2012 IEEE 15th International Conference on Computational Science and Engineering

MNK: Configurable Hybrid Flash Translation Layer for Multi-Channel SSD

Gyudong Shim, Sung Kyu Park, and Kyu Ho Park
Department of Electrical Engineering
Korea Advanced Institutue of Science and Technology (KAIST)
305-701, Guseong-dong, Yuseong-gu, Daejeon, Korea
Email: {gdshim, skpark}@core.kaist.ac.kr, kpark@ee.kaist.ac.kr

***Abstract*—For multi-channel Solid State Disks (SSDs), hybrid Flash Translation Layer (FTL) schemes were developed for increasing I/O parallelism and reducing garbage collection overhead. However, they still suffer from the high read and write latency, block thrashing problem, and load balancing problem. In order to overcome these problems, we design a configurable hybrid FTL, called MNK. MNK consists of a configurable mapping scheme, recycling log block scheme, and load balancing scheme. By applying the configurable mapping scheme, we can not only exploit the multi-channel architecture of SSD for I/O parallelism but also achieve bounded read/write latency with low garbage collection overhead by controlling the number of modules in striping (M), the number of logical blocks in a group (N), and the maximum allowed log blocks for a group of N logical blocks (K), respectively. In the recycling log block scheme, we can achieve low garbage collection overhead by reducing erase operations. Through the load balancing scheme, we make the erase counts of multiple modules even, thereby increasing the lifetime of SSD. In order to evaluate the performance of the proposed MNK scheme, we use a trace-driven simulator. MNK finally reduces read/write latency by up to 81% compared to previous hybrid FTL schemes such as MCSplit and SubGroup.**

***Keywords*-Storage Management; Hybrid Flash Translation Layer; Multi-Channel Solid State Disk; Garbage Collection;**

I. Introduction

NAND flash-based Solid State Disks (SSDs) are used as the main storage devices in mobile and general computing environments. An SSD consists of multiple NAND flash memories that are connected in a multi-channel architecture to expand the capacity and increase the throughput [1]. In order to emulate a block-device interface that is used for HDD, an SSD contains a special layer, called the Flash Translation Layer (FTL). Its main roles are the address mapping for translating logical addresses to physical addresses and garbage collection for reclaiming invalid pages.

Address mapping involves the translation of logical addresses of the file system to physical addresses of NAND flash memory. It is necessary to hide the erase-before-write characteristic of NAND flash memory. FTL stores the mapping information in a mapping table. According to the mapping unit size, the size of a mapping table would be changed. If the mapping unit is a page, the mapping table size increases to 2MB per 1GB storage considering that a page is 2KB. On the other hand, if the mapping unit is a block that contains several pages, the table size decreases to 32KB per 1GB storage considering that a block is 128KB. However, it results in degrading performance. In order to reduce the mapping table size and improve the performance, previous FTL schemes use hybrid mapping which applies both block-level mapping for data blocks and page-level mapping for log blocks [2]-[9]. The log blocks are allocated to only small portion of physical blocks due to the mapping table size. Whenever the number of available log blocks is less than a threshold value, it is necessary to perform the garbage collection operation which reclaims invalid pages for making new free log blocks.

It is important to reduce the garbage collection overhead because the garbage collection operation, which consists of valid page copies and block erase operations, critically affects the overall performance. The exclusive log block management scheme can guarantee the bounded read/write latency by allocating a log block for only one logical block [2]. However, it can suffer from low log block utilization on random access patterns, called the block thrashing problem. In order to solve this problem, the shared log block scheme which shares all logical blocks into one log block is presented [3], [5]. It can reduce the number of garbage collection operations, but it can suffer from high read/write latency. If a reclaimed log block contains valid pages from multiple logical blocks, each data block and log block are erased after creating a new data block for each logical block. As a result, the garbage collection causes extremely high read/write latency. Therefore, the hybrid log block management scheme restricts the number of logical blocks in a log block for achieving both advantages of the exclusive log block management scheme and shared log block management scheme [4], [6], [7].

It is also important for FTL to consider the multi-channel architecture of SSD as well as reduce the garbage collection overhead for improving the performance. In the multi-channel architecture, we consider to increase I/O parallelism and balance read, write, and erase operations. While MCSplit [8] and SubGroup [9] can increase I/O parallelism with the striping scheme, they has high garbage collection overhead and low log block utilization which means that the log blocks are erased with many unwritten pages. They also show that some channels are overloaded with page copy

978-0-7695-4914-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICCSE.2012.68

and erase operations, thereby resulting in low throughput and high read/write latency with shortened life time.

In order to achieve high I/O parallelism and low garbage collection overhead, we propose a configurable hybrid FTL scheme, MNK, which consists of a configurable mapping scheme, recycling log block scheme, and load balancing scheme. A configurable mapping scheme provides I/O parallelism to the hybrid log block management scheme. It can increase I/O parallelism and achieve the bounded read/write latency with low garbage collection overhead. A recycling log block scheme alleviates the problem of premature erase operations over the blocks with many free pages by reusing log blocks with many free pages for new log block allocation. It can increase the log block utilization and significantly reduces total erase operations by increasing the log block utilization. Finally, a dynamic module allocation method can balance the total erase counts and write counts over modules

The remainder of this paper is organized as follows. This paper begins with background about the operation flow in hybrid FTL. Section III describes the configurable MNK mapping, recycling log block, and load balancing schemes. The simulation results are explained in Section IV. Section V concludes this paper and presents future work.

