algorithm requires a smaller amount of mapping information when compared to sector mapping. However, if the file system issues write commands with identical logical sector numbers, many copy and erase operations are required, which severely degrades performance. When implementing the block mapping algorithm, block-level mapping information should be stored in flash memory to recover from a power-off event.

3.1.3. Hybrid mapping

As both sector and block mapping have some disadvantages, as mentioned in the previous two subsections, hybrid mapping approaches were introduced [9,10,14]. A hybrid technique, as its

IPR2025-00212/PHI EX. 1064 Page 2 of 12

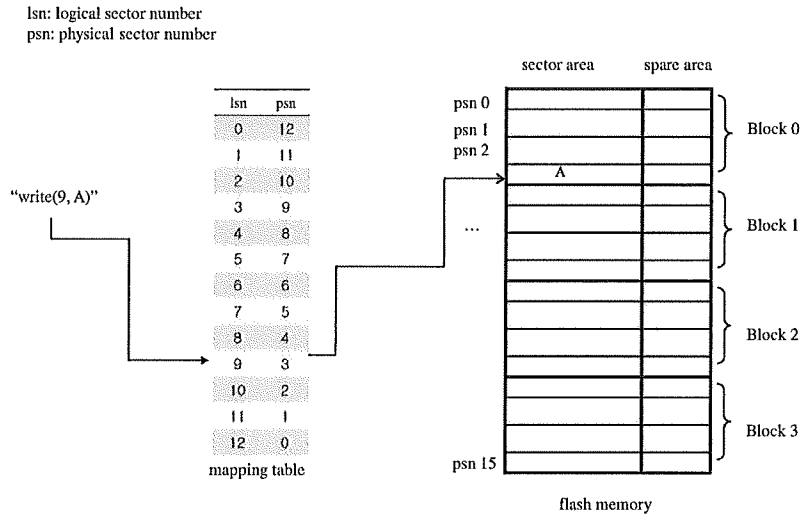


Fig. 2. Sector mapping.

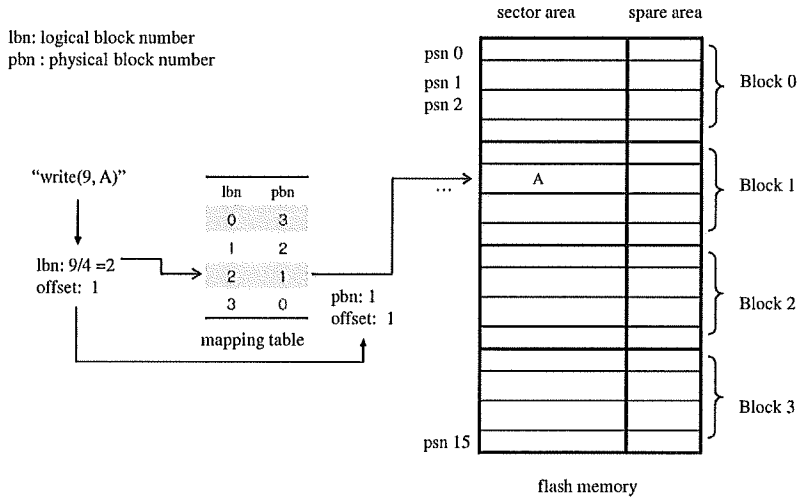


Fig. 3. Block mapping.

name suggests, first uses a block mapping technique to obtain the corresponding physical block. It, then, uses a sector mapping technique to locate an available empty sector within the physical block.

Fig. 4 shows an example of the hybrid technique. When the file system issues the command "write(9,A)", the FTL algorithm calculates the logical block number 2(=9/4) for the lsn, and then, locates physical block number 1 from the mapping table. After obtaining the physical block number, the FTL algorithm allocates an empty sector for the update. In the example, because the first sector of physical block 1 is empty, the data is written to the first sector location. In this case, in that the two logical and physical sector offsets (i.e., 1 and 0, respectively) differ from each other, logical sector number 9 should be written to the spare area in page 0 of physical block 1. To rebuild the mapping table, not only this information but also the logical block numbers have to be recorded in the spare areas of the physical blocks.

When reading data from flash memory, the FTL algorithm first locates the physical block number from the mapping table using the given lsn. Subsequently, by reading the logical sector numbers from the spare areas of the physical block, it can obtain the most recent value for the requested data.

3.1.4. Comparison

The performance levels of FTL algorithms are compared in terms of the file system-issued read/write performance and memory requirements to store the mapping information.

The read/write performance of an FTL algorithm can be measured according to the number of flash I/O operations (read, write, and erase), as the read/write performance is I/O-bounded. It is assumed here that the mapping table of an FTL algorithm is maintained in RAM, and that access cost of the mapping table is zero. The read and write costs can then be computed using, respectively, the following two equations. Though the following equations are general, they can be a starting point in design and analysis of FTL algorithms.

$$C_{\text{read}} = xT_r \tag{1}$$

$$C_{\text{write}} = p_i T_w + p_o(xT_r + T_w) + p_e(T_e + T_w + T_c) \tag{2}$$

C_{read} and C_{write} denote the costs of the read and write commands issued from the file system layer, respectively. T_r, T_w, and T_e are the costs of the read, write, and erase commands processed in the flash memory layer.

IPR2025-00212/PHI EX. 1064 Page 3 of 12

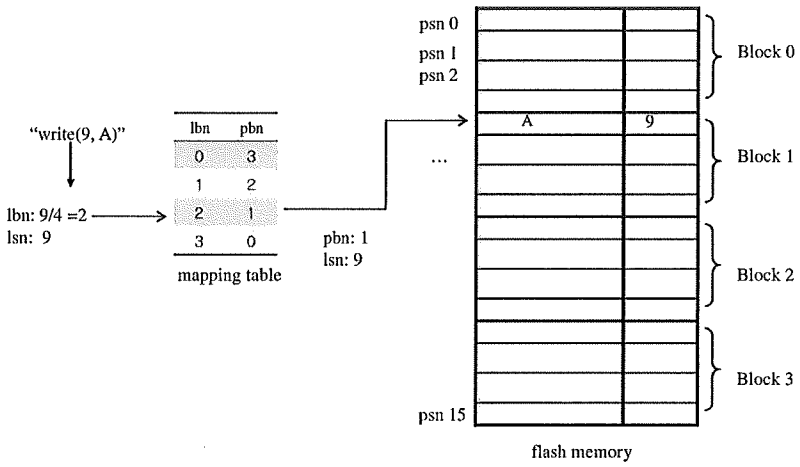


Fig. 4. Hybrid mapping.

