IEICE Electronics Express, Vol.7, No.11, 804–809

Workload-driven adaptive log buffer-based FTL

Dongkun Shin[a)]
School of Information and Communication Engineering
Sungkyunkwan University, Suwon, Korea
a) *dongkun@skku.edu*

Abstract: Flash translation layer (FTL) is generally used for NAND flash memory in order to handle the mapping between logical page address and physical page address. Log buffer-based FTLs provide good performances with small-sized mapping information. In designing the log buffer-based FTL, one important factor is to determine the mapping architecture between data block and log block, called associativity. While previous static schemes use fixed associativities, our scheme adjusts the associativity dynamically based on the run-time workload variation improving the performance by 5∼16% compared to the static scheme.

Keywords: flash memory, flash translation layer, log buffer, hybrid mapping, embedded storage
Classification: Storage technology

References

[1] J. Kim, et al., "A spaceefficient flash translation layer for compact flash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, 2002.
[2] S.-W. Lee, et al., "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 3, Article no. 18, 2007.
[3] S. Y. Park, et al., "A re-configurable FTL (flash translation layer) architecture for NAND flash based applications," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 4, Article no. 38, 2008.
[4] Samsung Electronics, NAND Flash Data Sheet K9K8G08U1A, 2007.
[5] Z. Liu, et al., "An adaptive block-set based management for large-scale flash memory," *Proc. SAC'09*, pp. 1621–1625, 2009.

1 Introduction

NAND flash memory has become the most important storage media in the mobile embedded systems. Unlike a traditional hard disk, NAND flash memory does not support *overwrite* operation because of its "erase-before-write" characteristic. This feature of NAND flash memory requires two storage management schemes: address mapping and garbage collection. The address mapping scheme is to map a logical address from the file system to a physical address of the flash memory by maintaining the address mapping table.

DOI: 10.1587/elex.7.804
Received April 28, 2010
Accepted May 13, 2010
Published June 10, 2010

The garbage collection scheme makes it possible to reclaim the invalidated pages by erasing the corresponding block after copying valid pages in the block to a free block. To support these two management schemes, a software layer called flash translation layer (FTL) is used between file system and flash memory. The address mapping schemes of the FTL can be divided into three classes depending on the mapping granularity: page-level mapping, block-level mapping, and hybrid mapping. The drawback of the page-level mapping technique is that its mapping table size is inevitably large and thus it requires a large SRAM for mapping table. Though the block-level mapping requires a small-sized mapping table, it invokes a large overhead even if only a small portion of a block is changed.

Hybrid mapping is a compromise between page-level mapping and block-level mapping. In this scheme, a small portion of physical blocks is reserved as a log buffer. So, it is called a log buffer-based FTL. While the log blocks in the log buffer use the page-level mapping scheme, the normal data blocks are handled by the block-level mapping. A log block in the log buffer can be used for one or several data blocks, i.e., data blocks and log blocks are associated each other. The number of associated data blocks of a log block is called the associativity of the log block. When the write request for a data block is sent to the FTL, the data is written to the associated log block, and the corresponding old data in the data block is invalidated. Hybrid mapping requires a small-sized mapping table since only the log blocks are handled by the page-level mapping. In addition, unlike block-level mapping, hybrid mapping does not invoke a large overhead for every write request.

When there is no empty space in the log buffer, one of the log blocks is selected as a victim and all of the valid pages in the log block are moved into the data blocks to make space for on-going write requests. This process is referred to as a log block merge. The merge cost of a log block L is calculated as follows:

$$N \cdot A(L) \cdot C_{copy} + (A(L) + 1) \cdot C_{erase} \quad (1)$$

where N, C_{copy}, and C_{erase} denote the number of pages in a flash block, the cost of one page copy, and the cost of one block erase, respectively. $A(L)$ is the associativity of the log block L, which means the number of data blocks that share the log block.

