International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.9 (2014), pp.299-310
http://dx.doi.org/10.14257/ijmue.2014.9.9.31

Virtually Separable Block Management in Flash Storage System

Seung-Ho Lim

Department of Digital Information Engineering
Hankuk University of Foreign Studies
slim@hufs.ac.kr

Abstract

File systems treats Flash storage device as a traditional storage media with their logical address. However inside the Flash storage device, Flash Translation Layer (FTL) remaps logical address to physical address to hide physical limitation of Flash memory cells. Due to the address translation, intentional logical separation of file system's layout does not directly applied to physical separation. As a result, the separate-intended file systems' requests are mixed in physical location, which degrades write throughput, as well as Flash's lifecycle. In this paper, we propose virtually separable Block management scheme for Flash storage system by introducing new common command interface and separable Block management in FTL. The experimental results show that the proposed scheme increases IO performance, as well as reduces flash-internal overhead.

Keywords: *Flash Memory, File system, FTL, Virtually separable Block management*

1. Introduction

NAND flash memory and corresponding devices has become one of the main storage media for computing systems including server, personal computers, and mobile systems, with the help of low power consumption, low density, high capacity, and high IO bandwidth. However, due to the physical limits for NAND flash memory, the file systems do not fully utilize bandwidth of storage media, which results in poor user responsiveness and service quality. There are two main physical limits for NAND flash memory. First, cell-erase operation should be preceded by cell-write operation, which means that the data cannot be written in same place without erasing it. Second, the base unit of erase operation is 'Block', whereas base unit of write operation is 'Page', where block is much larger than page. Actually, Block is composed of several Pages.

The physical limitation of NAND flash memory is covered by special firmware called Flash Translation Layer (FTL) [1, 2]. The NAND flash-based devices such as Solid State Disk (SSD) [3] and Multimedia Card (MMC) [4] embed FTL inside their systems as a form of firmware, so FTL runs on controller within the devices. FTL manages address mapping between file system and NAND flash memory, and does GC (Garbage Collection) [5]. File systems can treat Flash device just like usual storage media with the logical block address supported by FTL, and FTL hides internal mapping information and management schemes from host file systems.

Typically for NAND Flash memory, the size of one page is ranging from 4KB to 16KB, and the size of one block is ranging from 128KB to 4MB in recent Flash chip. Thus, the traditional notation for 'block' of file systems is match with the 'page' of flash, and flash 'Block' is much larger concept than file system's 'block'. In our notation, 'block' represents file system's unit, and 'Block' represents flash memory's unit, respectively.

ISSN: 1975-0080 IJMUE

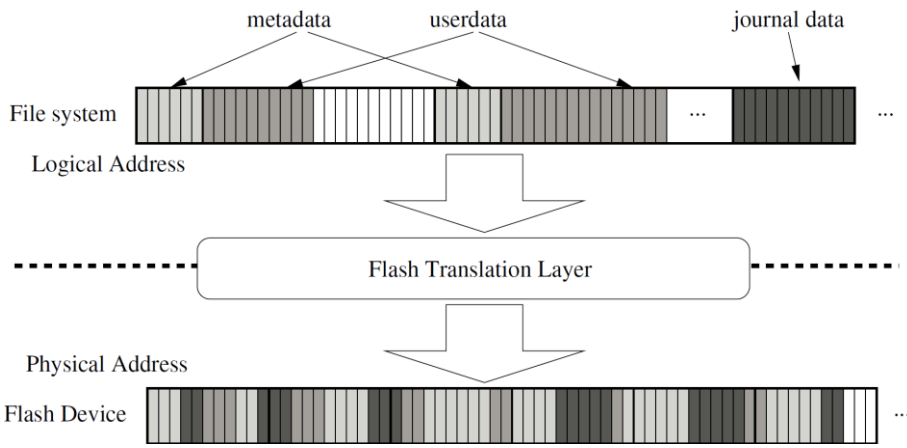


Figure 1. Logical Laout of File System and its Physical Allocation within Flash Device

Since file system uses Flash device as a view of logical address, there is a mismatch for view point between logical block and physical block. As a result, traditional file system architecture and layout design is little effective for Flash-based storage. For instance, many of typical file systems, such as FFS [6], Ext4 [7], and NTFS [8], separates metadata and data in physical location since the characteristics of these two types are totally different. In addition to that, Journaling data of file systems are also recorded in different region in storage [9]. The separation of these is effective for bandwidth and latency if traditional rotational disk is used as storage media, since it has same view of logical address and physical address. However, Flash device has different view of logical address and physical address, so the separation of file system's layout does not directly applied to the physical separation for Flash device. As a result, file system's data are mixed in physical location for Flash memory-based storage system even though they are logically separated by file system on purpose, as described in Figure 1.