When reading, variable x in Eq. (1) is 1 in the sector and block mapping techniques, as the sector to be read can be determined directly from the mapping table. However, in the hybrid technique, the value of variable x ranges from $1 \leqslant x \leqslant n$, where n is the number of sectors within a block. Here, the requested data can be read only after scanning the logical sector numbers stored in the spare areas of a physical block. Thus, the hybrid mapping scheme has a higher read cost compared to the sector and the block mapping techniques.

During a write operation, there are three cases. First, the write operation may be performed in the in-place location directly. p_i is the probability that a write request is performed in the in-place location. Second, the write operation should be performed after scanning the empty position in a block. p_o is the probability of this event. In this case, additional read operations may be required. Finally, the write operation may incur an erase operation. When a write request incurs an erase operation, it is assumed that FTL algorithms perform in as follows: First, valid data in the block to be erased is copied to another block that is a free block, and is copied back to a new block (T_c). Second, the write operation is performed (T_w), third, the mapping table is changed accordingly; and fourth, the source block is erased (T_e).

Given that T_e and T_c are high cost operations relative to T_r and T_w, the variable p_e is a key point when computing the write cost. In sector mapping, the probability of requiring an erase operation per write is relatively low, in block mapping, conversely, it is relatively high.

Another measure of comparison is the memory requirement for storing mapping information. Mapping information should be stored in persistent storage, and it can be cashed in RAM for better performance. Some FTL algorithms use combinations of sector, block, and hybrid mapping in the previous section. However, this paper assumes that each FTL algorithm is used in the overall flash memory system in the following analysis. Fig. 5 shows such memory requirements for the three address mapping techniques. Here, we assume that the capacities of flash memory are 128 MB (with 8192 blocks) and 8 GB (524288 blocks). Furthermore, each block is composed of 32 sectors [5]. In sector mapping, three bytes are needed to represent all sector numbers in both 128 MB and 8 GB flash memory, whereas in block mapping, only two bytes are needed to represent all block numbers in 128 MB flash memory and three bytes in 8 GB flash memory. Hybrid mapping requires two bytes for block mapping and one byte for sector mapping within a block in 128 MB flash memory. In 8 GB flash memory, three bytes for block mapping and one byte for sector mapping. It is clear that block mapping requires the smallest amount of RAM memory as expected.

3.2. Managing address mapping information

When implementing an FTL algorithm, it is necessary to consider a scheme to store mapping information. To be able to rebuild the mapping table during a power-on process, mapping information should not be lost in the sudden power-off events, therefore this information must be persistently kept somewhere in flash memory. The techniques for storing mapping information in flash memory can be classified into two categories: the map block method and the per block method.

3.2.1. Map block method

A map block method stores mapping information into some dedicated blocks of flash memory termed map blocks. Though a block can sufficiently store all mapping information in the case of block mapping, most FTL implementations provide more than one map block. If one map block is used, erase operations on the map block occur very frequently. Hence, several map blocks are used to lessen such frequent erases. Fig. 6 shows how map blocks can be configured to store the mapping information for a block mapping scheme. Mapping information – pairs of a logical block number and a physical block number – is recorded to one of the unused sectors in the latest map block. Physical block numbers are stored in the sector in the order of the logical block number. Of course, if one sector cannot sufficiently store all physical block numbers, two or more sectors are used.

If mapping information changes due to writes issued by the file system, the above recording job will be done. When performing the recording job, if there is no unused sector in the map blocks pool, erase operations have to be executed to free some map blocks. The mapping table can be cached in RAM for fast mapping lookups. In this case, the mapping table has to be rebuilt in RAM by reading the latest sector of the latest map block from flash memory during a power-on process.

3.2.2. Per block method

Mapping information can be stored to each physical block of flash memory. This process assumes that hybrid mapping is being used in Fig. 7. In contrast to the map block method, logical block numbers are stored in the spare area of the first page of each physical block. In addition, to maintain the mapping from a logical sector number to the sectors in a physical block, logical sector

IPR2025-00212/PHI EX. 1064 Page 4 of 12

	Bytes for addressing		Total	
	128MB	8GB	128MB	8GB
Sector mapping	3 B	3B	3B*8192*32=768KB	3B*524288*32=48MB
Block mapping	2 B	3B	2B*8192=16KB	3B*524288=1536KB
Hybrid mapping	(2+1) B	(3+1) B	2B*8192+1B*32*8192=272KB	3B*524288+1B*32*524288 =17920KB

Fig. 5. Memory requirement for mapping information.

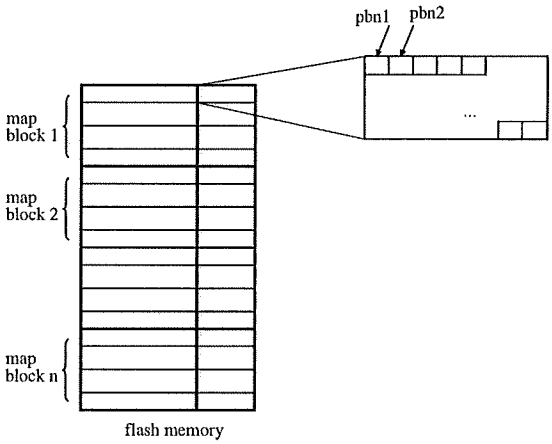


Fig. 6. Map block method.

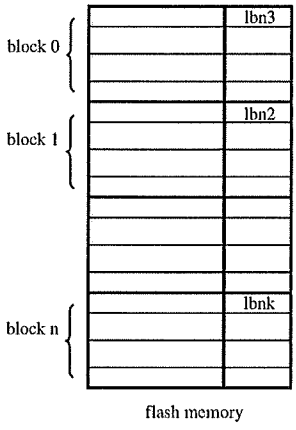


Fig. 7. Per block method.

numbers are recorded in each sector in the block. When rebuilding the mapping table due to a power-off event, both the logical block numbers and logical sector numbers in the spare areas of flash memory are used.

