LSTAFF: System Software for Large Block Flash Memory

Tae-Sun Chung[1], Dong-Joo Park[2], Yeonseung Ryu[1], and Sugwon Hong[1]

[1] Department of Computer Software, MyoungJi University, Kyunggido 449-728, Korea
{tschung,ysryu,swhong}@mju.ac.kr
[2] College of Information Science, School of Computing, Soongsil University, Seoul 156-743, Korea
djpark@computing.ssu.ac.kr

Abstract. Recently, flash memory is widely used in embedded applications since it has strong points: non-volatility, fast access speed, shock resistance, and low power consumption. However, due to its hardware characteristics, it requires a software layer called FTL (flash translation layer). We present a new FTL algorithm called LSTAFF (Large STAFF). LSTAFF is designed for large block flash memory. That is, LSTAFF is adjusted to flash memory with pages which are larger than operating system data sector sizes. We provide performance results based on our implementation of LSTAFF and previous FTL algorithms using a flash simulator.

1 Introduction

Flash memory has strong points: non-volatility, fast access speed, shock resistance, and low power consumption. Therefore, it is widely used in embedded applications, mobile devices, and so on. However, due to its hardware characteristics, it requires specific software operations in using it.

The basic hardware characteristics of flash memory is erase-before-write architectures [5]. That is, in order to update data on flash memory, if the physical location on flash memory was previously written, it has to be erased before the new data can be rewritten.

Thus, the system software called FTL (Flash Translation Layer) [1–3, 6, 8–10] is proposed. The proposed FTL algorithms are designed for small block flash memory in which the size of a physical page of flash memory is same to the size of data sector of operating system. Recently, however, major flash vendors have produced large block flash memory. In large block flash memory, the basic chunk for reading and writing is larger than operating system data sector sizes. That is, large block flash devices physically read and write data in chunks of 2K or 1K bytes. On the other hand, the data sector size of the most operating systems is 512 bytes.

Thus, the new FTL algorithm for large block flash memory should be addressed. We propose a high-speed FTL algorithm called LSTAFF (Large STAFF)

for the large block flash memory system. LSTAFF is adapted to large block flash memory using STAFF [3] which is an FTL algorithm for small block flash memory.

This paper is organized as follows. Problem definition and previous work is described in Section 2. Section 3 shows our FTL algorithm and Section 4 presents performance results. Finally, Section 5 concludes.

2 Problem Definition & Previous Work

2.1 Problem Definition

In this paper, we define operations units in the large block flash memory system as follows.

Definition 1. *Sector is the smallest amount of data which is read or written by the file system.*

Definition 2. *Page is the smallest amount of data which is read or written by flash devices.*

Definition 3. *Block is the unit of the erase operation on flash memory. The size of block is some multiple of the size of page.*

Figure 1 shows the software architecture of a flash file system. We will consider the FTL layer in Figure 1. The File System layer issues a series of read and write commands with logical sector number to read from, and write data to, specific addresses of flash memory. The given logical sector number is converted to a real physical sector offset within a physical page of flash memory by some mapping algorithm provided by FTL layer.

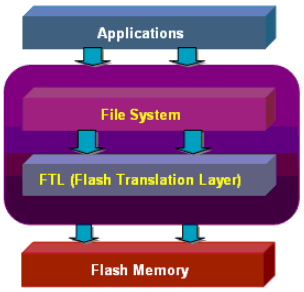


Fig. 1. Software architecture of flash memory system

Thus, the problem definition of FTL in the large block flash memory system is as follows. We assume that flash memory is composed of l physical pages which is composed of m physical sectors and the file system regards flash memory as n logical sectors. The number n is less than or equal to $l * m$.

Definition 4. *Flash memory is composed of blocks, each block is composed of pages, and each page is composed of sectors. Flash memory has the following characteristics: If the physical sector location within a physical page on flash memory was previously written, it has to be erased in the unit of block before the new data can be rewritten in the same location. The FTL algorithm in large block flash memory is to produce the physical sector offset within a physical page in flash memory from the logical sector number given by the file system.*

2.2 Previous FTL Algorithms

Since large block flash memory has recently been produced, there is no special work on FTL algorithm for large block flash memory. In small block case, we can classify the previous work into three categories: sector mapping [1], block mapping [2, 6, 9], and hybrid mapping [8, 10, 3].