II. Background

As a backgroud of our work, we describe the read and write operations in SSD which uses a hybrid FTL, as shown in Fig. 1. For example, write LPN 0 and 4, which represents the write request of new logical pages 0 and 4 is issued from the file system to SSD. The new logical page number (LPN) 0 and 4 are translated into the physical block number (PBN) 0 and 1 and physical page number (PPN) 0 and 1 by a hybrid FTL. The old logical page number 0 and 4, which were previously written in a data block, are invalidated and new logical page number 0 and 4 are written to a log block. If using only block-level mapping, all of the pages in a block are updated, thereby it results in degrading the performance. By exploiting the log block, which is managed by page-level mapping, a hybrid FTL can efficiently manage newly updated pages. When the read requests of logical page number 0 and 4 are issued, new logical page number 0 and 4, which are written in the log block, are served. This is a basic write and read operation flow in hybrid FTL. Upon policies of hybrid FTL, data blocks and log blocks are managed differently such as the restriction of the associativity which is the number of logical blocks written into a log block.

III. MNK: Configurable Hybrid FTL

We propose a configurable hybrid FTL scheme, called MNK, for improving the performance of multi-channel SSDs. MNK consists of a configurable mapping scheme, recycling log block scheme, and load balancing scheme. From now on, we describe the detailed design and implementations of those schemes in following sub-sections.

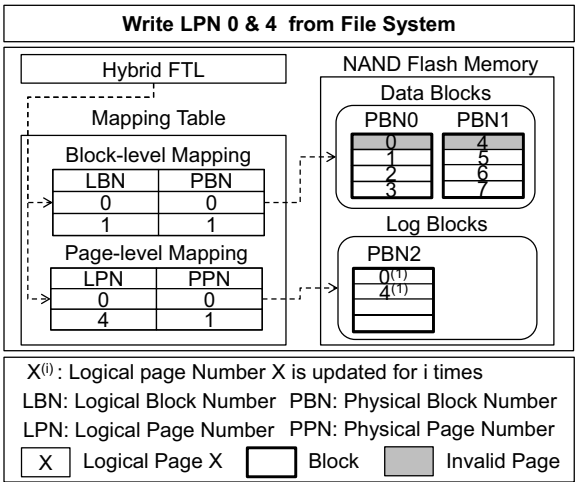


Figure 1. Operation Flow in Hybrid FTL

A. Configurable Mapping Scheme

MNK has three important parameters, which are M, N, and K. MNK manages the followings: M, which is the number of modules in striping; N, which is the number of logical blocks in a group; and K, which is the maximum allowed log blocks for a group of N logical blocks, as shown in Fig. 2 with the cases of MNK (2, 4, 8) and MNK (4, 8, 16). As M value increases, we can achieve high I/O parallelism, but it can also make high garbage collection overhead. This is because a logical block is distributed into multiple channels, thus many physical blocks are rearranged when a garbage collection operation is performed. As we assign a large N value, we can expect higher log block utilization because many logical blocks can share K log blocks, thus reducing garbage collection overhead which consists of valid page copies and block erase operations. However, this configuration can also make high read/write latency. As K value increases, we can achieve the opportunity to allocate updated page into log blocks, but there is a possibility to create a block thrashing problem due to low block utilization. Therefore, it is important to control M, N, and K values to achieve better performance.

There are many trade-offs when determining M, N, and K. M is assigned according to the architecture of SSD, and N and K values are determined by application patterns. If an application is latency-sensitive, we decrease N value, which is the number of logical blocks, thereby reducing read/write latency. While an application requires high throughput, we can increase N value for reducing the garbage collection overhead. If an application has much of frequently updated data, we increase K value, which is the maximum allowed log blocks, to reduce the number of valid page copies by invalidating the data in log blocks. This is the main contribution of MNK, which can control M, N, and K values according to application pattern and user's requirements.

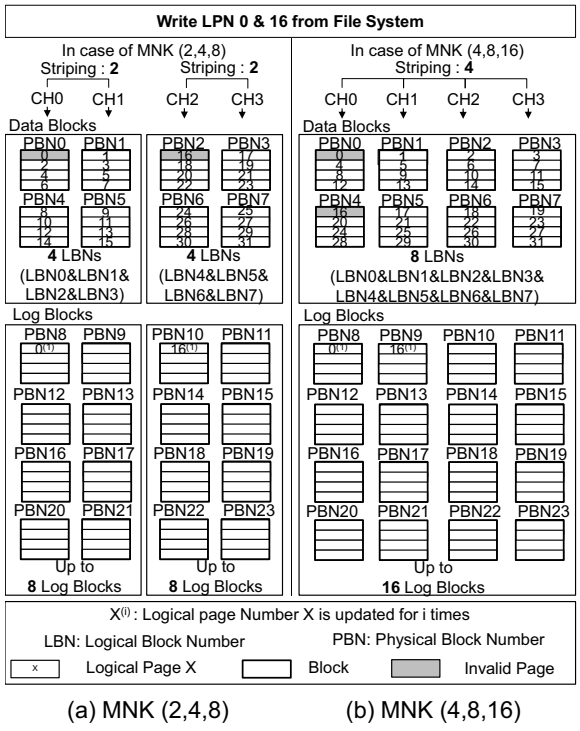


Figure 2. MNK Layout of (2,4,8) and (4,8,16)

