International Journal of Control and Automation
Vol.7, No.6 (2014), pp.169-176
http://dx.doi.org/10.14257/ijca.2014.7.6.17

Performance Evaluation of Flash Translation Layer Considering Utilization and Dynamic Over-provisioning

Ilhoon Shin
Seoul National University of Science & Technology
ilhoon.shin@snut.ac.kr

Abstract

Flash translation layer (FTL) is a firmware embedded in NAND-based block devices. It hides unique characteristics of NAND flash memory and emulates a standard block device interface. The overall performance of NAND-based block devices is mainly determined by the efficiency of the FTL schemes, and thus, it is important to evaluate the FTL performance to design high-speed NAND-based block devices, which is a main objective of this work. Whereas most previous works have not considered device utilization from a FTL perspective and have fixed the over-provisioning factor, this work evaluates their performance varying device utilization and allows unused space to be used as over-provision area. A trace-driven simulation shows that device utilization significantly influences on FTL's performance. Especially, the page mapping FTL is vulnerable against high utilization, even though it delivers a good performance at low utilization. In contrast, the fully associative sector translation (FAST) FTL is less sensitive to the utilization and delivers a competing performance. It even outperforms the page mapping FTL at high utilization.

Keywords: *dynamic over-provisioning, utilization, flash translation layer, NAND flash memory*

1. Introduction

NAND flash memory is light, silent, shock resistant, and power efficient compared to hard disk drives (HDDs). It is a kind of Electrically Erasable Programmable Read Only Memory (EEPROM). Unlike HDDs, head positioning overhead is not involved and thus it delivers good random read performance. As a result, block devices that use NAND flash memory as storage media such as memory cards have been widely used in mobile computing systems, and nowadays, they are substituting HDDs in laptop, PC, and even server or cloud in a form of solid state drives (SSDs).

An inferior feature of NAND flash memory is that an overwrite operation is not supported, which is called erase-before-write feature. In order to write data to a cell, the cell should be clean status, which means that it was erased before. Because a unit of an erase operation is larger than a write unit, implementing an in-place update is not possible. Therefore, NAND-based block devices embed a firmware called flash translation layer (FTL) to support the overwrite operation. FTL emulates a standard block device interface and feature by performing an out-of-place update, which writes data to a new location. In the out-of-place update, a location of valid data becomes different on every write request, and thus FTL maintains the mapping information between logical sector number that an above file system uses and its physical location.

The overall performance of NAND-based block devices is mainly determined by the efficiency of a FTL scheme and thus intensive works to improve the performance of the FTL schemes have been performed. In the most previous works, the influence of device utilization is not sufficiently investigated and the over-provisioning has been generally fixed. In this work, we evaluate the influence of device utilization on the performance of the FTL schemes. When doing performance evaluation, unused space is allowed to be used as over-provisioning space. Thus, over-provisioning area is dynamically changed depending on the utilization.

A trace-driven simulation shows the followings. First, when the utilization is low, the page mapping FTL delivers the best performance. Second, the page mapping FTL is however vulnerable against high utilization. Its performance significantly degrades as the device is highly utilized. In contrast, fully associative sector translation FTL delivers a relatively good performance at high utilization.

The rest of the paper is organized as follows. Section 2 explains the representative FTL schemes in detail. Section 3 describes why considering the device utilization is important and how to modify the existing FTL schemes for dynamic over-provisioning. A performance evaluation results are presented in Section 4. Finally, Section 5 states our conclusions.

2. Flash Translation Layer

NAND flash memory consists of blocks and pages. A page is a unit of a read/write operation and generally 2 KB or 4 KB in size. A block is a unit of an erase operation and generally 128 KB or 256 KB in size. Therefore, a block consists of 64 or 128 pages. As described in section 1, the direct over-write operation is not supported. Once a page is written, it cannot be re-written before the block that the page belongs to is erased. Thus, FTL performs the out-of-place update. The detailed mechanism of the out-of-place update is different according to the FTL schemes.