The unintentional physical mix for logically separated data from file system degrades IO bandwidth as well as latency, significantly. In this paper, we propose virtually separable Block management scheme for Flash storage system. For the separable Block management system, we propose common command interface between file systems and underline flash device to support the separation. In the proposed system, file system has identifier, i.e., ID, to identify data type. For instance, file system having metadata, user data, and journal data has three types of data to be stored in separated region, the IDs are allocated for each type of data, and the file system makes read/write commands with the logical address, size, and its ID. In the flash devices, the flash Blocks are virtually separated according to the data types. For instance, data and metadata, and journal data are assembled under the separated flash Blocks virtually, where the 'virtually' means that the Blocks are not necessarily consecutive in physical position. When write command arrives, FTL, which makes overall management of flash devices, writes the arriving data to the relevant Block according to its ID.

As a result, Blocks having same ID are virtually connected each other, while Blocks having different ID are virtually separated. This virtual separation can make file system's read/write requests sequentially, which leads bandwidth increase. In addition, the GC efficiency of FTL gets better, which decrease flash-internal overhead and increase lifetime of Flash device. The experimental results show that the proposed scheme increases IO performance, as well as reduces flash-internal overhead.

The remainder of this paper is organized as follows. In Section 2, background and related work is described. The proposed virtually separable Block manage scheme is explained in Section 3, and its performance evaluation is described in Section 4. Section 5 concludes this paper.

2. Background and Related Work

In this Section, the backgrounds and related work are illustrated. Most of the background is referred by our previous work [16].

2.1. NAND Flash Memory and FTL

NAND flash memory is array of memory cells that consist of floating-gate transistors. There are three commands used in NAND flash memory; read, program (write), and erase. The read and program command are related with data transfer between host and Flash devices, whose data unit is Page. The erase command has no data transfer between host and Flash, and the erase is operated at the Block-based. In NAND flash memory, the size of one Page is 4KB and doubles as manufacturing process advances and a Block is composed of 64 or 128 Pages, typically. Due to the size mismatch between write and erase operation, write operation should consider efficient erase operation. Typically, read for one Page consumes about 125us, including Page read to internal chip register and bus transfer from chip register to host side. As a same manner, write for one Page consumes about 200us~400us.

FTL manages address mapping table between logical address of host part and physical address of Flash memory. Indeed, except the mapping management, FTL does many other roles, including wear leveling, garbage collection, bad block management, and request queuing and caching, and so on. However, the mapping management is most important job among many FTL's roles, and others are mostly dependent on the mapping management scheme. The FTL keeps track of the address mapping information between the logical address and the physical address. In this manner, the FTL prevents in-place updates of data and hides the latency of the erase operation. When the number of free pages is insufficient for write operations, free pages should be made by garbage collection (GC), where GC is the process that makes available free region by selecting one Block, moving data of valid Pages to other region, and erasing the Block. Thus, the selected victim Block should have minimum valid Pages for more efficient garbage collection.

According to the mapping granularity, the FTL mapping management scheme can be divided into three categories; Page-level mapping, Block-level mapping, and hybrid-level mapping. In page mapping scheme, mapping table is maintained as a Page level, so that a logical Page number is mapped to a physical Page number in the mapping table. In Block mapping scheme, mapping table is maintained as a Block level, so that a logical Block number is mapped to a physical Block number. Accordingly, a logical Page can be identified by the physical Page offset within the corresponding physical Block number. Hybrid mapping mixes these two mapping table. The advantage of Block-level mapping is that it has small mapping table, however, basically it gives poor performance due to the lack of flexibility. Likewise, Page-level mapping can give high performance with good mapping flexibility, but it requires large mapping table maintained in main memory.