Figure 2 shows the layout of MNK mapping of (2,4,8) and MNK mapping of (4,8,16). MNK mapping of (2,4,8) has two striping levels, four logical blocks in a group, and eight maximum allowed log blocks for a group of four logical blocks. Therefore, the logical block number (LBN) 0, 1, 2, and 3 can make a group with two striping levels and share eight log blocks. Similar to MNK mapping of (2,4,8), MNK mapping of (4,8,16) has four striping levels, eight logical blocks in a group, and sixteen maximum allowed log blocks for a group of four logical blocks. In these configurations, we show the scenario when when the logical pages 0 and 16 are updated and written into log blocks. When logical page 0 is updated, logical page 0 in the data block group is invalidated, and updated logical page 0 is written to the log block group. Then, logical page 16 is also a similar process with logical page 0. These invalidated values are reclaimed by garbage collection operations. Because we allocate pages in the log block group by considering a multi-channel architecture, we can increase I/O parallelism in the data block group as well as that in the log block group.

B. Recycling Log Block Scheme

In MNK, the garbage collection operation is performed whenever K log blocks are fully written by updated logical pages or the number of free blocks is less than a threshold value. When selecting a victim during a garbage collection operation, MNK chooses a group of logical blocks to be reclaimed from the LRU list of accessed logical groups. The oldest logical group is chosen as a victim because it is not likely to be updated immediately. The victim selection is done by only O(1) complexity, while ASA [11] chooses a victim block after comparing the cost of each block to be reclaimed from entire log blocks. This process requires additional meta-data of log blocks in memory.

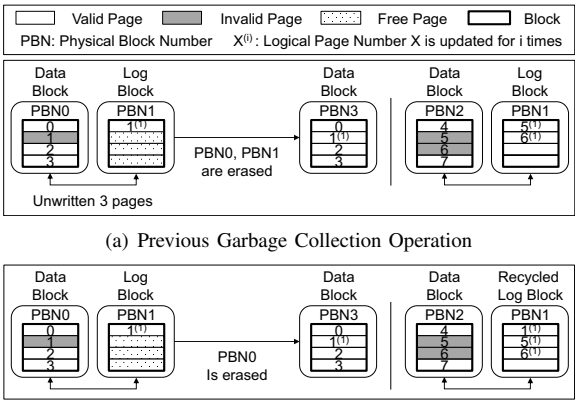


Figure 3. Example of Recycling Log Block Scheme

During a garbage collection operation, MNK recycles log blocks in order to gain better log block utilization. We found that many log blocks with many unwritten pages are erased during a garbage collection operation in hybrid FTLs with exclusive log block allocation, such as BAST, MCSplit, and SubGroup. This problem not only reduces the expected lifetime of blocks but also increases the read/write latency of garbage collection operation. Therefore, we propose the recycling log block scheme for better log block utilization and reduce the number of erase operations.

As shown in Figure 3(a) which is the case of BAST, SubGroup, and MCSplit, PBN1 with many unwritten pages is reclaimed by garbage collection. PBN1 has three free pages, but it is erased in order to obtain a free block for log block allocation for logical block 1, which contains logical pages from 4 to 7. We assume the total log block is limited to one. A write request of logical pages 5 and 6 triggers garbage collection to obtain a new log block for the log block in case of BAST, SubGroup, and MCSplit. Since PBN1 has non-zero offset logical pages in a block, it should be reclaimed by a garbage collection. During the garbage collection, PBN0 and PBN1 are erased after reconstructing new data of PBN3.

In MNK, we can reduce the erase count and increase log block utilization by recycling log block PBN1 instead of erasing it. The physical block PBN1 is reallocated into the next log block for logical block 1, as shown in Figure 3(b). Since the update logical page 5 is not zero offset within the logical block, the log block cannot be switched to a data block. The log block for logical block 1 should be reclaimed by a garbage collection operation later. Therefore, it is safe

to allocate the recycled log block for the random access pattern. However, if a recycled log block is allocated to a sequential log block, the log block cannot be switched to data blocks. For this reason, the recycled log block should be allocated with a random access pattern only. Because the logical pages have non-zero offset with a random pattern, it is safe to allocate PBN1 as the log block with three pages that are larger than the two pages to be written.

However, it is not always good to operate the recycling log block scheme. Therefore, we exploit a threshold value, RECYCLE_TH. If RECYCLE_TH is 0.6, log blocks whose written pages are less than 60% are recycled. If RECYCLE_TH is set to 0, there is no recycled log block, while any log blocks with unwritten pages can be recycled if RECYCLE_TH is set to 1. This threshold value will be determined by various experiments in Section IV.