The FTL schemes are mainly categorized into page mapping [10], block mapping [11], and hybrid mapping schemes [1-4] according to the mapping granularity between the logical sector number and its physical location.

The page mapping scheme [10] uses a NAND page as mapping granularity. In other words, the logical page number that is calculated from the logical sector number is matched with the physical page number. The offset inside a page is not changed, which is a similar address translation mechanism with a paging system in virtual memory. Upon a write request, FTL searches for a clean page and writes data to the found clean page. At the time, the old page is marked as invalid because its data become obsolete, and the mapping table is updated accordingly. Upon a read request, the valid location is easily found from the mapping table by indexing it with the logical page number.

Meanwhile, continuous write requests will exhaust clean pages. At the time, a garbage collection is initiated to reclaim invalidated pages to clean pages. It first selects a victim block. Then, the valid pages of the victim block is copied to an extra block that has been preserved in a clean status. After copying the valid pages, the victim block is erased and switched to a new extra block. The following write requests are served with the previous extra block until its clean pages are exhausted.

Therefore, the garbage collection time of the page mapping FTL is calculated as follows. T_{read} , T_{write}, T_{erase}, and V denote NAND page read time, NAND page write time, NAND block erase time, and valid page count of the victim block, respectively [5].

$$GC_{page\_mapping} = V * (T_{read} + T_{write}) + T_{erase} \qquad (1)$$

As seen in the equation (1), the garbage collection time of the page mapping scheme depends on the valid page count of a victim block. The greedy replacement scheme that selects the most invalidated block as victim [8] makes the garbage collection time the shortest. However, if the device utilization is high, the average value of valid page count in the victim blocks will increase, and the performance of the page mapping scheme degrades.

The block mapping scheme [11] uses a NAND block as mapping granularity. In other words, the logical block number that is calculated from the logical sector number is matched with the physical block number. The offset inside a block is not changed. Upon a write request, FTL searches for a clean block and writes data to the found clean block, together with the unmodified pages of the previous block. The old block is marked as invalid because its data become obsolete, and the mapping table is updated accordingly. Upon a read request, the valid location is easily found from the mapping table by indexing it with the logical block number. The block mapping scheme is vulnerable against a small sized random write pattern. When modifying a small amount of data, the entire block needs to be copied.

Hybrid mapping schemes compromise the page mapping and the block mapping schemes. Basically, they operate the block mapping scheme. But, a portion of NAND blocks are used as write buffer to absorb small sized write requests, which is called log blocks (figure 1). The other blocks are called data blocks. Thus, the visible area to file system is data blocks. Log blocks are not visible to file system and can be regarded as over-provisioning area to improve the performance.

