International Journal of Engineering Inventions
ISSN: 2278-7461, www.ijeijournal.com
Volume 1, Issue 4 (September2012) PP: 77-79

An Optimal FTL for SSDs

Ilhoon Shin
Department of Electronic Engineering, Seoul National University of Science & Technology, South Korea

Abstract— SSDs (Solid State Drives) are replacing the traditional HDDs (Hard Disk Drives) in personal and enterprise computing environments. SSDs use NAND flash memory as storage medium and require to deploy FTL (Flash Translation Layer) to emulate the overwrite feature, because NAND flash memory does not support it. The efficiency of FTL is a prime factor to determine the overall performance of SSDs. This work analyses the strengths and the weaknesses of the representative FTL schemes, and presents the directions to design the optimal FTL for SSDs.

Keywords — sector mapping, flash translation layer, NAND flash memory, SSD

I. INTRODUCTION

SSDs (Solid State Drives) are replacing the traditional HDDs (Hard Disk Drives) in personal and enterprise computing environments, due to the strengths of light-weight, silence, low energy consumption, shock-resistance, and fast performance. The drawbacks are a relatively low write performance and a fluctuation in the write performance, which mainly stem from the property of NAND flash memory that is used as storage medium in SSDs.

NAND flash memory is a kind of EEPROM (Electrically Erasable Programmable Read Only Memory) that consists of blocks and pages. It basically provides three operations: read, write, and erase. The overwrite operation is not supported. Once a cell is written, it should be erased first to write new data, which is called erase before write feature. The worse thing is that an erase unit that is a block is larger than a read/write unit that is a page. In order to support the overwrite feature like the standard block devices, SSDs deploy a special firmware called FTL (Flash Translation Layer). FTL emulates the overwrite operation with an out-of-place update, which writes new data to another clean page and invalidates the previous page. In the out-of-place update, the location of valid data becomes different on every overwrite operation, and thus FTL maintains the mapping information between the logical address space and their physical addresses. For example, if the above file system requests to read the sector 50, the FTL refers to the mapping table, finds the current physical location of the sector 50, and finally reads the data of the sector 50. For a fast access, the mapping table generally resides in the internal RAM.

Meanwhile, the clean pages will be eventually lacking by the continuous overwrite requests, which triggers a garbage collector. The garbage collector reclaims the invalidated pages to clean pages by the erase operation. The garbage collection accompanies multiple block erasures and page copies, and thus it is a main source of the bad write performance and the write performance fluctuation. In order to improve the write performance and to reduce the fluctuation, lots of works have been conducted to design the effective FTL schemes. This work examines the strengths and the drawbacks of the representative FTL schemes, and presents the direction to design the optimal FTL scheme for SSDs.

II. FLASH TRANSLATION LAYER

The representative FTL schemes are categorized to page mapping [1], block mapping [2], and hybrid mapping [3-4], according to the mapping unit between the logical address space and the physical address space. The page mapping scheme [1] uses a NAND page as a mapping unit. Upon a write request, it searches for a clean page, and the data are written to the found page. If the amount of the data is less than the page size, the unmodified partial data are copied from the old page. After finishing writing the data, the previous page is invalidated, and the mapping table is updated to remember the new location.

If the clean pages are lacking by the continuous overwrite operations, the garbage collector is conducted. It selects a victim NAND block to reclaim. The valid pages of the victim block are copied to a reserved clean block, and the victim block is erased. The erased victim block becomes the reserved clean block, and the previous reserved block serves the subsequent overwrite requests. Thus, the latency of the garbage collector is proportional to the number of valid pages in the victim block. In addition, the frequency of the garbage collection is also influenced by the number of valid pages, because the number of the created clean pages by the garbage collector is anti-proportional to the number of valid pages. Thus, it is beneficial to select the most invalidated block as victim [5].

The page mapping scheme delivers a good write performance by fully utilizing the clean pages. The garbage collection is delayed as possible. The drawback is a large memory consumption to maintain the mapping table. It requires the same number of the mapping entries with the number of pages in NAND. Even though SSDs deploy a large sized internal RAM, the large mapping table can be a burden.

The block mapping scheme [2] uses a NAND block as a mapping unit. Upon a write request, it searches for a clean block, and the data are written to the found block. If the amount of the data is less than the block size, the unmodified partial data are copied from the old block, and the page order inside the block is always preserved. After finishing writing the data,

the previous block is reclaimed to a clean block with the erasure operation, and the mapping table is updated to remember the new location.