C. Load Balancing Scheme

Load-balancing over multiple modules is crucial because it influences the lifetime and overall performance of SSDs. In order to maximize the lifetime of SSDs, each module should be erased evenly. A skewed distribution of erase operations over modules can reduce the expected time considerably. If the limit of erase counts over a module is set by a value, the module cannot be updated anymore after the erase count over the module reaches the value. The performance is also affected by load-balancing of valid pages over modules. If accessed data are positioned in a specific module, the module can be a bottleneck and other modules are under-utilized.

In order to achieve load-balancing on MNK, MNK applies compensation schemes of ASA [11] to log block allocation with a best effort policy. When a log block is allocated, the block from a module is allocated to compensate valid pages and erase counts considering the sequential/random pattern. In addition, the victim block in a garbage collection is chosen from a specific module to be compensated by the LRU list of accessed logical groups. Log blocks in MNK can be classified into hot or cold blocks according to the random/sequential patterns. Since a large-sized sequential write has cold pages, it will increase the valid pages in a module. On the other hand, small-sized random log blocks would cause a garbage collection operation, which increases the erase count of the module. Therefore, MNK allocates log blocks from modules dynamically, considering the erase count over modules and the number of free blocks in a module. Since counting every valid page per module triggers frequent updates on meta-data, MNK approximates the number of valid pages in a module by the number of free blocks in a module.

When a log block with sequential pattern is allocated, the module with the least free blocks is chosen. When a log block with random pattern is allocated, the module with the least erase count is chosen. Since MNK allocates M log blocks from M modules as a group for striping, the modules are grouped statically. If there are L modules, there are L/M module groups. The load balancing scheme is applied at the module group level. For managing the number of free blocks or erase count per module group, MNK manages two lists which are ordered by the number of free block or erase count. The ties are resolved by the last access time.

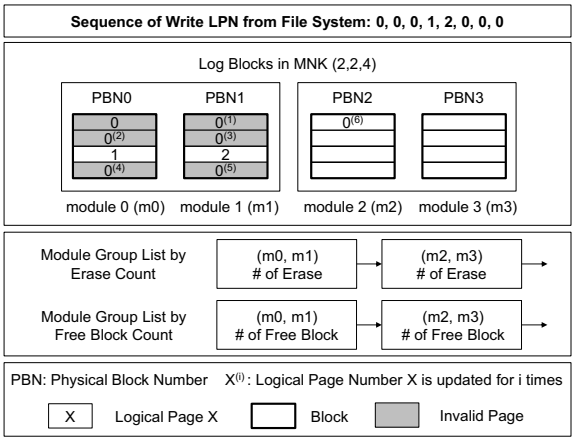


Figure 4. Example of MNK Load Balancing Scheme

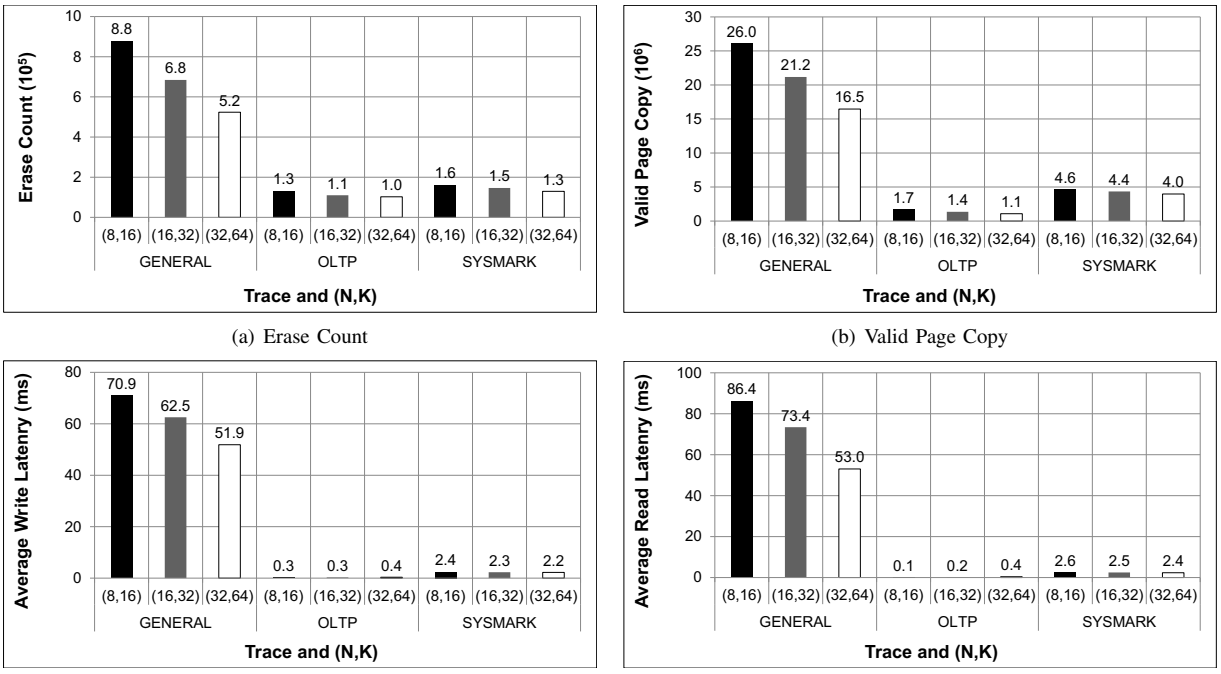
Examples of load balancing schemes are shown in Figure 4. The logical pages are updated by a sequence of 0, 0, 0, 0, 1, 2, 0, 0, and 0. M, N, and K values are set to 2, 2, and 4, respectively. The log block allocation occurs just before the first and the ninth write. Since the first logical page 0 is regarded as a random pattern due to the small size, the log blocks for logical block group 0, which consists of logical block 0 and 1, are allocated by the modules 0 and 1. Whenever allocation or collection occurs, the list is updated by moving the allocated module group into the last position of the same key value, the erase count or free blocks. When log blocks with a random pattern are allocated, the two lists are updated simultaneously. When log blocks with sequential pattern are allocated, only the list with free blocks is updated. When a garbage collection occurs, the two lists are updated according to free blocks and the erase count. The compensation of the erase count and the number of free blocks is triggered at the M block level because MNK allocates or reclaims log blocks with M blocks.