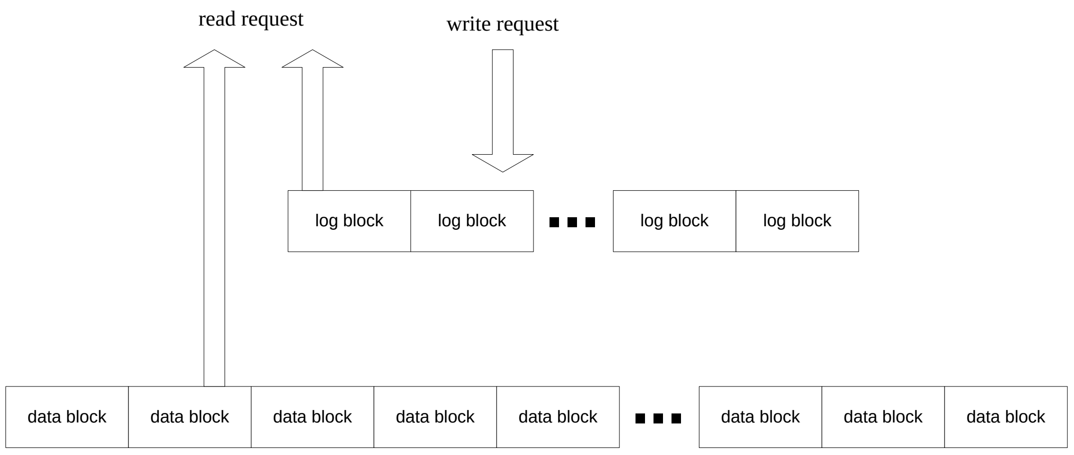

Figure 1. Hybrid Mapping Schemes

Hybrid mapping schemes are classified according to the association between data blocks and log blocks. The block associative sector translation (BAST) scheme [1] associates data block and log block in one to one. Upon a write request, it searches for an associated log block with a target data block and writes the data to the log block. If the data block is not associated with a log block, a clean log block is allocated. If there is no clean log block, the garbage collection is initiated. It selects a victim log block with the LRU replacement scheme. The victim log block is merged with the associated data block. First, a clean data block is allocated. The valid pages of the log block and the old data block are copied to the clean data block. Finally, the victim log block and the old data block are reclaimed by the erase operation. Therefore, the garbage collection time is calculated as follows. N denotes the page counts of a block [5].

$$GC_{BAST} = N * (T_{read} + T_{write}) + 2 * T_{erase} \tag{2}$$

The read request is served by referring to the block mapping table. First, it finds an associated log block. If there is the associated log block and the valid target page is found in the log block, the read request is served from the log block. On the contrary, if there is no valid target page in the log block or if the target data block is not associated with a log block, the read request is served from the target data block.

The drawback of the BAST scheme is that it is vulnerable against random write pattern. Because the log blocks are not shared by data blocks, clean log blocks are frequently scarce, and the victim log blocks are frequently merged even though it has clean pages [2].

In order to solve the problem, the Fully Associative Sector Translation scheme (FAST) [2] allows log blocks shared by multiple data blocks. Upon a write request, it writes the data to the current working log block, regardless of its target data block number. If there is no clean page in the current log block, it is inserted to FIFO (First-In First-Out) list, and a new clean log block is allocated and used as a new current working log. If there is no available clean log block, the garbage collection is initiated. It selects a victim log block from the FIFO list with the FIFO replacement scheme. The victim log block is merged with the associated data blocks. Note that the victim log block can be associated with multiple data blocks. The merge process is repeatedly performed with each of the associated data blocks. After all the merge processes are finished, the victim log block and the old data blocks are reclaimed by the erase operation. Therefore, the garbage collection time is calculated as follows. A denotes the number of the associated data blocks [5].

$$GC_{FAST} = A * (N * (T_{read} + T_{write}) + T_{erase}) + T_{erase} \tag{3}$$

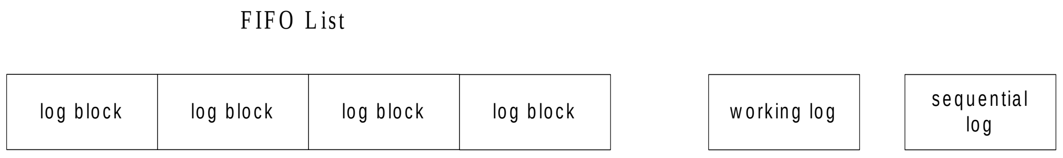

Figure 2. Log Block Management of the FAST Scheme

Therefore, the garbage collection time depends on the association degree. In order to decrease the garbage collection overhead for the sequentially written block, the FAST scheme allocates an additional sequential log block, which serves a sequentially written data block. Figure 2 shows the final log block management of the FAST scheme.

Meanwhile, in order to serve the read request, the FAST scheme should scan all the log blocks to find the valid data, which incurs a considerable computation overhead. In order to address the problem, using a hashed page table is presented [3]. The hashed page table maintains the location information of valid data that reside in the log blocks.

By referring to the hashed page table, the computation overhead problem of the FAST scheme is reduced to the competing degree with the BAST scheme [3].