In Fig. 7, logical block number 3 is mapped to physical block number 0, logical block number 2 to physical block number 1, and so on.

3.3. RAM table

The size of the RAM is very important in designing FTL algorithms because it is a key factor in the overall system cost. The smaller the RAM size, the lower the system cost. However, if a system has enough RAM, performance can be improved. FTL algorithms have their own RAM structures and the FTL algorithms can be classified according to their RAM structures. RAM is used to store following information of FTL algorithms.

- Logical-to-physical mapping information: The major usage of RAM is to store the logical-to-physical mapping information. By accessing RAM, the physical flash memory location for reading or writing data can be found efficiently.
- Free memory space information: Once free memory space information in flash memory is stored in RAM, an FTL algorithm can manage the memory space without further flash memory accesses.
- Information for wear-leveling: Wear-leveling information may be stored in RAM. For example, the erase count of flash memory blocks may be stored in RAM.

For example, Fig. 8 shows an example of a RAM table [6]. The flash memory block of this system is composed of 16 sectors. In the example, a 1:2 block mapping method is used. That is, a logical block can be mapped to at most two physical block of flash memory. PBN1 and PBN2 show the first and second physical block addresses. Logical block 00 is mapped to physical blocks 00 and 10 in Fig. 8. The right side of Fig. 8 shows where the valid data is stored. For example, sector 0 of logical block 00 is stored in sector 0 of physical block 10.

In the example RAM table, several flags are used. The 'move' flag indicates that some sectors of one block are stored in another block. The 'used' flag shows that a block is being used. The 'old' flag shows that data in a block is no longer valid, and the 'defect' flag shows that a block is a bad block.

4. Case study

4.1. Mitsubishi

The Mitsubishi algorithm [13] is based on block mapping in Section 3.1.2. Its goal was to overcome the limitations of the sector mapping scheme, that is, (1) the large storage necessary for the map table, (2) the high overhead of the map table construction cost when the power is turned on. One logical block is mapped to one physical block, representing 1:1 block mapping. Compared to the block mapping technique in Section 3.1.2, the Mitsubishi technique suggested a concept of space sectors. That is, a physical block is composed of a general sector area and a space sector area. If a logical block consists of m sectors, a physical block consists of m sectors and some additional n space sectors. Here, the physical sector offsets and the logical sector offsets are identical in the general sector area, and the offsets of the space sectors need not be identical.

IPR2025-00212/PHI EX. 1064 Page 5 of 12

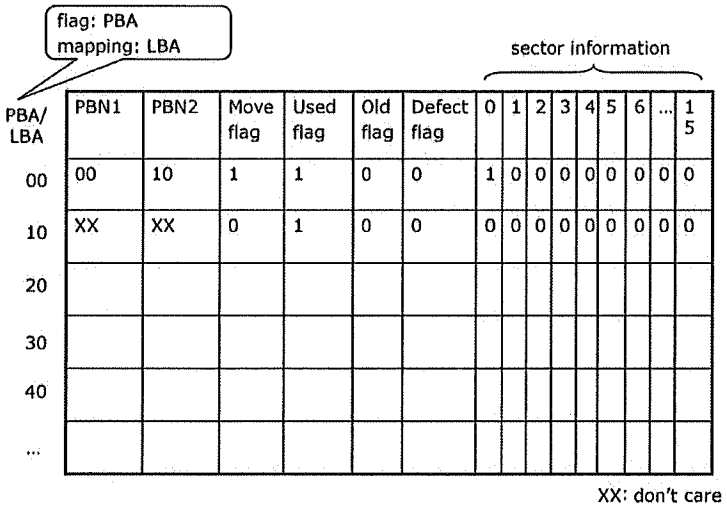


Fig. 8. RAM table.

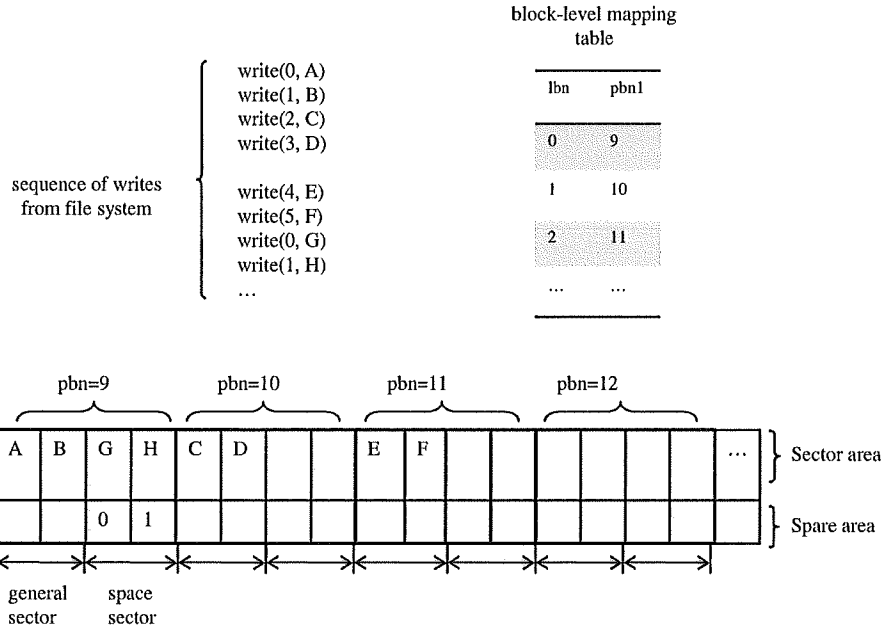


Fig. 9. Running example: Mitsubishi.