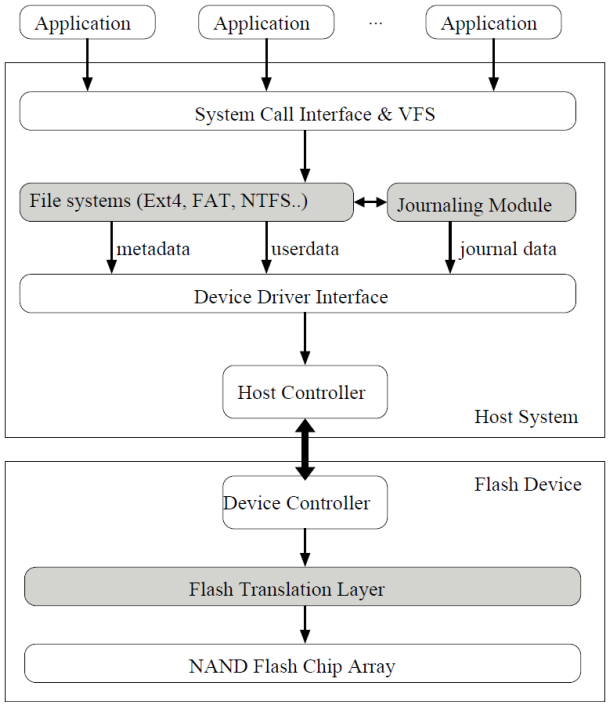


Figure 2. The Overall System and Software Architecture for Flash-based Storage System

2.2. Related Work

There have been some previous researches for FTL. At the early stage of FTL, the design is focused on the mapping management issues to reduce mapping table, while preserving IO bandwidth in portable flash device [10-14]. In [11], one log Block per one data Block is used, and the Page-level mapping is used for log Blocks for performance enhancement. In [11], several Blocks are grouped into superblock, and in superblock, Page-level mapping is used for more efficient use of superblock. Lee et al use only one log Block for all the Blocks with flash memory to reduce log Block management overhead and enhance log Block utility [12]. For more efficient use of blocks, the DFTL [13] deploy demand-based caching of Page-level mapping table, it implies Page-level mapping management outperforms any other Block-level mapping management. Dongzhe et al considered Page-level FTL management for small NAND flash memory system [14].

There is previous work that considers request characteristics and patterns for file system and apples it to flash storage system and FTL development. Chang et al separate metadata requests and userdata requests based on the knowledge of the layout of file systems and metadata filter to reduce file systems performance degradation [15], where different FTL management algorithms are applied to each metadata region and userdata region, explicitly. It is most related work to the work of this paper, in that they tried to separate metadata and userdata in Flash storage device. However, our approach is different from [15] in that there are only one FTL in our system, whereas there are two FTLs in one system. In addition, our system gives much more common design methodology for separating file system's overall data. According to our design, the several file system's region can be separated virtually in Flash device without any addition filtering algorithm, and all the separated regions are managed by only one FTL,

which is efficient for manufacturing Flash storage device. From next Section, the detailed system design issues are described.

3. Virtually Separable Flash Storage System Design

3.1. System Architecture

The overall system architecture of Flash memory-based storage systems is shown in Figure 2. In the system, the Flash memory is connected to a host system through standard IO interface such as ATA, SATA or MMC/SD interface. The Device controller takes charge of the interfacing. FTL provides functionality in mapping management with address translation, garbage collection, and wear-leveling. Inside the Flash device, usually there are Flash memory chip array consisting of dozens of single functionality of flash chip for higher bandwidth.

Since Flash device is connected with standard interface, host system makes use of it as a traditional storage device by stacking standard storage software which is composed of low level device driver, block device layer, and file systems, for instance, in Linux Operating System. The file system and its related system modules generate IO requests with appropriate strategies. In the figure, file system writes user data while update the corresponding metadata. At the same time, journaling module generates and writes transactions that are logged to logically separated area. Journaling module is important file system integrity and essentially installed for most storage system.