Sector Mapping First intuitive algorithm is sector mapping [1]. In sector mapping, if there are m logical sectors seen by the file system, the raw size of logical to physical mapping table is m.

If the physical sector location on flash memory was previously written, the FTL algorithm finds another sector that was not previously written. If it finds it, the FTL algorithm writes the data to the sector location and changes the mapping table. If it can not find it, a block should be erased, the corresponding sectors should be backed up, and the mapping table should be changed.

Block Mapping Since the sector mapping algorithm requires the large size of mapping information, the block mapping FTL algorithm [2, 6, 9] is proposed. The basic idea is that the logical sector offset within the logical block corresponds to the physical sector offset within the physical block.

In block mapping method, if there are m logical blocks seen by the file system, the raw size of logical to physical mapping table is m.

Although the block mapping algorithm requires the small size of mapping information, if the file system issues write commands to the same sector frequently, the performance of the system is degraded since whole sectors in the block should be copied to another block.

Hybrid Mapping Since the both sector and block mapping have some disadvantages, the hybrid technique [8, 10, 3] is proposed. The hybrid technique first uses the block mapping technique to find the physical block corresponding to the logical block, and then, the sector mapping techniques are used to find an available sector within the physical block.

When reading data from flash memory, FTL algorithm first finds the physical block number from the logical block number according to the mapping table, and then, by reading the logical sector numbers within the physical block, it can read the requested data.

3 LSTAFF (Large STAFF)

LSTAFF is our FTL algorithm for the large block flash memory system. LSTAFF applies STAFF (State Transition Applied Fast Flash Translation Layer) [3] in the large block flash memory system.

3.1 Review: STAFF

STAFF introduced the states of the block. A block in STAFF has the following states.

- F state: If a block is an F state block, the block is a free state. That is, the block is erased and is not written.
- O state: If a block is an O state block, the block is an old state. The old state means that the value of the block is not valid any more.
- M state: The M state block is a first state from a free block and is in place. That is, the logical sector offset within the logical block is identical to the physical sector offset within the physical block.
- S state: The S state block is created by the swap merging operation. The swap merging operation is occurred when a write operation is requested to the M state block which has no more space.
- N state: The N state block is converted from an M state block and is out of place.

We have constructed a state machine according the states defined above and various events occurred during FTL operations. The state machine is formally defined as follows. Here, we use the notation about automata in [7]. An automaton is denoted by a five-tuples $(Q, \sum, \delta, q_o, F)$, and the meanings of each tuple are as follows.

- Q is a finite set of states, namely $Q = \{F, O, M, S, N\}$.
- $\sum$ is a finite input alphabet, in our definition, it corresponds to the set of various events during FTL operations.
- δ is the transition function which maps $Q \times \sum$ to Q.
- q_0 is the start state, that is a free state.
- F is the set of all final states.

Figure 2 shows the block state machine. The initial block state is F state. When an F state block gets the first write request, the F state block is converted to the M state block. The M state block can be converted to two states of S and N according to specific events during FTL operations. The S and N state block is converted to O state block in the event e4 and e5, and the O state block is converted to the F state block in the event e6. The detailed description of the events is presented in [3].

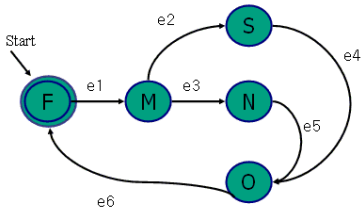


Fig. 2. Block state machine

3.2 Large Block Extension

Since a page is composed of more than one sector in large block flash memory, we should consider more complicated logical to physical mapping scheme. That is, we can classify the mapping technique in small block case into two kinds: sector and block mapping. In large block case, however, we should consider sector, page, and block level mapping.