Fig. 9 shows an example. This example assumes that a physical block is composed of two general sectors and two space sectors. When processing the first write request 'write(0, A)', the logical block number 0(=0/2) and the logical page offset 0(=0%2) are calculated and the physical block number (9) is obtained using the block-level mapping table. As physical page offset 0 of $pbn = 9$ is empty initially, the data A is written to the first sector location of $pbn = 9$. Additional write operations are performed in the same way. When processing the write request of 'write(0,G)', as the physical page offset 0 of $pbn = 9$ is already occupied, it is written to the first space sector area. In this case, the logical sector number (0) is written to the spare area of flash memory. In the pure block mapping technique, the 'write(0,G)' request incurs a copy and erase operation. When reading the logical sector number 0, the most up-to-date version can be found by scanning the spare areas of flash memory.

A reorganization process is as follows. The FTL algorithm obtains a free block and copies valid sectors from the old physical block to the new block. The content of the logical/physical conversion table is changed appropriately. After the sectors in the original physical block are completely transferred to the new block, the original block is erased and returned to the free block list. The Mitsubishi scheme always keeps at least one free block for reorganization.

The Mitsubishi scheme uses the map block method in Section 3.2.1 for the mapping information; the size of the conversion table is relatively small as it is based on block mapping.

4.2. M-systems

M-systems proposed the algorithms known as ANAND and FMAX for flash memory management systems. Basically, their

IPR2025-00212/PHI EX. 1064 Page 6 of 12

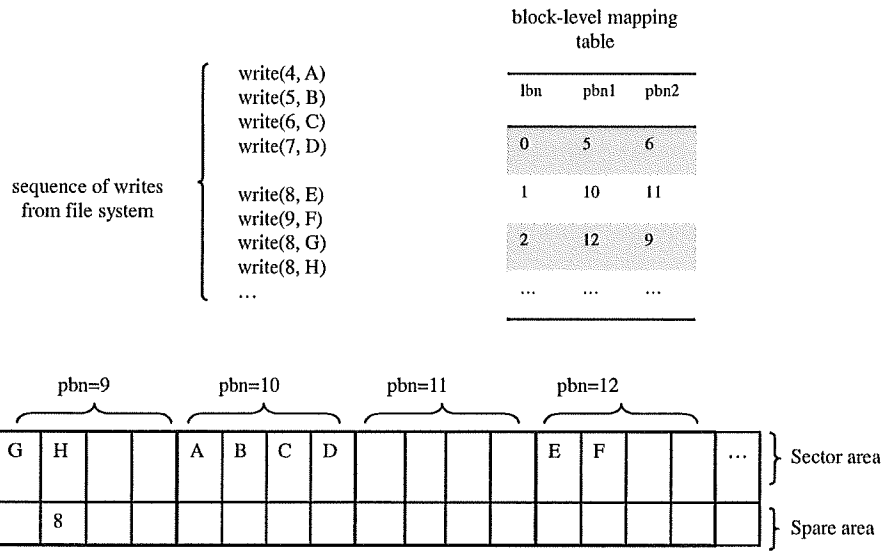


Fig. 10. Running example: M-Systems.

techniques are also based on the block mapping technique. However, compared to the block mapping technique in Section 3.1.2, in their schemes, one logical block can be mapped to more than one physical block. As the basic structure of ANAND is similar to that of FMAX, only the FMAX algorithm is explained in this paper.

In FMAX, one logical block can be mapped to the two physical blocks known as primary block and a replacement block. Here, the logical sector offset is identical to the physical sector offset in the primary block. However, in the replacement block, the logical sector offset may differ from the physical sector offset.

Fig. 10 shows an example. This example assumes, that a physical block is composed of four sectors. When processing the first write request 'write(4,A)', the logical block number 1(=4/4) and the logical page offset 0(=4%4) are calculated, and the physical block number (10) is obtained using the block-level mapping table. Given that the physical page offset 0 of $pbn = 10$'s is empty initially, the data A is written to the first sector location of $pbn = 10$. Additional write operations are performed in the same way. When processing the write request of 'write(8,G)', as $pbn = 12$ is already occupied, the write request is performed in the second physical block ($pbn = 9$). If the second physical block has no free sectors, copy and the erase operation are performed.

FMAX uses the per-block method to store mapping information. In addition, FMAX manages a RAM table to map a logical block to two physical blocks. This requires more RAM size compared to the block mapping in Section 3.1.2.

4.3. SSR

SSR [14] uses the hybrid address mapping scheme in Section 3.1.3. Compared to the previous techniques, the inventors of SSR provide two hash functions when determining a logical block number from a logical sector number. The hash functions are as follows. In the equations, *lsn* is the logical sector number and *ns* is the number of sectors in a block.

$$H1(lsn) = lsn/ns = lbn \quad (3)$$

$$H2(lsn) = lsn\%ns = lbn \quad (4)$$

The SSR algorithms based on the hash functions $H1$ and $H2$ are known as a "clustered mode" and a "scattered mode" respectively. Previous FTL algorithms are based on the hash function $H1$.

Fig. 11 shows a running example (clustered mode). It is assumed here that a physical block is composed of four sectors. When processing the first write request 'write(5,A)', the logical block number 1(=5/4) is calculated, and the physical block number (10) is then obtained using a block-level mapping table. As the physical page offset 0 of $pbn = 10$ is empty initially, the data A is written to the first sector location of $pbn = 10$ and the logical sector number (5) is written to a spare area in flash memory. In SSR, the logical sector offset need not be identical to the physical sector offset. Thus, the logical sector number should be written to the spare area of flash memory. More write operations are performed in the same way. When the read operation is performed, if there is more than one instance of data with the same logical sector number, the most recent data is the first sector from the back end of the block. In the example, the valid data corresponding to logical sector number 9 is the fourth data (H) of $pbn = 11$. If a physical block has no free sectors, copy and erase operation are preformed.

4.4. Log block scheme

Kim et al. [9] proposed a log block based FTL scheme. The main objective of this scheme is to efficiently handle both access patterns efficiently: numerous long sequential writes and a small number of random overwrite operations. To achieve this purpose, the log block scheme maintains most of the physical blocks at the block addressing level – data blocks – and a small fixed number of physical blocks at the sector addressing level – log blocks. Data blocks mainly use storage spaces for long sequential writes and log blocks for random overwrites. Once a sector is initially written to a data block, a overwrite operation to the same logical sector is forwarded to a log block that has been allocated from a pool of log blocks. All subsequent overwrites to the same logical sector are then processed using the log block. If there are no free sectors in the log block, data of the log block is merged to that of corresponding data block and the log block is returned to the pool of log blocks for later overwrites.