3.2 Separable Flash

To separate physical area in accordance with logical separation, it is required to identify each logical area. In our design, we allocated ID to each logical area, and the ID is transferred to Flash device in conjunction with typical information for data transferring, i.e., logical address and size of the request. Since there is no command interface for transferring identifier for each command, we introduce new command interface between file system and low level drivers. For each read and write command from file system, or its related IO subroutine, the form of read/write command has information like following.

$$<command> = <ID, LBA, size>$$

Where, ID is identifier for logically separable region, LBA is the first Logical Block Address of corresponding request, and size is length of the request, respectively. For example, metadata, userdata, and journal data are assigned with different identifier, ID, to differentiate logical area, shown in the figure 3. In the figure, whenever file system makes requests for underlined storage device, it identifies the request type and assigns appropriate ID to the request. Then the request is passed to lower layer. The low layer, i.e., device driver and host controller, transfers the request by shaping command form for appropriate interface type. In the layer, requests can be re-scheduled or merged each other according the request scheduling algorithm. The re-scheduling is okay, however merging of requests can be a problem when requests having different ID are merged. In this case, we prohibit the merge operation, and the requests are transferred individually.

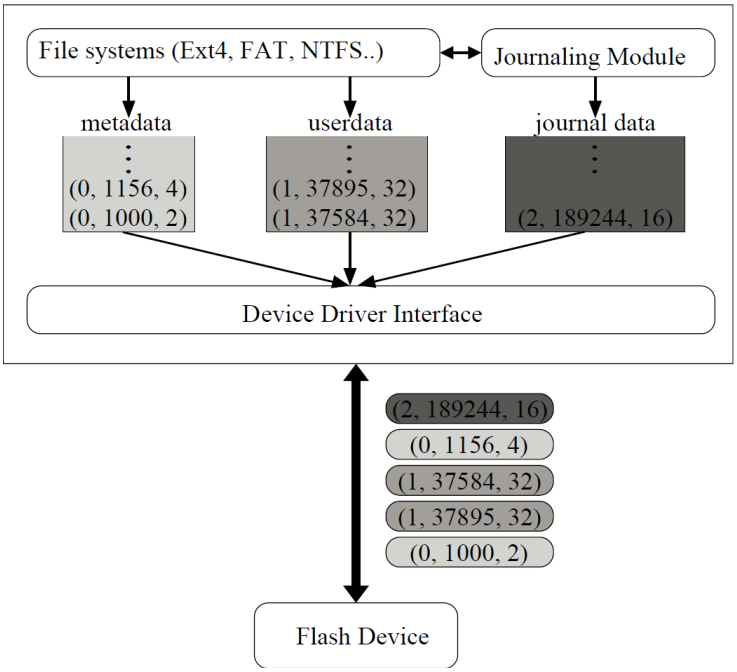


Figure 3. ID Classification and Request Generationss for Each Logically Separable File System Region

In the Figure 3, file system makes five requests for throughout the file system region, and passes theses to lower layer with file system's own strategies. The lower layer does re-scheduling and merging algorithm with the requests during its plug-in time slot. In this case, there is not merge among requests, but might exist some request re-scheduling according to the scheduling algorithm. The re-scheduling does not have an effect on our system any more, since the purpose of our system is not scheduling or reduction

The separable Block management within Flash device is described in Figure 4. As shown in the figure, there are Block buckets in accordance with logical region identifier, *i.e.*, ID. In each bucket, Blocks are connected virtually with doubly linked list. When write requests are arrived from host system, FTL checks the ID of the requests, assigns it to the corresponding bucket, and data to the available Pages of the Block in the bucket. If the possible area are all consumed for the Block and more Pages are required to complete the request, FTL allocates another Block from free Block list to the bucket and does the remaining writing job. After that, FTL updates mapping table.

Bucket management can be done as ad-hoc manner, which means that bucket can be added or deleted at runtime dynamically, not fixed statically. When Flash device gets request that has newly assigned ID by host system, FTL assigns new bucket for the new ID, allocates Block to the new bucket, and writes data to the Block. Likewise, if the bucket has no Block, the bucket is removed. The removal of Block from bucket is related with GC operation. When one Block is selected as a victim block in GC, the Block is removed from the assigned bucket, and moved to free Block list after GC. After GC operation, FTL checks whether the bucket has Blocks included in. If there is no Blocks, FTL removes the bucket and the corresponding ID is not used until requests having the ID is arrived.