IV. Evaluation

In this section, we describe the evaluation environment such as the simulator, trace characteristics, and performance metrics and the results compared to previous hybrid FTLs.

A. Evaluation Environment

The performance of MNK is evaluated by a trace driven simulator. The simulator supports a write buffer, various FTL schemes, and multiple NAND flash memories. A write buffer is managed by block-level LRU. The simulator can configure



(a) Erase Count

(b) Valid Page Copy

(c) Average Write Latency

(d) Average Read Latency

Figure 5. Performance Effect according to N and K Parameters (M is set to 8 for full I/O parallelism)

three FTL schemes: MNK, SubGroup, and MCSplit. Multiple NAND flash memories construct in 8-channel 4-way, which is used in current commercial SSDs.

We used three workload traces which are OLTP, GENERAL, and SYSMARK traces [13], [14]. The OLTP trace was extracted from OLTP applications running at a large financial institution. This trace had a large number of random writes whose sizes were about 5.3KB. It was made available courtesy of Ken Bates from HP, Bruce McNutt from IBM, and the Storage Performance Council. GENERAL and SYSMARK traces were obtained from a Microsoft Windows-based laptop computer. These traces represent the workload of typical PC usage scenarios. GENERAL trace was obtained from a 5-day long general PC usage and SYSMARK trace was collected from SYSmark 2007 Preview.

We first investigated the effect of M, N, and K and recycling threshold value, RECYCLE_TH. After that, we compared the overall performance of MNK with that of MCSplit and SubGroup. We evaluate the erase count, valid page copies, average write latency, and average read latency as the performance metrics. We also show the erase counts in all modules for identifying the load balancing effect.

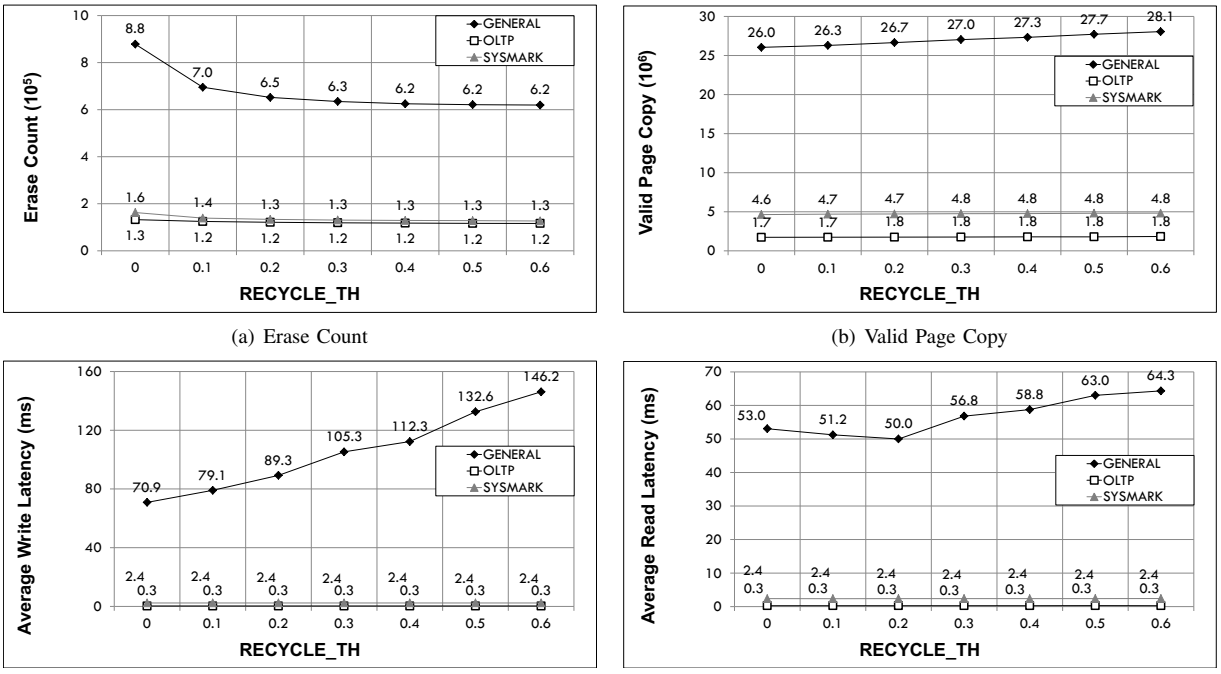
B. Effect of Parameters

We provide the evaluation results according to parameters which are M, N, and K values and RECYCLE_TH. From the results, we can determine appropriate values that give the best performance for comparison with other FTL schemes.