If the FTL algorithm cannot locate any free log blocks for the overwrite, it first chooses a victim log block among the log blocks in use, and data of the victim log block is merged with that of corresponding data block. The victim log block is then erased and finally allocated for the current overwrite.

IPR2025-00212/PHI EX. 1064 Page 7 of 12

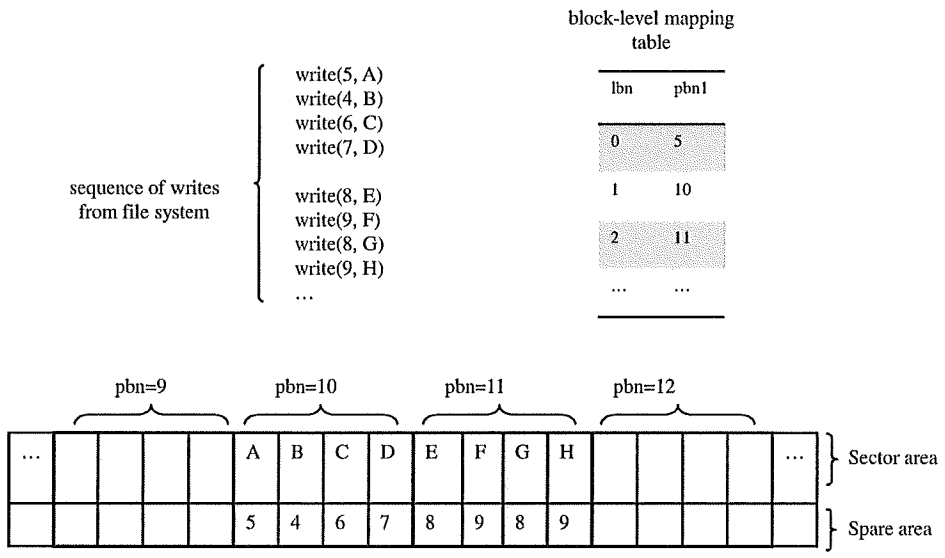


Fig. 11. Running example: SSR.

For address mapping, the log block scheme maintains two different types of mapping tables in RAM: the first is a block mapping table for data blocks and the second is a sector mapping table for log blocks.

Fig. 12 shows an example. This example, assumes that a physical block is composed of four sectors. When processing the first write request 'write(4,A)', the logical block number 1(=4/4) and the logical page offset 0(=4%4) are calculated and the physical block number (10) is obtained using the block-level mapping table. As the physical page offset 0 of $pbn = 10$ is empty initially, the data *A* is written to the first sector location of $pbn = 10$. When processing the write request of 'write(4,C)', given that the physical page offset 0 of $pbn = 10$ is already occupied, it is written to log block $pbn = 20$. The logical sector offset need not be identical to the physical sector offset; hence, the logical sector number is written to the spare area of flash memory. For the log blocks, the sector-level mapping table is constructed as in Fig. 12. More write operations are performed in the same way.

In the log block scheme, the map block method of Section 3.2.1 is adopted to manage the mapping information. The aforementioned block mapping table for data blocks is stored into one of the map blocks. To obtain the latest version of the block mapping table in the pool of map blocks, a map directory is maintained in RAM. For recovery from a power-off event, the map directory is also stored in a specific block of flash memory (a checkpoint block) whenever the sector mapping table for log blocks is updated.

Recently, some variations of the log scheme, including those known as FAST [11] and STAFF [3] have been proposed. In the FAST scheme, more than one logical block can be mapped to a physical block, which improves the space utilization of a log block. The key idea of STAFF is to minimize the erase operation by introducing states that are assigned to blocks and used to control address mapping. The detailed algorithms are omitted due to a lack of space.

5. Evaluation

5.1. Simulation methodology

In the overall flash system architecture presented in Fig. 1, the FTL algorithms presented in Section 4 are implemented. The physical flash memory layer is simulated by a flash emulator that has the same characteristics as a real flash memory.

It is assumed that the file system layer in Fig. 1 is the FAT file system [4], which is widely used in many embedded systems. Fig. 13 shows the disk format of the FAT file system. It includes a boot sector, one or more file allocation tables, a root directory, and the volume files. A recent study [4] contains a more detailed description of the FAT file system. Here, it is clear that logical spaces corresponding to the boot sector, file allocation tables, and the root directory are accessed more frequently compared to the volume files.

For the simulation, various access patterns that the FAT file system issues to the block device driver when it receives a file write request were obtained. The performance results over real workloads of Symbian [15] and Digicam are reported. The first of these is the workload of a 1M byte file copy operation in the Symbian operating system, and the second is the workload of a digital camera. It was found that the access pattern of Digicam is mostly sequential while that of Symbian has many random patterns. In detail, the access pattern of Digicam contains small random writes and frequent large sequential writes. On the other hand, the access pattern of Symbian contains many random writes and infrequent large sequential writes.

5.2. Result

Fig. 14 shows the total elapsed time for the Digicam pattern. The *x* axis is the test count that is the iteration count of workloads and the *y* axis is the total elapsed time in milliseconds. As the size of this workload is small, flash memory is not occupied after performing the workload one time. Thus, to determine the characteristics of FTL algorithms when flash memory is occupied, the processing of the workloads was done many times. Initially, the flash memory was empty. It became occupied as the test count increased.