To solve the FTL problem presented in Section 2.1, we first get logical page number, logical block number, and logical sector offset from the input logical sector number. If the input logical sector (lsn) is given, the logical block number (lbn), logical page number (lpn), and logical sector offset (lso) is calculated as follows. Here, np is the number of pages in a block and ns is the number of sectors in a page within a block.

$$lbn = lsn/np \tag{1}$$
$$tmp = lsn\%np \tag{2}$$
$$lpn = tmp/ns \tag{3}$$
$$lso = tmp\%ns \tag{4}$$

Example 1. If a flash memory is composed of 1024 blocks, each block composed of 64 pages, and each page is composed of 4 sectors, the capacity of the flash memory is 128MB. If the logical sector number is 101, the lbn is 1, lpn is 9, and lso is 1.

To store mapping information on flash memory, we designed the page format of LSTAFF as in Figure 3.

In Figure 3, we assume that a page in a block is composed of four sectors. As mentioned earlier, three levels of logical to physical mapping exits in the large block flash system: sector, page, and block mapping. Thus, logical sector numbers (lsn1, lsn2, lsn3, and lsn4), logical page number (lpn), and logical block number is stored in spare area which exits in each page and is usually used for storing meta information. Here, the lsns and lpn should be stored in each page and the logical block number may be stored in a page per block. From the mapping information, the logical to physical mapping table may be constructed by scanning the spare are of flash memory. Finally, the state of the block may be stored per block.

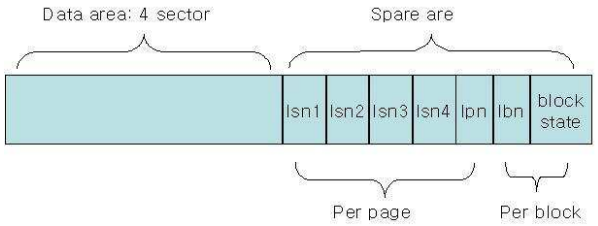


Fig. 3. Page format

In LSTAFF, the F, S, O, and M state blocks do not require lsns and lpn since the logical page (sector) offsets are identical to the physical page (sector) offsets in those state blocks. The N state block only require lsns or lpn. There are some options in writing to the N state block as follows.

- Page mapping: By using lpn area in Figure 3, we can map a logical page number to a physical page number. In this case, the logical sector numbers (lsn1, lsn2, lsn3, and lsn4) need not be written to flash memory.
- Sector mapping: By using lsn area in Figure 3, we can map a logical sector number to a physical sector number. In this case, the logical page number (lpn) need not be written to flash memory.
- 1:n mapping: In both page and sector mapping above, a logical page (sector) may be mapped to more than one physical page (sector).

Example 2. Figure 4 shows a write example of page and sector mapping. For the input lsn sequence (0,1,2,3,0,1,2,3,8,9,9), the upper side of Figure 4 shows the page mapping and the lower side of Figure 4 shows sector mapping. If we assume 1:1 page mapping, the data D8 in the upper side of Figure 4 should be copied to the location of physical page number 3. In reading from flash memory, most recently written data is valid data. For instance, in upper side of Figure 4, the valid data of lsn 9 is the one in the physical page number 3.

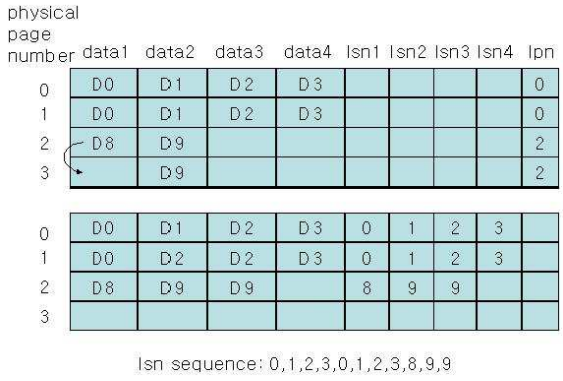


Fig. 4. Map example

4 Experimental Evaluation

4.1 Cost Estimation

The cost function for LSTAFF is similar to that of STAFF. However, since more than one sectors in a page can be read or written simultaneously in large block flash memory, the overall performance of LSTAFF is estimated to be better than that of STAFF. In LSTAFF, the read/write cost can be measured by the following equations:

$$C_{read} = p_M T_r + p_N k_1 T_r + p_S T_r \ (where\ p_M + p_N + p_S = 1) \tag{5}$$

$$\begin{aligned} C_{write} &= p_{first}[(T_f + T_w)] + (1 - p_{first})[p_{merge}\{T_m + \\ &\quad p_{e_1} T_w + (1 - p_{e_1})(k_2 T_r + T_w)\} + \\ &\quad (1 - p_{merge})\{p_{e_2}(T_r + T_w) + (1 - p_{e_2})(k_3 T_r + T_w) + \\ &\quad T_r + p_{MN} T_w\}] \end{aligned} \tag{6}$$

where $1 \leq k_1, k_2, k_3 \leq n$. Here, n is the number of pages within a block. In the equation (5), p_M, p_N, and p_S are the probability that data is stored in the M, N, and S state block, respectively.

In the equation (6), p_{first} is the probability that the write command is the first write operation with the input logical block and p_{merge} is the probability that the write command requires the merging operation. p_{e_1} and p_{e_2} are the probability that input logical sector can be written to the in place location with merging and without merging operation, respectively. T_f is the cost for allocating a free block. It may require the merging and the erasing operation. T_m is the cost for the merging operation. Finally, p_{MN} is the probability that the write operation converts the M state block to the N state block. When the write operation converts the M state block to the N state block, a flash write operation is needed for marking states.

The cost function shows that the read and write operations to the N state block requires some more flash read operation than the M or S state block. However, in flash memory the read cost is very low compared to the write and erase cost. Thus, since T_f and T_m may require the flash erase operation, they are dominant factors in evaluating the overall system performance. LSTAFF is designed to minimize T_f and T_m that require the erase operation.

4.2 Experimental Result

Simulation Methodology In the overall flash system architecture presented in Figure 1, we have implemented LSTAFF algorithm and basic block mapping algorithm presented in Section 2.2. In the block mapping algorithm, a logical block can be mapped to two physical blocks, which is more efficient algorithm. The physical flash memory layer is simulated by a flash emulator which has same characteristics as real flash memory.

We assume that the file system layer in Figure 1 is the FAT file system [4] which is widely used in embedded systems. Figure 5 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. Please refer [4] for more detailed description of the FAT file system. Here, we can see that logical spaces corresponding to the boot sector, file allocation tables, and the root directory are accessed more frequently than the volume files. We got access patterns that the FAT file system on Symbian operating system [11] issues to the block device driver when it gets a file write request. The access patterns are very similar to the real workload in embedded applications.

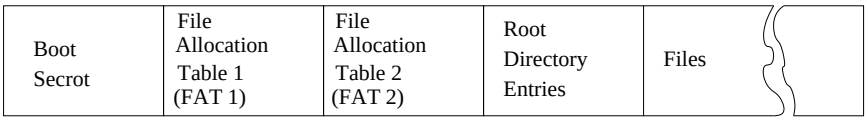
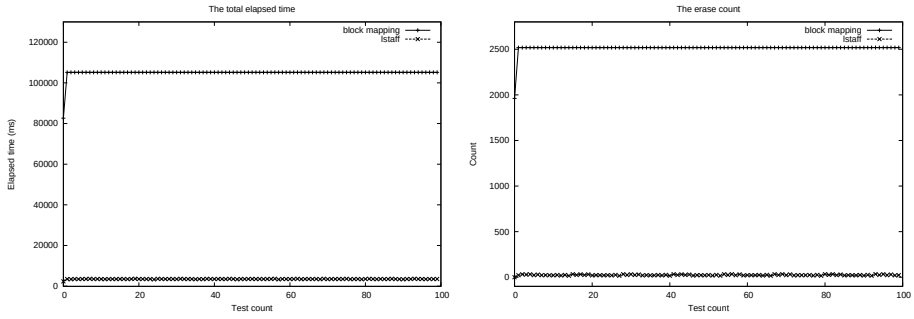