The Shared BAST (SBAST) scheme [4] allows a log block shared by multiple data blocks, similarly with the FAST scheme. The difference is that it does not allow a data block to use multiple log blocks in order to restrict the search space upon a read request. The SBAST scheme delivers a lower average performance than the FAST scheme because log blocks are not always fully utilized.

3. Device Utilization and Dynamic Over-provisioning

In the page mapping scheme, all blocks except an extra clean block for the garbage collection are visible to file system. Thus, the visible area is almost the same with the real device size if there is no explicit over-provisioning to improve the performance. The interesting point in the page mapping scheme is that unused space functions as the over-provisioning area. For example, if 10 % of the device size has valid data, in other words, if the device utilization is 0.1, 90 % of the entire space functions as the over-provisioning area for the valid data, because the garbage collection is delayed until the rest 90 % of the space are all invalidated. Thus, the page mapping scheme delivers a good performance when the utilization is low. Note that low utilization implies that the over-provision is high. In contrast, when the utilization is high, the performance will degrade because the over-provisioning is low and thus the garbage collection will be frequently initiated.

In reality, the utilization from the FTL perspective tends to increase as time passes because of the following reason. The utilization from the file system perspective depends on users. If users create more files or extends the file size, the utilization from the file system perspective increases. In contrast, if users delete files or shrink the file size, the utilization decreases. However, the utilization from the FTL perspective is not decreased by deleting or shrinking files. Deleting or shrinking files is generally served by modifying the metadata of files and file system. No modifications are done to the data blocks of the files. Thus, FTL cannot know the file deletion or file shrink event, and it thinks that the deleted or shrunk file data are still valid. They are preserved even in the garbage collection.

As a result, the utilization from the FTL perspective continuously increases as time passes. Thus, the performance of NAND-based block devices that deploy the page mapping scheme degrades gradually as time passes. In order to address the problem, TRIM command was proposed in the SATA specification. TRIM is a SATA command that sends the deleted or shrunk logical sector numbers to FTL. Using TRIM command, the utilization of the FTL perspective can be decreased when users delete or shrink files. However, legacy operating systems such as window XP do not support the TRIM command, and implementing the TRIM with low overhead is not straightforward in RAIDs with parity bits [9].

Thus, when evaluating the FTL performance, the device utilization should be considered. However, previous researches that evaluated the FTL performance have not sufficiently considered the utilization [5-7]. Thus, in this work, we vary the device utilization and measure the FTL performance.

Meanwhile, in the hybrid mapping schemes, the visible area to file system is only data blocks. Log blocks are invisible. Log block area is an explicit over-provisioning space in order to improve the performance. The performance of the hybrid mapping scheme is mainly determined by the size of this over-provisioning space. For example, if a sufficient number of log blocks are used in the FAST scheme, the time that a log block is selected as victim after being inserted to the FIFO list extends and thus written pages have the enough chances to be overwritten, namely invalidated. Therefore, if the log blocks are sufficient, the victim blocks will be mostly invalidated in the most cases, and the garbage collection time will be decreased, as seen in the equation (3).

The previous researches that evaluated the FTL performance fixed the log block counts regardless of the device utilization and measured the performance of the hybrid mapping schemes. Thus, when the utilization is low, the page mapping scheme gets a relative advantage because it exploits the unused space as the over-provisioning area, while the hybrid mapping schemes have the fixed number of log blocks regardless of the device utilization.

In fact, in the hybrid mapping schemes, unused data blocks can be used as log blocks. For example, if the device utilization is low and there are a sufficient number of clean data blocks, clean data blocks are switched to log blocks, and the garbage collection is delayed. If the device utilization gets higher and there is no enough clean data blocks, the garbage collection process is initiated and the victim log block is merged with the associated data blocks and switched to a clean data block. Therefore, dynamic over-provisioning can be also applied to the hybrid mapping schemes, and we can fairly evaluate the FTL performance.