As the associativity of log block is large, the average cost of log block merge is large but the log block merge is invoked infrequently. Therefore, it is important to find the optimal associativity in order to reduce the log block merge overhead. Several recently proposed FTLs use static schemes which select fixed associativities at the design time. In this paper, we propose an adaptive scheme, called A-SAST, which adjusts the associativity between data blocks and log blocks depending on the run-time workload to minimize the log block merge overhead. Experimental results show that our scheme increases the performance by 5∼16% over the previous scheme that uses the static best associativity determined by examining the target workload pattern at the design time.

DOI: 10.1587/elex.7.804
Received April 28, 2010
Accepted May 13, 2010
Published June 10, 2010

2 Related works

There are several kinds of log buffer-based FTL schemes such as BAST [1], FAST [2] and SAST [3]. The SAST scheme groups N number of sequential data blocks into a data block group (DBG). One DBG can be associated with only one log block group (LBG) that has K number of log blocks at maximum. A log block of an LBG can have the updated pages of any data block of the associated DBG. Therefore, $N+K$ number of physical blocks can be allocated for N number of logical blocks, thus SAST is referred to N:$N+K$ mapping. If both N and K are 1, the SAST scheme is same to the BAST scheme. Another extreme case of SAST can be made by grouping all data blocks into one group, which is equal to the FAST scheme.

Fig. 1 illustrates the SAST scheme. We assume that one flash memory block consists of four pages. One data block group consists of four sequential data blocks and one log block group consists of two log blocks at most, i.e., N=4 and K=2. The log block groups LBG_0, LBG_1 and LBG_2 are associated with the data block groups DBG_0, DBG_1 and DBG_2, respectively. For example, the physical blocks with physical block numbers (PBNs) 100, 101, 102, and 103 are allocated as data blocks for DBG_0, and the physical blocks with PBNs 301 and 302 compose LBG_0. When the update requests on the logical page numbers 2, 3, 4, 5, 6, and 7 are sent to FTL, the new data are written at the log blocks in LBG_0 and the old pages in DBG_0 are invalidated. When there is no free space in the corresponding LBG of the target DBG, SAST selects the least-recently-used log block among all log blocks and merges it with data blocks to make a clean log block.

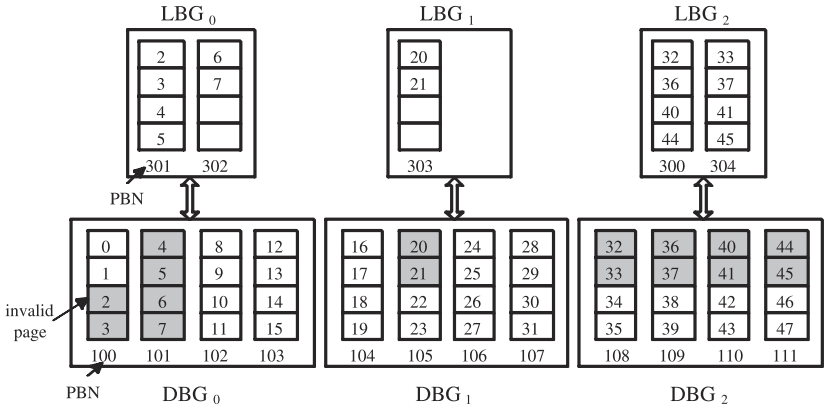

Fig. 1. SAST scheme (N:4, K:2).

In SAST, the values of N and K have a significant influence on the FTL performance, and the optimal values for N and K depend on the I/O request pattern. We can find the optimal values by exhaustive simulation for target I/O workloads. A more efficient approach is to use the workload analysis technique proposed in [3]. To find the proper value for N, the workload analysis technique examines the request density of a logical block, which is the ratio of the number of requests accessed in the logical block to the total number of requests. If the request density is high, there is a high spatial locality for the logical block. So, a large value should be used for N. The proper value for K is determined by measuring the temporal locality. If there

DOI: 10.1587/elex.7.804
Received April 28, 2010
Accepted May 13, 2010
Published June 10, 2010

are many updates, a large value should be used for K.

However, the optimal values of N and K are changed during the run time and they are different depending on the logical address since the I/O pattern varies according to the execution time and the logical address. However, the values of N and K are equally applied throughout all the address space and the values are fixed during the run time in the SAST scheme. Moreover, it is difficult to know the exact run-time workload for analysis at design time. Therefore, it is necessary to develop an adaptive scheme which can change the sizes of DBG and LBG depending on the run-time I/O workload.