The block mapping scheme reduces the mapping table greatly, because the number of the mapping entries is the same with the number of blocks in NAND. Note that a block generally consists of 64 or 128 pages. The drawback is a bad write performance. Even upon a small sized write request, the whole block is copied to the new block. Due to its low write performance, the block mapping scheme is rarely used in SSDs.

In order to cope with the small sized write pattern with reasonable memory consumption, the BAST (Block Associative Sector Translation) scheme [3], which is a hybrid of the block mapping scheme and the page mapping scheme has been presented. The BAST scheme uses several NAND blocks as write buffer, which are called log blocks, in order to absorb the small sized write requests. The log blocks are managed with the page mapping scheme, which means that the pages in the log block can be written regardless of the logical order, and the other blocks that are called data blocks are managed with the block mapping scheme, which means that the pages in the data block should be written in the logical order. A log block is associated with one data block. Upon a write request, it searches for the associated log block with the target data block from the block mapping table of the data blocks, and writes the requested data to the log block sequentially, regardless of the logical order. If the data block does not have the associated log block, a clean log block is allocated. At this time, if there is no clean log block, the garbage collector is triggered. It selects a victim log block with the LRU (Least Recently Used) replacement scheme and merges the victim log block with the associated data block. During the merge process, the log block and the data block are erased after the valid pages in them are copied to a new clean data block.

The BAST scheme reduces the mapping table because the page mapping tables are required only for the log blocks. Also, it copes with the small sized write pattern by absorbing them with the log blocks. However, it is vulnerable against the widely distributed random write pattern [4]. In the random write pattern, the clean log blocks are easily exhausted because the log block cannot be shared by multiple data blocks. The garbage collector is frequently initiated, and the log blocks are erased even though they still have clean pages.

In order to address this problem, the FAST (Fully Associative Sector Translation) scheme [4] allows a log block shared by multiple data blocks. Upon a write request, the data are written to the current working log block, regardless of its target sector number. If there are no clean pages in the working log, the next log block becomes the current working log, and if the next log block was used as the working log before and does not have clean pages, the garbage collector is triggered. The garbage collector selects a victim log block with the FIFO (First In First Out) replacement scheme. The victim log is merged with the associated data blocks. The merging process is repeatedly conducted the same number of times as the associated data blocks, which makes the garbage collection latency significantly longer than in the BAST scheme.

The FAST scheme reduces the mapping table also, and it delivers a good average performance even in the random write pattern by fully utilizing the log space. However, as mentioned above, the latency of the garbage collector can be significantly long, which causes a serious fluctuation of the write performance.

III. REQUIREMENTS OF OPTIMAL FTL

Most of the previous works have mainly focused on improving the average performance of the FTL schemes and the worst-case performance or the performance fluctuation has not been sufficiently studied. However, in some computing environments, the worst-case performance may be as important as the average performance. Thus, the optimal FTL requires delivering both the good average performance and the worst-case performance. In order to achieve the goal, the optimal FTL should incur infrequent garbage collection, and the latency of the garbage collection should be short and evenly distributed. The computation overhead and the memory consumption should be low also, because the FTL scheme is executed by the internal controller using internal RAM inside SSDs. Even though SSDs tend to deploy a large sized RAM inside it, the available RAM space except the region to maintain the mapping tables can be used as the read/write buffer and contribute to increase the overall performance by absorbing some read/write requests without accessing NAND flash memory. Reducing the number of the write requests to NAND flash memory with the internal buffer leads to reducing the frequency of the garbage collection.

Table 1. Evaluation of the FTL schemes

	Computation overhead	Memory consumption	Frequency of garbage collection	Latency of garbage collection
Page mapping	insignificant	large	infrequent	skewed
Block mapping	insignificant	small	significantly frequent	evenly distributed
BAST	insignificant	small	frequent	evenly distributed
FAST	significant	small	infrequent	seriously skewed

Table 1 shows the evaluation of the representative FTL schemes in the four aspects described above. From the aspect of the frequency of the garbage collection which mainly affects the average performance, the block mapping scheme is not the optimal FTL for SSDs. Its significant page copying overhead on every write requests hurts the effectiveness despite of the other strengths. The BAST scheme is also not the optimal FTL due to the same reason.