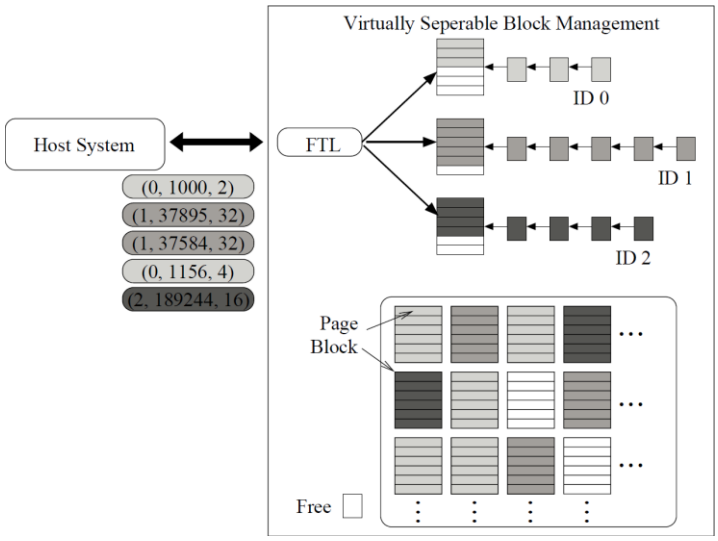


Figure 4. It describes Separable Block Management within Flash Device, where FTL Classifies and Assigns Requests to Corresponding Block Bucket

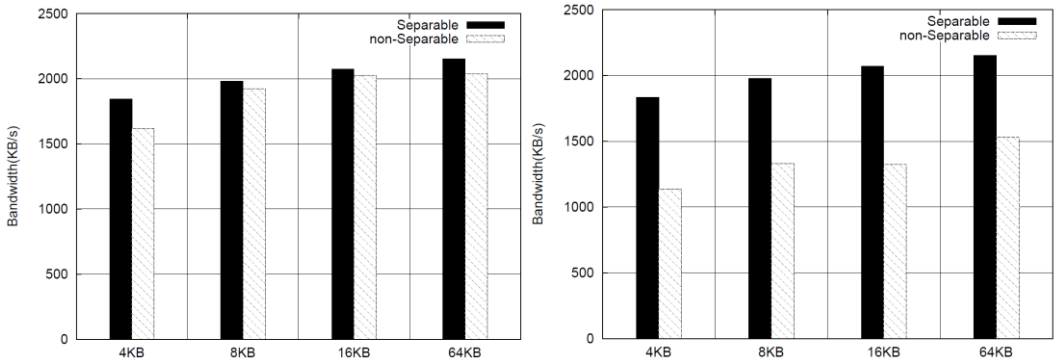
The separable Block management is independent on specific FTL mapping algorithm and GC algorithm, since it only related with virtual allocation and de-allocation of Block list with bucket structure. Therefore, separable Block management can be applied to any FTL with slight changes of Block management issue. In our development, the Block management is applied to simple Page-level mapping-based FTL, so mapping table exists which is apart from Block management information. Also GC is also Page-level mapping -based. In GC, the victim Block is selected from the all the Blocks, not from some bucket. Likewise, the virtually separable Block management can be also applied to Block-level or hybrid-level mapping-based FTL.

With the bucket and Block list, the virtually separable Block management is possible, and intended logical separation of file system can be done in Flash device. For instance, home data of file system, whose ID is 0, is stored with some physical flash Block consecutively, where these Blocks are grouped together. Also, other file system's data, such as journal data are separated each other. With the Block separation, more sequential writes are generated within Flash device, which leads that the Page validation and invalidated occurs much more largely. As a result, GC efficiency and write amplification is improved. The improvement of flash-internal operations leads device-outside bandwidth.

4. Evaluation

Since our proposal introduces new command interface between file system and flash device, and includes Block management issue of FTL which reside in flash-device-side, it is hard to implement with real flash device. Instead of that, we make use of simulation environment. In Linux system, the NAND flash-based simulation environment can be setup with MTD device layer. In the MTD device layer, Page-mapping-level FTL is first developed, actually which is previous work. Based on the Page-mapping-level FTL and MTD device layer, the proposed scheme is implemented. In our implementation, we only separate two file system's region, the one is file system's home data, and the other is journal data, to see the feasibility of the virtually separable Block management. Actually, separation of two region is

enough for file system's logical separation, since these two regions are representative file systems' IO operations.



(a) Sequential Write (b) Random Write

Figure 5. Experimental Results for IOzone Benchmark; The graphs plot bandwidth for sequential write and random write, respectively, as request size increases from 4KB to 64KB.