The result shows that the Log scheme provides the best performance. It is interesting that the Log scheme shows better performance than sector mapping which requires a considerable amount of RAM resources for mapping. This can be explained in that the workload of the Digicam is mostly composed of sequential write operations. Moreover, the Log scheme operates almost ideally in the sequential write patterns. As sector mapping uses the LSN-to-PSN mapping table, it must update the mapping table with every overwrite operation. This implies that it has to write the

IPR2025-00212/PHI EX. 1064 Page 8 of 12

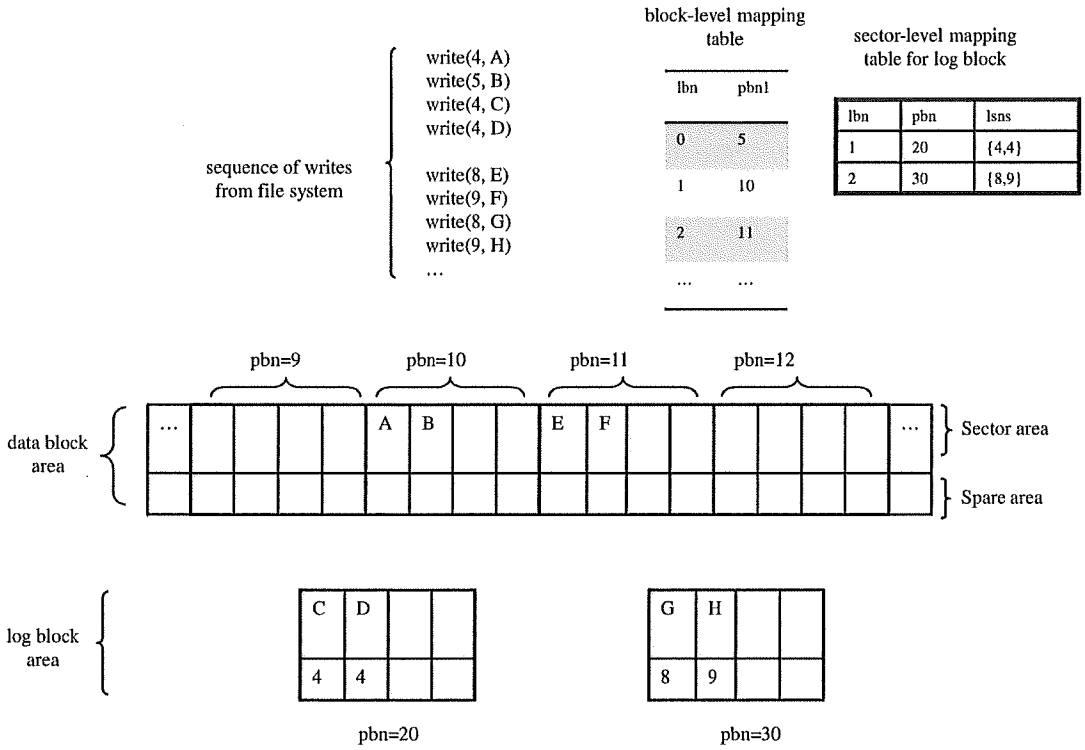


Fig. 12. Running example: Log scheme.

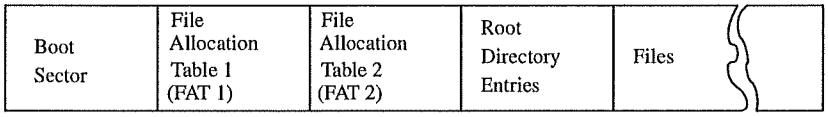


Fig. 13. FAT file system.

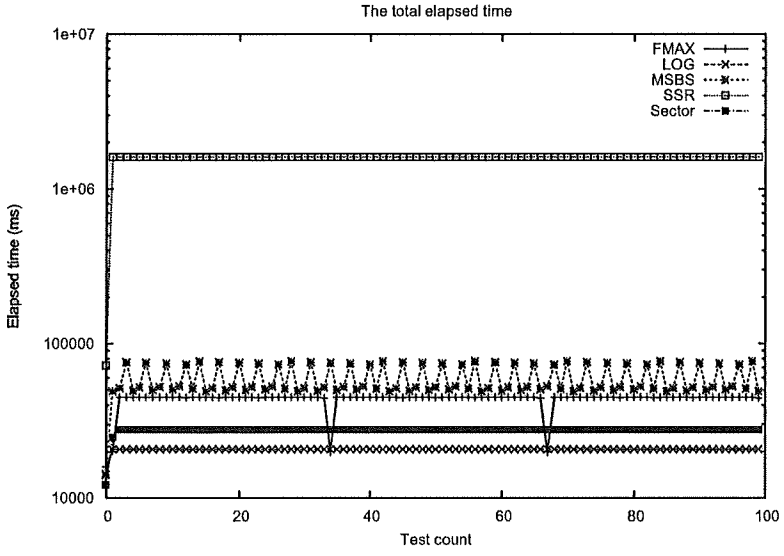


Fig. 14. Digicam: The total elapsed time.

change in the mapping table to flash memory whenever a logical sector is overwritten, which does not occur as often in the Log scheme, which uses LBN-to-PBN mappings. In the experiment, there are 32 sectors in a block. Therefore, the LSN-to-PSN mapping table has to be updated approximately 30 times more often than with LBN-to-PBN mapping. Consequently, although the sector

IPR2025-00212/PHI EX. 1064 Page 9 of 12

mapping scheme incurs less erase operations (Fig. 15, it has a longer overall execution time than the Log scheme.

The SSR technique shows the poor performance compared to the other techniques because one logical block is mapped to only one physical block in the SSR technique. In particular, when erasing a block in the SSR technique, as many valid sectors exist in the erased block, many copy operations are necessary and the probability that the erased block will be erased again in the future is very high. By allowing a logical block to be mapped to more than one block (as in the Log scheme and FMAX), FTL algorithms have better performance. In the Mitsubishi technique, a logical block can be mapped to a physical block. However, space sectors play the role of another block. Additionally, in the Mitsubishi technique, there is a periodic drop in the total elapsed time because the same workload is performed repeatedly and many merge operations are periodically required. The Log scheme shows better performance than FMAX and Mitsubishi; as there are a few log blocks for random writes, the merging algorithm of the Log scheme is more efficient than the Mitsubishi and FMAX schemes.

Fig. 15 shows the erase count. The result is similar to the result of the total elapsed time because the erase count is the most dominant factor in the overall system performance. A recent study [5] found a running time ratio of read (1 page), write (1 page), and erase (1 block) is approximately 1:7:63. It is clear that the sector mapping requires the smallest erase counts. In the sector mapping scheme, a logical sector can be mapped to any free physical sector. Most of the blocks are either full of invalid blocks or full of valid blocks. Therefore merge operations between blocks with valid sectors are uncommon, causing fewer erase operations.