1) Result with M, N, and K Values: The first experiment showed performance effect according to M, N, and K parameters. We set M to 8 in order to achieve full I/O parallelism by exploiting full channels (8-channels) in the simulator.

Figure 5(a) and 5(b) represent a garbage collection overhead that consists of the erase counts and the number of valid page copies. The results show that the garbage collection overhead is reduced as N and K values increase. In GENERAL trace, the erase count is reduced by up to 40.9% and the number of valid page copies is reduced by up to 36.5%. This is because the higher N and K values become the higher log block utilization, thus making a small number of garbage collection operations. It can result in improving the performance of SSD.

Figure 5(c) and 5(d) describe the average write and read latencies. In GENERAL and SYSMARK traces, both read and write latencies are reduced by up to 18.5% as N and K values increase. This is because the garbage collection overhead is reduced. In OLTP trace, however, read and write latencies are increased as N and K values increase. Because the higher N and K values have more logical blocks for garbage collection, the read and write latencies can be increased in this case. For some request patterns, there is a tradeoff between garbage collection overhead and performance. Therefore, it is important to determine appropriate N and K values according to the pattern of applications.



(a) Erase Count

(b) Valid Page Copy

(c) Average Write Latency

(d) Average Read Latency

Figure 6. Performance Effect according to RECYCLE_TH ((M,N,K) = (8,32,64))

2) Result with RECYCLE_TH Values: The effect of the recycling log block scheme is given according to various RECYCLE_THs from 0.1 to 0.6. We set M, N, and K values to (8, 32, 64) which shows low garbage collection overhead.

Figure 6(a) and 6(b) show garbage collection overhead. As RECYCLE_TH increases, the erase count is reduced, but the valid page copies are increased. Although the recycling log block scheme needs to copy more valid pages, it can reduce the erase count by increasing log block utilization. If RECYCLE_TH is 0.1, the total erase count is reduced by 21% with 1% additional valid page copy overhead.

Figure 6(c) and 6(d) represent the average read/write latency. From the results, additional page copy makes considerable read/write latency penalties in GENERAL trace. As RECYCLE_TH becomes larger, average read/write latency is increased. In GENERAL trace, there are heavy requests and queuing delays for garbage collection. A small increase of valid page copies causes a high read/write latency.

In OLTP trace, the recycling log block scheme cannot critically affect the performance. This is because the log block utilization in this trace is high; thus, it cannot make additional valid page copy overhead. The recycling log block scheme is efficient when the log block utilization is low.

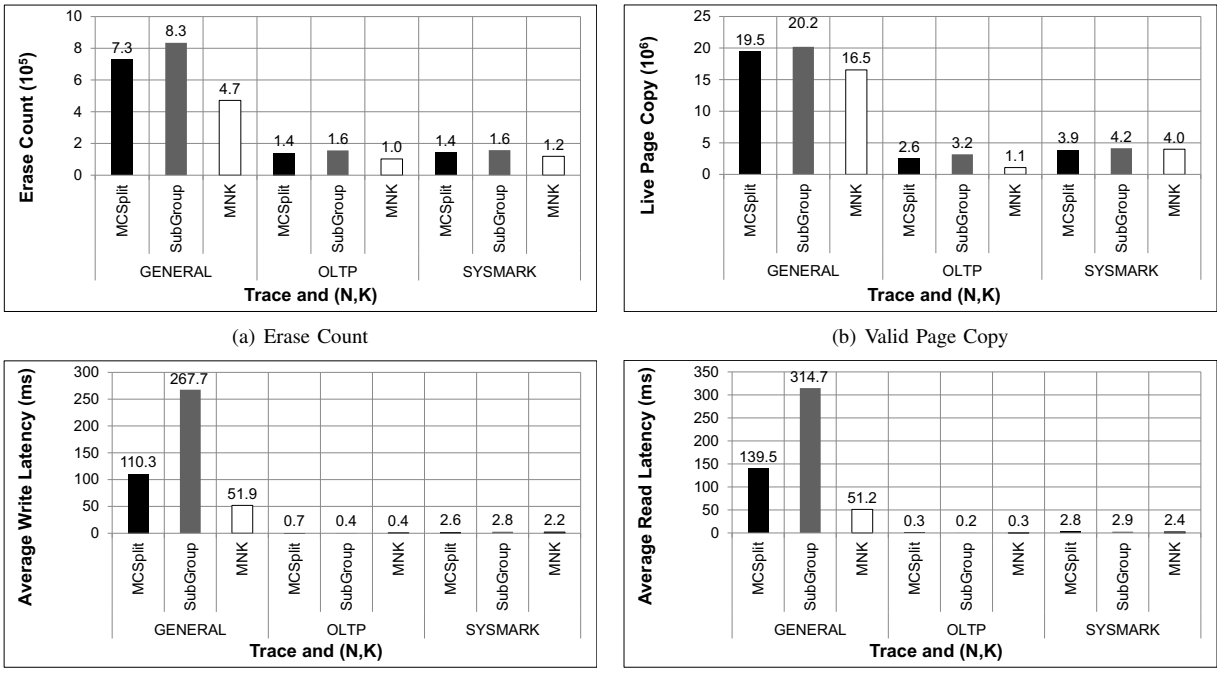
C. Performance Compared with Hybrid FTLs