3 Adaptive SAST scheme

The optimal size of a DBG is related to the log block utilization and the log block merge cost. Generally, as we increase the size of data block group, the log block utilization increases because several data blocks can share a log block. However, the log block merge cost also increases because the associativity of a log block increases. Therefore, if an LBG has a low utilization, which means that the associated DBG cannot utilize log blocks effectively, it is better to use a larger value for the size of corresponding DBG since it has a low request density. If an LBG has too high associativity thus it has a high average merge cost, a smaller value is proper to the size of corresponding DBG since it has a random request pattern.

The proposed adaptive SAST scheme, called A-SAST, adjusts the size of each DBG to adapt to the I/O pattern of the corresponding address range and the change of I/O pattern at run time. The adjustment is performed by merging or splitting data groups. For example, in Fig. 1, LBG_0 and LBG_1 consume small numbers of pages thus have low utilizations. This is because there were little updates for DBG_0 and DBG_1. In this case, if we create a new larger data block group by merging DBG_0 and DBG_1 and assign one log block group to the merged DBG, the log block utilization will increase. On the other hand, LBG_2 has a high utilization but the merge cost for each log block is high because each log block is associated with several data blocks. This means that the write pattern for DBG_2 is quite random. If we split DBG_2 into two smaller data groups, the merge cost for each log block can be reduced.

Fig. 2 shows how A-SAST changes the block groups by merging and splitting for the example in Fig. 1. By merging DBG_0 and DBG_1, a new DBG which consists of 8 data blocks is created and it requires only two log blocks. As a result, it increases the log block utilization and saves one log block. Merging data block groups is performed when the following conditions are satisfied for two consecutive DBGs, DBG_i and DBG_j:

$$\frac{P_{used}(\phi(DBG_i))}{P_{total}(DBG_i)} < \alpha, \frac{P_{used}(\phi(DBG_j))}{P_{total}(DBG_j)} < \alpha \quad \text{and} \tag{2}$$

$$\forall L \in \phi(DBG_i) \cup \phi(DBG_j), A(L) < \beta \tag{3}$$

where $\phi(g)$, P_{used}, and P_{total} are the associated log block group of data block group g, the number of used pages and the number of allocated pages for

DOI: 10.1587/elex.7.804
Received April 28, 2010
Accepted May 13, 2010
Published June 10, 2010

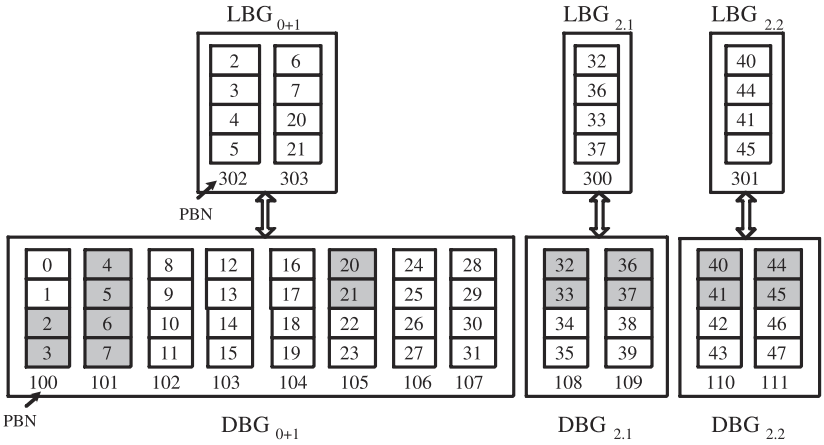

Fig. 2. Reorganizing data block groups in A-SAST.

DBG or LBG, respectively. The first condition checks the utilizations of two LBGs and the second condition checks the associativities of all log blocks of DBG_i and DBG_j. When a log block receives random write requests, the associativity of the log bock increases and thus the merge cost is high. Therefore, the second condition prevents DBGs from being merged when each DBG requires a high merge cost. For example, the log block group utilizations of DBG_0 and DBG_1 are 6/16 and 2/16, respectively, and the associativities of their log blocks are not larger than 2 in Fig. 1.