Fig. 16 and Fig. 17 show the performance result in the Symbian workload. In the Symbian workload, the sector mapping shows the best performance. This result comes from the fact that the workload of Symbian, in contrast to that of Digicam, has many random write operations. Thus, in the Log scheme, erase

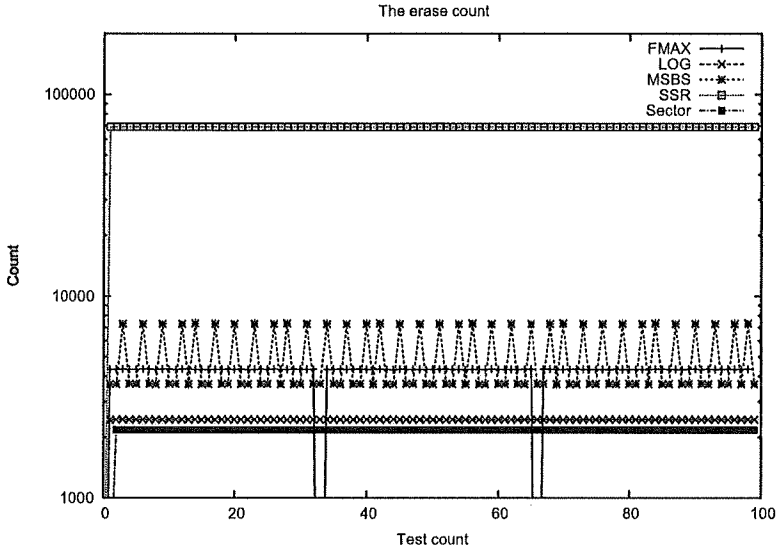


Fig. 15. Digicam: The total erase counts.

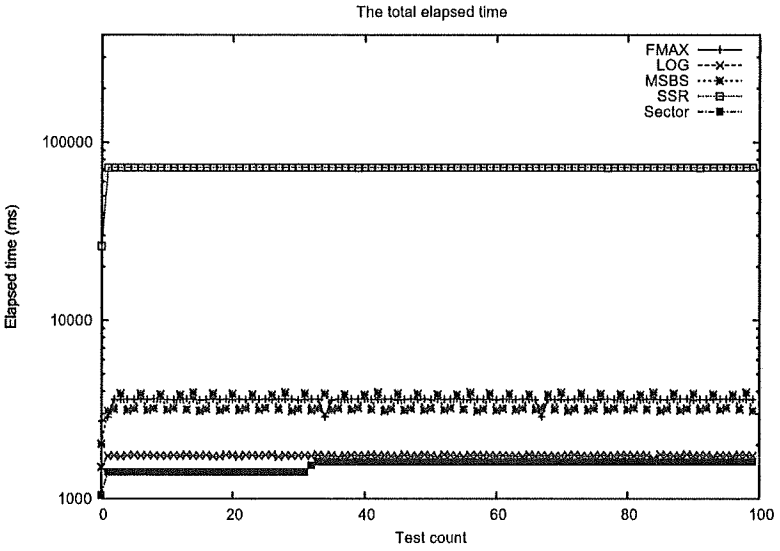


Fig. 16. Symbian: The total elapsed time.

IPR2025-00212/PHI EX. 1064 Page 10 of 12

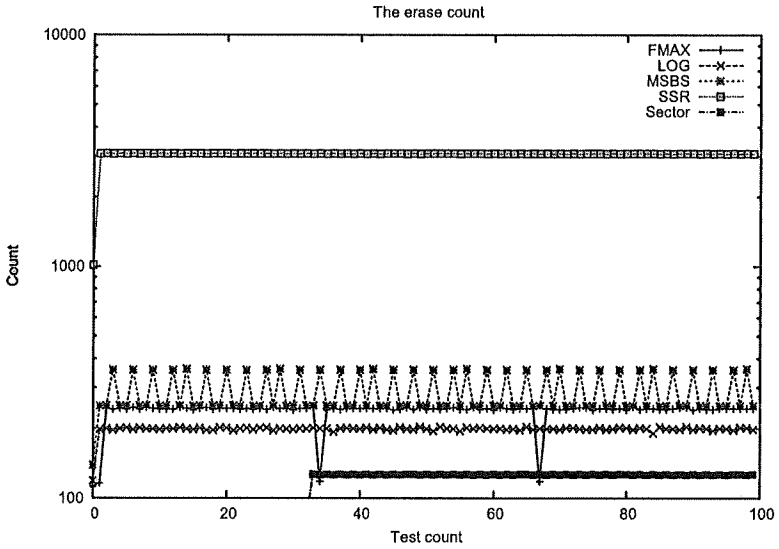


Fig. 17. Symbian: The total erase counts.

operations occur frequently compared to the sector mapping. Particularly, there is a lack of points in sector mapping, as shown in Fig. 17, due to the full-associativity of the sector mapping scheme.

6. Conclusion

This paper surveys state-of-the-art FTL algorithms. A taxonomy of FTL algorithms is provided based on sector, block, and hybrid address mapping. To overcome the "erase before write" architecture, the sector mapping scheme shows the best performance in that it can delay the erase operation as much as possible if there are free sectors in flash memory. However, because the sector mapping scheme requires a considerable amount of mapping information, it is not be applicable in embedded applications. Thus, the block mapping and the hybrid scheme are addressed. The block mapping scheme requires the least mapping information. However, it shows very poor performance when the same logical sector numbers are frequently updated. The hybrid mapping scheme overcomes the problems of 'updating the same logical sector numbers frequently' but requires more mapping information than the block mapping scheme.

Another issue when designing FTL algorithms involves managing mapping information. Current FTL techniques can be classified into the map block and the per block method. The map block method requires number of dedicated blocks to store mapping information, whereas the per block method stores mapping information in each block of flash memory. When implementing FTL algorithms, the RAM usage is also important. Current FTL algorithms use RAM to the store information of logical-to-physical mapping, of free memory space, and of wear-leveling.

Various FTL algorithms, in this case FMAX, sector mapping, Log scheme, SSR, and Mitsubishi were implemented, and the performance results were shown. If one logical block is mapped to only one physical block (as in SSR), it is clear that the FTL performance is poor. Using space sectors (Mitsubishi) and replacement blocks (FMAX), the FTL performance can be improved. If a logical block can be mapped to more than one block in FTL algorithms, the merging operation becomes a key factor in the overall performance. The Log scheme shows a good solution with its use small log blocks and many data blocks.