We provide the evaluation results compared with previous hybrid FTLs which are MCSplit and SubGroup. We first evaluated the erase count, valid page copies, average write latency, and average read latency, which represent the overall performance of FTL. We then provide the erase counts of all modules for showing the load balancing effect.

1) Overall Performance: In order to compare the overall performance with MCSplit and SubGroup, we set M, N, and K values and RECYCLE_TH to (8, 32, 64) and 0.1, respectively. The configuration of (8, 32, 64) can achieve low erase count and a low number of valid page copies and reduce the average write and read latency. The reason why we set RECYCLE_TH to 0.1 is that this configuration shows low garbage collection overhead with negligible valid page copy overhead.

Figure 7(a) and 7(b) show the erase count and valid page copies. The evaluation results show that MNK has fewer erase counts and a smaller number of valid page copies than those of MCSplit and SubGroup. This is because MNK can achieve the high log block utilization with high N and K values. MNK can also reduce the erase count by the log block recycling scheme. MNK provides a lower erase count by up to 43% and decreases valid page copies by up to 66% compared to those of MCSplit and SubGroup.

Figure 7(c) and 7(d) represent the average read/write latency. The results show that MNK can reduce the average read/write latency compared to those of MCSplit and Subgroup. This is because MNK can increase I/O parallelism and reduce the erase count and number of valid page copies. In GENERAL trace, MNK especially can reduce the write latency by up to 81% and the read latency by up to 84%.



(a) Erase Count

(b) Valid Page Copy

(c) Average Write Latency

(d) Average Read Latency

Figure 7. Overall Performance compared to MCSplit & SubGroup ((M,N,K,RECYCLE_TH) = (8,32,64,0.1))

By comparing the previous hybrid FTL schemes of MCSplit and Subgroup, we show MNK reduces the garbage collection overhead which contains the erase count and the number of valid page copies and write and read latencies by exploiting the configurable mapping scheme and the recycling log block scheme. However, there is a tradeoff between garbage collection overhead and latency in some request patterns. When N and K values increase, these can reduce the garbage collection overhead, but it can show the highest latency problem due to the large number of logical blocks, N. Therefore, it is important to determine M, N, K, and RECYCLE_TH. Although we set the same parameters in this evaluation, MNK performs better if we change these values according to the pattern.

2) Load Balancing: From now on, we evaluate the erase count of modules of SSD to show the load balancing effect of MNK compared to MCSplit and SubGroup as the final evaluation. For this evaluation, we also set M, N, and K values and RECYCLE_TH to (8, 32, 64) and 0.1, respectively. Figure 8 shows the evaluation result of the load balancing effect. Because the load balancing scheme is performed in module groups, the erase count of modules that have the same key (module number % 4) is the same. In Figure 8, the x-axis means the module number from 1 to 32 and the y-axis represents the erase count of each module.

As shown in Figure 8, MNK has an even distribution of the erase count over multiple modules. MNK also has the lowest erase count in all traces of GENERAL, OLTP, and SYSMARK traces. This means that the load balancing scheme of MNK is performed well; thus, it can maximize the lifetime and overall performance of SSDs. However, in the cases of MCSplit and SubGroup which use static allocation, some modules have higher erase counts than other modules. This means that those modules cannot be updated anymore. The reason is the SubGroup separate hot group and cold group. This hot group makes a higher erase count.