The page mapping scheme delivers the good average performance by fully utilizing the clean pages. The garbage collector is infrequently triggered. Also, its computation overhead is low because the location of the valid data is easily found by accessing the page mapping table. The time complexity is O (1). However, it consumes the large memory, which restricts the benefits of the internal buffer. This drawback can be mitigated by loading only hot portion of the page mapping table and storing the cold region to NAND flash memory [6]. However, the tradeoff between the benefits of the buffer and the miss penalty of the page mapping table access has not been sufficiently studied. Also, in the page mapping scheme, the latency of the garbage collection fluctuates according to the number of the valid pages in the victim block. The garbage

collection latency is proportional to the number of the valid pages because the valid pages should be copied to another clean block. Thus, the greedy replacement scheme that selects the mostly invalidated block as victim is beneficial to improve the average performance. However, if the invalidated pages are evenly distributed among the blocks, even the greedy scheme is not helpful. The latency of write requests can fluctuate, and the average performance will be hurt, because of the long garbage collection latency. In this case, we need to move the valid pages of a block to another block during the idle time to create the mostly invalidated blocks. If there are the mostly invalidated blocks, the garbage collection latency becomes short and thereby the fluctuation of the write latency becomes flatter.

Therefore, in order to make the page mapping scheme optimum for SSDs, the original scheme should be revised so that only hot portion of the page mapping table is loaded to RAM and that the valid pages are moved to another clean blocks in background to create the mostly invalidated blocks. The optimal tradeoff point between the benefit of the buffer and the miss penalty of the mapping table access should be investigated.

The FAST scheme also delivers the good average performance by fully utilizing the log space. The garbage collector is infrequently triggered. Also, its memory consumption is low, which means that the available memory space can be used as the buffer and contribute to improve the average performance more. However, its computation overhead of finding the location of valid data can be significant because the valid data are distributed across the entire log blocks, which means that we need to scan all the page mapping tables of the entire log blocks in the worst-case. However, this drawback can be mitigated by using the hashed page table, which records the physical location of the valid data [7]. The hashed page table is accessed in O (1) time using the logical sector number. The tradeoff is the increased memory consumption, but it is not serious. Another main drawback of the FAST scheme is the large fluctuation of the garbage collection latency, which can hurt the worst-case write performance significantly. It stems from the excessive association of the victim log block. In order to reduce the worst-case write latency of the FAST scheme, the KAST (K-way set Associative Sector Translation) scheme [8] restricts the maximal association of each log block to K. The cost is increased memory consumption. The block mapping table should record the associated data block numbers additionally, and thus the memory consumption is proportional to K. Also, a small value of K reduces the worst-case latency, and however the average performance is hurt, because the utilization of the log blocks becomes lower than the FAST scheme. Therefore, in order to make the FAST scheme optimum for SSDs, the similar works to reduce the computation overhead and the worst-case latency like [7-8] should be conducted, while at the same time the average performance should not be hurt.

IV. CONCLUSION

This work analyzed the strengths and the weaknesses of the representative sector translation schemes, and presented the directions to design the optimal FTL for SSDs. The block mapping scheme and the BAST scheme could not be the optimal FTL because of their low average performance. The page mapping scheme could be the optimal FTL, if its large memory consumption is adequately reduced and the background page movement is performed without hurting the average performance. The FAST scheme also could be the optimal FTL because it delivers the good average performance However, the computation overhead and the worst-case latency should be reduced without hurting the average performance significantly.

ACKNOWLEDGEMENTS

This work was supported by Seoul National University of Science and Technology.

REFERENCES

1. Ban, A. Flash file system, United States Patent, 1995, No. 5,404,485.
2. Ban, A. Flash file system optimized for page-mode flash technologies, United States Patent, 1999, No. 5,937,425.
3. Kim, J.; Kim, J. M.; Noh, S.; Min, S.; Cho, Y. A space-efficient flash translation layer for compactflash systems, IEEE Transactions on Consumer Electronics, 2002, 48, pp. 366–375.
4. Lee, S.; Park, D.; Chung, T.; Choi, W.; Lee, D.; Park, S.; Song, H. A log buffer based flash translation layer using fully associative sector translation. ACM Transactions on Embedded Computing Systems, 2007, 6 (3).
5. Kawaguchi, A.; Nishioka, S.; Motoda, H. A flash-memory based file system. Proceedings of USENIX Technical Conference, 1995, pp. 155-164.
6. Gupta, A.; Kim, Y.; Urgaonkar, B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. Proceedings of ACM ASPLOS, 2009.
7. Shin, I. Reducing computational overhead of flash translation layer with hashed page tables. IEEE Transactions on Consumer Electronics, 2010, 56 (4), pp. 2344–2349.
8. Cho, H.; Shin, D.; Eom, Y. KAST: K-associative sector translation for NAND flash memory in real-time systems. Proceedings of DATE, 2009.