In a future study, intensive workloads in real embedded applications will be generated and the theoretical performance optimum for flash memory will be explored under a given workload.

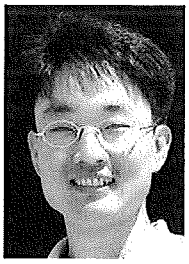
Acknowledgements

This work was supported in part by the Defense Acquisition Program Administration and Agency for Defense Development under Contract Number UD060048AD, was partly supported by MKE, Korea under ITRC IITA-2009-(C1090-0902-0046) and also supported partly by KRF, Korea under KRF-2008-0641.

References

[1] Amir Ban, Flash File System, United States Patent, No. 5,404,485, 1995.
[2] Amir Ban, Flash File System Optimized for Page-mode Flash Technologies, United States Patent, No. 5,937,425, 1999.
[3] Tae-Sun Chung, Hyung-Seok Park, STAFF: a flash driver algorithm minimizing block erasures, Journal of Systems Architecture 53 (12) (2007).
[4] Microsoft Corporation, Fat32 File System Specification, Technical Report, Microsoft Corporation, 2000.
[5] Samsung Electronics, Nand Flash Memory & Smartmedia Data Book, 2007.
[6] Petro Estakhri, Berhanu Iman, Moving sequential sectors within a block of information in a flash memory mass storage architecture, United States Patent, No. 5,930,815, 1999.
[7] Eran Gal, Sivan Toledo, Algorithms and data structures for flash memories, ACM Computing Surveys 37 (2) (2005) 123.
[8] A. Kawaguchi, S. Nishioka, H. Motoda, Flash Memory based File System, in: USENIX 1995 Winter Technical Conference, 1995.
[9] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, A space-efficient flash translation layer for compactflash systems, IEEE Transactions on Consumer Electronics 48 (2) (2002).
[10] Se Jin Kwon, Tae-Sun Chung, An efficient and advanced space-management technique for flash memory using reallocation blocks, IEEE Transaction on Transactions on Consumer Electronics 54 (2) (2008).
[11] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, Ha-Joo Song, Log buffer based flash translation layer using fully associative sector translation, ACM Transactions on Embedded Computing Systems 6 (3) (2007).
[12] M. Resenblum, J. Ousterhout, The design and implementation of a log-structured file system, ACM Transactions on Computer Systems 10 (1) (1992).
[13] Takayuki Shinohara, Flash Memory Card with Block Memory Address Arrangement, United States Patent, No. 5,905,993, 1999.
[14] Bum Soo Kim, Gui Young Lee. Method of Driving Remapping in Flash Memory and Flash Memory Architecture Suitable Therefore, United States Patent, No. 6,381,176, 2002.
[15] Symbian, 2007, <http://www.symbian.com>.
[16] M. Wu, W. Zwaenepoel, eNVy: a non-volatile, main memory storage system, in: International Conference on Architectural Support for Programming Languages and Operating Systems, 1994.

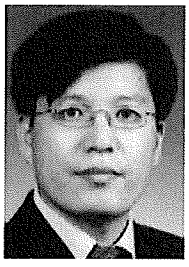
IPR2025-00212/PHI EX. 1064 Page 11 of 12



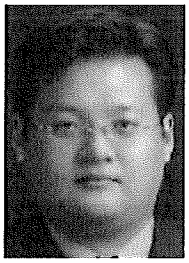
Tae-Sun Chung received the B.S. degree in Computer Science from KAIST, in February 1995, and the M.S. and Ph.D. degree in Computer Science from Seoul National University, in February 1997 and August 2002, respectively. He is currently an associate professor at School of Information and Computer Engineering at Ajou University. His current research interests include flash memory storages, XML databases, and database systems.



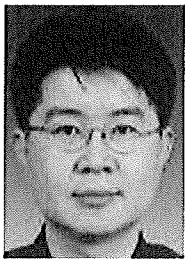
Dong-Joo Park received the B.S. and M.S. degrees in the Computer Engineering Department from Seoul National University, February 1995 and February 1997, respectively, and the Ph.D. degree in School of CS&E from Seoul National University, August 2001. He is currently an assistant professor in School of Computing at Soongsil University. His research interests include flash memory-based DBMSs, multimedia databases, and database systems.



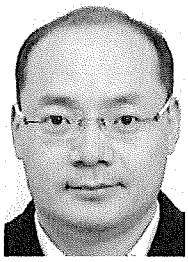
Sangwon Park received the B.S. and M.S. degrees in the Computer Engineering Department from Seoul National University, February 1995 and February 1997, respectively, and the Ph.D. degree in School of CS&E from Seoul National University, February 2002. He is currently an associate professor in Hankuk University of Foreign Studies. His research interests include flash memory-based DBMSs, multimedia databases, and database systems.



Dong-Ho Lee received the BS degree from Hong-Ik University, and the MS and PhD degrees in computer engineering from Seoul National University, South Korea, in 1995, 1997, and 2001, respectively. From 2001 until 2004, he worked in software center, SAMSUNG Electronics Ltd., where he was involved in several digital TV projects. He is currently an assistant professor in Department of Computer Science and Engineering at Hanyang University, South Korea. His research interests include system software for flash memory, embedded database systems, and multimedia information retrieval systems.



Sang-Won Lee is an associate professor with the School of Information and Communication Engineering at Sungkyunkwan University, Suwon, South Korea. Before that, he was a research professor at Ewha Womans University and a technical staff at Oracle, Korea. He received a Ph.D. degree from the Computer Science Department of Seoul National University in 1999. His research interest is in flash-based database technology. He can be reached at swlee@skku.edu.



Ha-Joo Song received the B.S. and M.S. degrees in the Computer Engineering Department from Seoul National University, February 1993and February 1995, respectively, and the Ph.D. degree in School of CS&E from Seoul National University, August 2000. He is currently an assistant professor in Pukyong National University. His research interests include flash memory based database systems and sensor networks.

IPR2025-00212/PHI EX. 1064 Page 12 of 12