4. Performance Evaluation

In order to evaluate the FTL performance, we perform a trace-driven simulation. We gathered a real workload in window XP PC formatted with NTFS file system with diskmon tool, while updating operating system, installing applications, accessing the internet, editing documents, writing programs, and so on. The target partition was 66 GB in size. The total amount of read bytes was 201 GB and the written bytes were 187 GB. Among 66 GB space, about 4 GB space has not been written at all. In other words, the real device utilization was about 0.94.

A target NAND-based block device is modeled as follows. We assume that 8 NAND flash memory chips are linked in 2-channel & 4-way structure in order to enlarge the capacity and the performance, which is a general architecture of SSDs. In this structure, 8 physical pages compose one clustered page, and similarly 8 physical blocks compose one clustered block. Each clustered one can be regarded as a virtual one. Thus, the simulated device is one virtual NAND flash memory chip, whose page is 16 KB in size and a block is 1 MB in size, under the assumption that a physical page size of each physical chip is 2 KB and a block is 128 KB. The read and write latencies of a virtual page is calculated as (physical page read/write latency + (channel latency * 4)), because two channels can transfer the data at the same time and in the NAND chips connected to the same channel, reading or writing a page from a flash chip is performed at the same time while transferring data from another chip via channel. The erase latency of a virtual block is the same with a physical block erase latency because the transferring time of the erase command is very short compared to the data transfer time and it can be neglected. A physical page read/write latency and a block erase latency is assumed 25 us, 200 us, 2 ms, respectively. Transferring time of a physical page is 70 us.

Meanwhile, in order to vary the device utilization, we vary the target device capacity from 78 GB to 155 GB, where the utilization is 0.8 to 0.4.

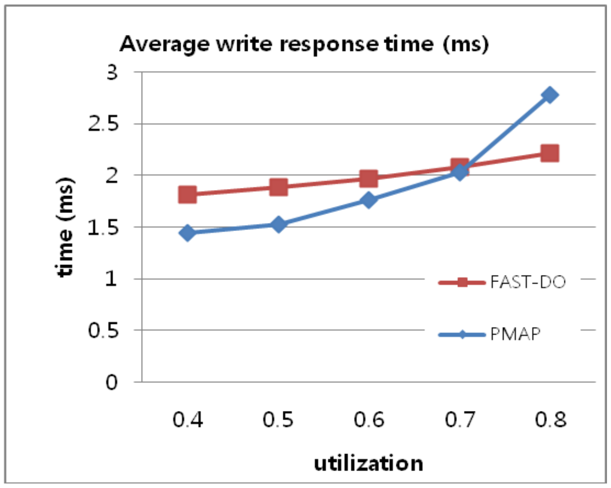

Figure 3. Average Write Response Time

Figure 3 shows the average write response time. The x-axis denotes the device utilization. The y-axis denotes the average write response time in milliseconds. We depict only the write latency because the read latency is similar among the FTL schemes. We compare the performance of the page mapping and the FAST scheme with dynamic over-provisioning. Other hybrid schemes were excluded because they show a worse performance than the FAST scheme. Thus, we implement the dynamic over-provisioning only to the FAST scheme. In the figure, PMAP denotes the page mapping scheme and FAST-DO denotes the FAST scheme with the dynamic over-provisioning.

The result shows that the page mapping scheme delivers a shorter latency than the FAST scheme at lower utilization than 0.7. The performance difference ranges from 2.5 % to 26 %. As the utilization is low, the gap increased. Even though the page mapping scheme delivers a good performance at low utilization, it is vulnerable against high utilization. As the utilization increases, its write latency steeply increases, and becomes longer than that of the FAST scheme at the utilization 0.8. The performance difference reaches to 21 %. The gap will increase more at higher utilization. The FAST scheme was less sensitive to the high utilization. This is a different result from the general agreement that the page mapping scheme is the best from the performance aspect. When considering device utilization and applying dynamic over-provisioning to the FAST scheme, the FAST scheme outperforms the page mapping scheme at the high device utilization.