The group merge condition is examined for the DBG whose one of associated log blocks is selected as a victim log block. By merging data block groups which have small number of updates and sequential write patterns, we can use the log blocks more efficiently.

If the log block association is high due to many random updates for a certain group, the performance can be enhanced by splitting the DBG. Since the associativities of the log blocks in LBG_2 are large in Fig. 1, DBG_2 is split into two data block groups in Fig. 2. If write requests for each page occur as the order of (32, 36, 40, 44, 33, 37, 41, 45), the associativity of each log block is reduced by half compared to Fig. 1. A-SAST checks the following DBG split condition when a new log block is allocated for the DBG:

$$A(L) > \gamma \tag{4}$$

where γ is the split threshold and L is the lastly written log block of the log block group.

The proper values for α, β, and γ are derived from the experiments on the target workloads. The group merge and split conditions actually compare the randomness of workload, which is represented by the merge cost of log blocks, with the threshold values. Therefore, the best ranges of α, β and γ are irrelevant to the workloads.

We eliminated the constraint for the value of K. Thus an LBG can have any number of log blocks ($K = \infty$) though the total number of log blocks is fixed. The LRU policy for log buffer automatically control the proper number of log blocks. That is, if a DBG needs a large number of log blocks due to

DOI: 10.1587/elex.7.804
Received April 28, 2010
Accepted May 13, 2010
Published June 10, 2010

its high temporal locality, the victim selection policy takes a log block from the DBG with a low temporal locality and gives the log block to the DBG with a high temporal locality.

4 Results

Our proposed scheme is evaluated using simulation. Four I/O workloads are used: PCtrace, RandomFile, Iozone-4, and Iozone-80. PCtrace workload is collected by executing several Windows applications. RandomFile is composed of random I/O requests for multiple files. Iozone-4 and Iozone-80 are collected using Iozone benchmark program. While Iozone-4 generates random I/O pattern, Iozone-80 generates both random and sequential I/O requests. The timing parameters for NAND flash operations are based on [4].

We compared our proposed A-SAST scheme with SAST and BSFTL [5]. For SAST, we used the fixed optimal sizes of DBG and LBG for each target workloads while A-SAST adjusts the values dynamically. BSFTL is an adaptive scheme which adjusts the data group size at run time. BSFTL focuses on only the hotness of data blocks and thus uses 1:1 mapping for hot blocks and uses 1:N mapping for cold blocks to reduce the log block merge cost. Fig. 3 shows the total I/O execution times of BSFTL, SAST, and A-SAST. A-SAST shows the performance improvements by 30∼43% and 5∼16% compared to BSFTL and A-SAST, respectively.

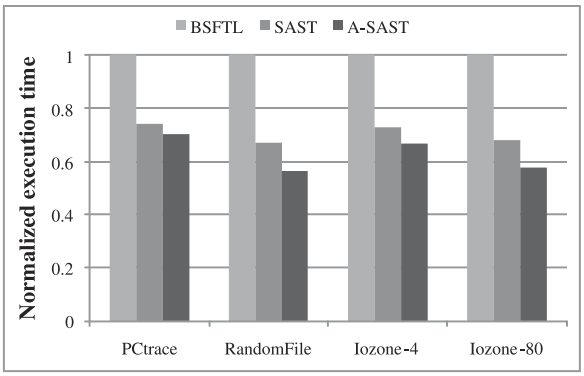

Fig. 3. Performance Comparison.

5 Conclusion

We proposed a novel FTL scheme to deal with proficiently irregular I/O patterns in NAND flash memory. While the previous FTL schemes use fixed log block associativities determined at design time, the proposed A-SAST scheme adjusts dynamically it during run time considering the variations of I/O workload. As a result, A-SAST can reduce the log block merge overhead of FTL significantly.

Acknowledgments

This paper was supported by Faculty Research Fund, Sungkyunkwan University, 2007.

DOI: 10.1587/elex.7.804
Received April 28, 2010
Accepted May 13, 2010
Published June 10, 2010