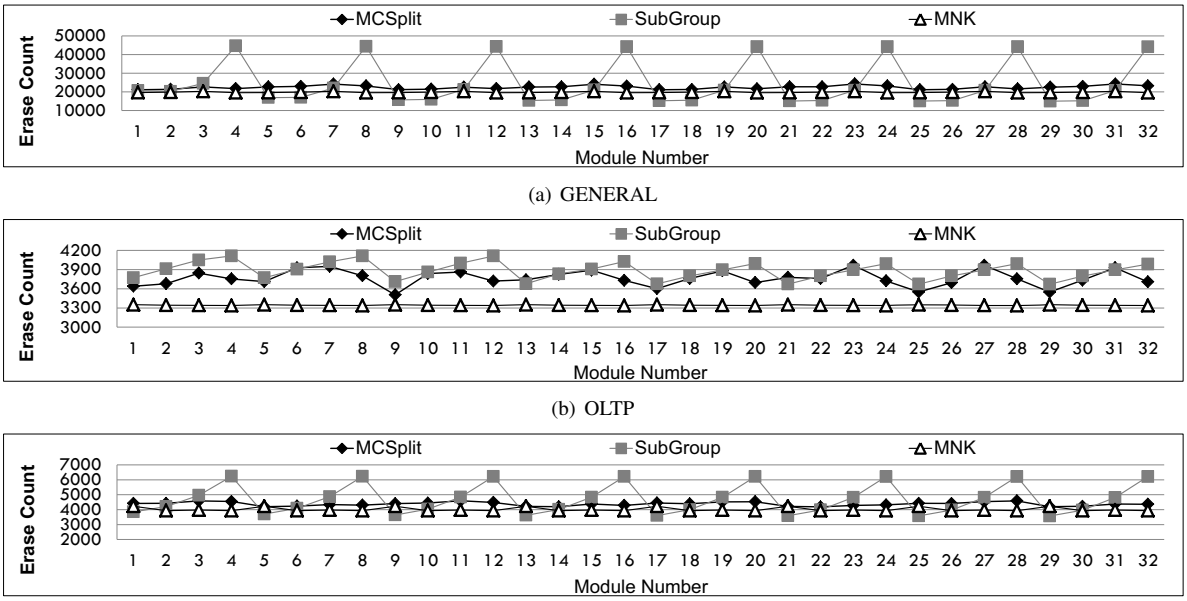
V. Conclusion

We presented a configurable hybrid FTL, called MNK, which exploits the configurable mapping scheme to bound the read/write latency and increase I/O parallelism. We also proposed a recycling log block scheme that addresses the problem of low block utilization during exclusive log blocks. Finally, we make erase counts of multiple modules, even through the load balancing scheme. Therefore, performance and lifetime of SSD are improved. MNK provides better performance than MCSplit and SubGroup by up to 81%.

For future work, we will dynamically change parameters according to the request pattern. We will also integrate MNK scheme with the write buffer management scheme to achieve a synergetic effect. By considering both FTL and the write buffer, we can highly improve performance of SSD.

Acknowledgment

The work presented in this paper was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. 10035231-2010-01.



(a) GENERAL

(b) OLTP

(c) SYSMARK

Figure 8. Load Balancing Effect compared to MCSplit & SubGroup ((M,N,K,RECYCLE_TH) = (8,32,64,0.1))

REFERENCES

[1] Y. J. Seong, E. H. Nam, J. H. Yoon, H. Kim, J.-Y. Choi, S. Lee, Y. H. Bae, J. Lee, Y. Cho, and S. L. Min, Hydra: A Block-Mapped Parallel Flash Memory Solid-State Disk Architecture, *IEEE Transactions on Computers*, vol. 59, no. 7, pp. 905-921, July, 2010.

[2] J. Kim, J. M. Kim, S.H. Noh, S. L. Min, and Y. Cho, A Space-Efficient Flash Translation Layer for CompactFlash Systems, *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366-375, 2002.

[3] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, A Log Buffer-Based Flash Translation Layer using Fully-Associative Sector Translation, *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 3, Jul., 2007.

[4] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim, A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications, *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 4, pp. 1-23, Jul., 2008.

[5] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems, *ACM Operating System Review*, vol. 42, no. 6, pp. 36-42, Oct., 2008.

[6] H. Cho, D. Shin, and Y. I. Eom, KAST: K-Associative Sector Translation for NAND Flash Memory in Real-Time Systems, *Proc. the Conference on Design, Automation and Test in Europe (DATE '09)*, pp. 507-512, 2009.

[7] D. Koo and D. Shin, Adaptive Log Block Mapping Scheme for Log Buffer-based FTL (Flash Translation Layer), *Proc. International Workshop on Software Support for Portable Storage (IWSSPS '09)*, 2009.

[8] J. H. Kim, S. H. Jung, and Y. H. Song, Cost and Performance Analysis of NAND Mapping Algorithms in Shared-Bus Multi-Chip Configuration, *Proc. International Workshop on Software Support for Portable Storage (IWSSPS '08)*, 2008.

[9] J. Park, G.-H. Park, C. Weems, and S. Kim, Sub-Grouped Superblock Management for High-Performance Flash Storages, *IEICE Electronics Express*, vol. 6, no. 6, pp. 297-303, 2009.

[10] J.-Y. Shin, Z.-L. Xia, N.-Y. Xu, R. Gao, X.-F. Cai, S. Maeng, and F.-H. Hsu, FTL Design Exploration in Reconfigurable High-Performance SSD for Server Applications, *Proc. the 23rd International Conference on Supercomputing (ICS '09)*, pp. 338-349, June, 2009.

[11] L.-P. Chang and T.-W. Kuo, An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems, *Proc. the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '02)*, Sep., 2002.

[12] A. Gupta, Y. Kim, and B. Urgaonkar, DFTL: A Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address Mappings, *Proc. the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, 2009.

[13] Y.-G. Lee, D. Jung, D. Kang, and J.-S. Kim, μ-FTL: A Memory-Efficient Flash Translation Layer Supporting Multiple Mapping Granularities, *Proc. the 8th ACM International Conference on Embedded Software (EMSOFT '08)*, pp. 21-30, 2008.

[14] OLTP Trace from UMass Trace Repository, http://traces.cs.umass.edu/index.php/Storage/Storage.