Therefore, in the computing systems where the TRIM command can be used and the FTL perspective utilization is maintained to be low, the page mapping scheme is the best candidate from the performance aspect. However, in the computing systems where the TRIM command is hard to be applied and the FTL perspective utilization tends to increase gradually, the hybrid scheme with dynamic over-provisioning can be an adequate candidate.

5. Conclusion

In this work, we evaluated the FTL performance considering the device utilization and applying dynamic over-provisioning to the hybrid scheme. The trace-drive performance evaluation result showed the followings. First, the page mapping scheme delivered the best performance at low utilization by using the unused space as over-provisioning area. Second, however, it was vulnerable against high utilization. The average write latency steeply increased as the utilization grew higher. Third, the FAST scheme with dynamic over-provisioning is less sensitive to the utilization. When the utilization is low, it delivered a worse performance than the page mapping scheme. However, at high utilization, it outperformed the page mapping scheme. Fourth, if the dynamic over-provisioning is applied, the hybrid scheme such as the FAST scheme delivers a competing performance with the page mapping scheme.

The limit of the work is the restricted number and the type of the trace file. In order to evaluate the performance varying the utilization, we need more traces of various operating systems whose actual utilization is high, which is one of our future study. The performance of other hybrid schemes that enhanced the FAST scheme also needs to be evaluated applying dynamic over-provisioning.

Acknowledgements

This study was financially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2013R1A1A2011586).

References

[1] J. Kim, J. M. Kim, S. Noh, S. Min and Y. Cho, “A space-efficient flash translation layer for compact flash systems”, IEEE Transactions on Consumer Electronics, vol. 48, no. 2, **(2002)**.
[2] S. Lee, D. Park, T. Chung, W. Choi, D. Lee, S. Park and H. Song, “A log buffer based flash translation layer using fully associative sector translation”, ACM Transactions on Embedded Computing Systems, vol. 6, no. 3, **(2007)**.
[3] I. Shin, “Reducing Computational Overhead of Flash Translation Layer with Hashed Page Tables”, IEEE Transactions on Consumer Electronics, vol. 56, no.4, **(2010)**.
[4] I. Shin, “Light Weight Sector Mapping Scheme for NAND-based Block Devices”, IEEE Transactions on Consumer Electronics, vol. 56, no. 2, **(2010)**.
[5] I. Shin, “Evaluating the Worst-Case Performance of Flash Translation Layer”, Communications in Computer and Information Science, vol. 341, **(2012)**.
[6] I. Shin, “Performance Evaluation of the Sector Mapping Schemes for NAND Flash Memory Considering Mapping Table Size”, Communications in Computer and Information Science, vol. 351, **(2012)**.
[7] I. Shin, “Second Chance Replacement Considering a Garbage Collection Cost of FAST scheme”, SERSC International Journal of Multimedia and Ubiquitous Engineering (IJMUE), vol. 7, no. 2, **(2012)**.
[8] A. Kawaguchi, S. Nishioka and H. Motoda, “A flash-memory based file system. Proceedings of USENIX Technical Conference”, **(1995)**.
[9] N. Jeremic, G. Mucl, A. Busse and J. Richling, “The pitfalls of deploying solid-state drive RAIDs”, Proceedings of the 4th Annual International Conference on Systems and Storage, **(2011)**.
[10] A. Ban, “Flash file system”, U.S. Patent, vol. 5, **(1995)**, pp. 404-485.
[11] A. Ban, Flash file system optimized for page-mode flash technologies”, U.S. Patent, vol. 5, **(1999)**, pp. 937-425.

Author

Ilhoon Shin

Ilhoon Shin received the B.S., the M.S., and the ph.D degrees in computer science and engineering from Seoul National University, Korea. He is currently an assistant professor of the department of electronics and information engineering at Seoul National University of Science & Technology. His research interests include storage systems, embedded systems, and operating systems.