Fig. 5. FAT file system

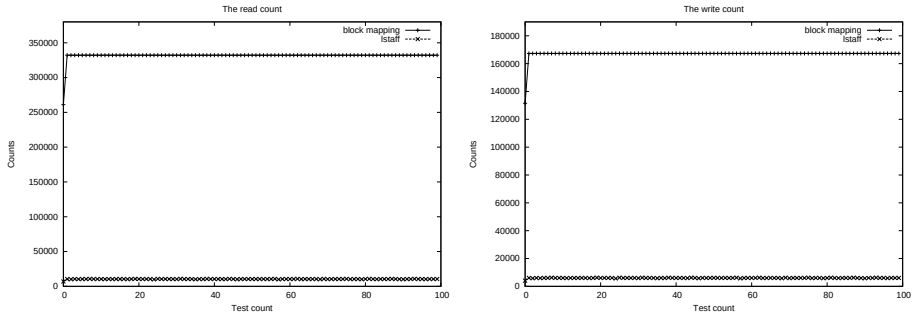
Result Figure 6-(a) shows the total elapsed time. The x axis is the test count and the y axis is the total elapsed time in millisecond. At first, flash memory is empty, and flash memory is occupied as the iteration count increases. The result shows that LSTAFF has much better performance than block mapping.



(a) The total elapsed time (b) Erase counts

Fig. 6. The total elapsed time and erase counts

Figure 6-(b) shows the erase count. The result is similar to the result of the total elapsed time. This is because the erase count is a dominant factor in the overall system performance. [5] says that the running time ratio of read (1 page),



(a) Read counts (b) Write counts

Fig. 7. The read and write counts

write (1 page), and erase (1 block) is 1:4:20 approximately. In addition, LSTAFF shows the consistent performance although flash memory is fully occupied.

Figure 7-(a) and Figure 7-(b) shows the read and write counts respectively. We can see that LSTAFF requires much smaller read and write operations. This is because that the merging operation occurs frequently in the block mapping method.

5 Conclusion

In this paper, we propose a novel FTL algorithm called LSTAFF for large block flash memory. LSTAFF is designed to provide maximum performance in large block flash memory whose page size is larger than file system's data sector size. In LSTAFF which have same characteristics as STAFF, the state of the erase block is converted to the appropriate states according to the input patterns, which minimizes the erase operation. Additionally, we have provided some heuristics for storing to large block flash memory.

Compared to the previous work, our cost function and experimental results show that LSTAFF has better performance.

References

1. Amir Ban. Flash file system, 1995. United States Patent, no. 5,404,485.
2. Amir Ban. Flash file system optimized for page-mode flash technologies, 1999. United States Patent, no. 5,937,425.
3. Tae-Sun Chung, Stein Park, Myung-Jun Jung, and Bumsoo Kim. STAFF: State Transition Applied Fast Flash Translation Layer. In *17th International Conference on Architecture of Computing Systems with Lecture Notes in Computer Science (LNCS) Springer-Verlag*, 2004.

4. Microsoft Corporation. Fat32 file system specification. Technical report, Microsoft Corporation, 2000.
5. Samsung Electronics. Nand flash memory & smartmedia data book, 2004.
6. Petro Estakhri and Berhanu Iman. Moving sequential sectors within a block of information in a flash memory mass storage architecture, 1999. United States Patent, no. 5,930,815.
7. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Company, 1979.
8. Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics*, 48(2), 2002.
9. Takayuki Shinohara. Flash memory card with block memory address arrangement, 1999. United States Patent, no. 5,905,993.
10. Bum soo Kim and Gui young Lee. Method of driving remapping in flash memory and flash memory architecture suitable therefore, 2002. United States Patent, no. 6,381,176.
11. Symbian. http://www.symbian.com, 2003.