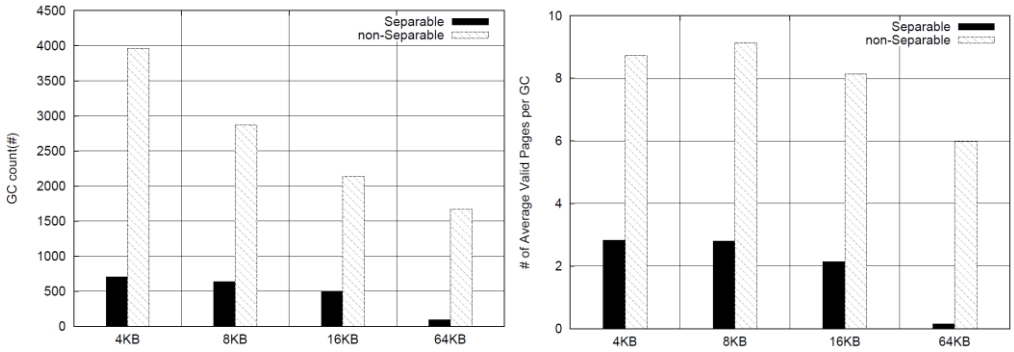
The implemented system was run on embedded computing system development board with real bare-NAND flash chip. The embedded board is composed of 1GHz ARM processor and 256MB main memory. The NAND flash device has 1GB, whose page size is 2KB and block size is 128KB. Typically, read and write time for one Page is 25us and 200us, respectively, and erase time for one Block is 2ms.

We have evaluated the proposed scheme with well-known file system benchmark, called IOzone [17]. The IOzone file system benchmark generates and measures bandwidth for file system operations such as read, write, re-read, re-write, and random write with file generation. In our experiments, IOzone generates 8MB files, and does file operations with write and random write for request size 4KB, 8KB, 16KB and 64KB.

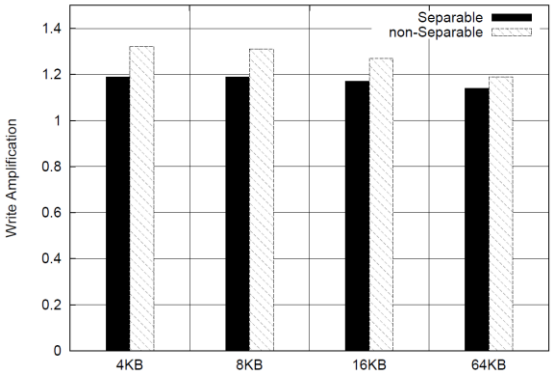
During the benchmark running, we mainly focus on estimating write requests rather than read requests, since the proposed scheme mainly affects write operations. Indeed, the system performance of Flash storage system is mainly dependent on the write request, as well. The bandwidth of write and random write operation is estimated as request size varies from 4KB to 64KB. The experimental results are shown in Figure 5(a) and 5(b) for sequential write and random write requests, respectively. For the sequential write requests, the proposed scheme gives slightly better bandwidth than legacy system. However, the proposed scheme outperforms legacy system for random write requests greatly. The separable Block management within Flash device makes it possible to allocate correlated data to Blocks which are virtually connected, while relatively uncorrelated data are separated. As a result, the random requests from host system stored within Flash device as if sequential write requests, which results in bandwidth improvement for random write requests. From the Figure 5(b), we identify that the bandwidth increases about 30%-40% in comparison with legacy system that does not separate between journal data and file system's home data.

In addition to the bandwidth, to give the insight that how the proposed invalidation scheme affects on the flash-internal usage, we have collected flash-internal statistical metrics, such as Number of GC (NGC), Number of Valid Page Copies per GC (NVPC), and Write Amplification (WA), respectively, during the IOzone benchmark running. The NGC represents how many NGC are performed, and NVPC represents how many live Pages exist for the GC victim Block, respectively. The figure 6(a) and 6(b) plots NGC and NVPC,

respectively. As shown in the figure, both of NGC and NVPC dramatically decrease for separable Block management system. The reason for reducing NGC is that many Blocks are totally invalidated as a whole when data are updated, due to the spatial localization for virtually separable Block area, and these Blocks are recycled to free Block without GC operations. Likewise, a victim Block has less valid pages, so the NVPC is also reduced in our system. The reduction of NGC and NVPC decreases flash-internal operations, as a result, computation time of flash-internal operations decreases.



(a) # of GC (B) # of Average Valid Page Copies per GC



(c) Write Amplification

Figure 6. Experimental Results for IOzone Benchmark; The graphs plot number of average valid page copy per GC, total GC count, and write amplification, respectively, as request size increases increases from 4KB to 64KB.

Lastly, WA is investigated, and the experimental results for WA are depicted in Figure 6(c). The WA represents the ratio between the amount of writes from host system and the amount of flash-internal writes. Since GC makes data moving of valid data for victim Block, there are always more real flash writes than host writes, which means that WA is greater than 1 and the Flash storage system is as good as WA is less. From the figure, we identify that the proposed scheme reduce WA about 10% for every request size, which is mainly from the reduction of NVPC.

In Summary, the reduction of NGC, NVPC, and WA results in improvement of flash-internal operations, such as latency, lifecycle of Block, so write throughput of flash storage system is improved.

5. Conclusion

Although NAND flash memory is becoming main storage resources for computing system, there are several physical limitations. The limitation of is covered by software layer called Flash Translation Layer (FTL). File systems can treat Flash device just like usual storage media with the logical block address supported by FTL, and FTL hides internal mapping information from host file systems.

In the view of file system, the mismatch between logical address and physical address makes unintentional performance degradation, as a result, traditional file system architecture and layout design is little effective for Flash-based storage. Flash device has different view of logical address and physical address, so the separation of file system's layout does not directly applied to the physical separation for Flash device. Therefore, file system's data are mixed in physical location for flash device.

In this paper, we propose virtually separable Block management scheme for Flash storage system by introducing new common command interface between file systems and underline flash device. In the proposed system, file system has identifier, i.e., ID, to identify data type, where the IDs are assigned for each type of data, and the file system makes read/write commands with the corresponding ID. In the flash devices, the flash Blocks having same ID are virtually connected, where the 'virtually' means that the Blocks are not necessarily consecutive in physical position. The experimental results show that the proposed scheme increases write throughput, as well as reduces flash-internal overhead by making separable Block management within Flash device.

Acknowledgements

This work was supported by Hankuk University of Foreign Studies Research Fund.

References

[1] A. Ban, "Flash file system optimized for page-mode flash technologies", U.S. Patent 5,937,425., Filed October 16 1997.
[2] Intel Corporation, "Understanding the flash translation layer(FTL) specification", http://developer.intel.com/.
[3] Webopedia, "What is solid state disk? - A Word Definition from the Webopedia Computer Dictionary" IT Business Edge, (**2012)**.
[4] JEDEC, "Embedded Multimedia Card Electrical Standard", (**2013)** September.
[5] Micron, Garbage Collection in Single-Level Cell NAND Flash Memory, Technical Note, TN-2960.
[6] M. K. McKusick, W. N. Joy, S. J. Leffler and R. S. Fabry, "A Fast File System for UNIX", ACM Transactions on Computer Systems, vol. 2, no. 3, (**1984**), pp. 181–197.
[7] M. C. Avantika Mathur and S. Bhattacharya, "The new ext4 filesystem: current status and future plans", In Proceedings of the Linux Symposium, (**2007**), pp. 21–33.
[8] Microsoft, "New Technology File System", NTFS Technical Reference, http://technet.microsoft.com/en-us/library/cc758691(WS.10).aspx
[9] T. M. Jones, "Anatomy of Linux journaling file systems", IBM Developer Works, (**2008**).
[10] 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, (**2002**) May, pp. 366-375.
[11] J. U. Kang, H. Jo, J. S. Kim and J. Lee, "A superblock-based flash translation layer for NAND Flash memory", Proceedings of the 6th ACM & IEEE International conference on Embedded Software, (**2006**) October.
[12] S. W. Lee, W. K. Choi, and D. J. Park, "FAST: An efficient flash translation layer for flash memory", Embedded and Ubiquitous Workshops, (**2006**) August, pp. 879–887.
[13] A. Gupta, Y. Kim and B. Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Pagelevel Address Mappings", Proceeding of the 14th international conference on Architectural support for programming languages and operating systems, (**2009**).

[14] D. Ma, J. Feng and G. Li, "LazyFTL: A Page-level Flash Translation Layer Optimized for NAND Flash Memory", Proceesing of the ACM SIGMOD, (**2011**).
[15] Y.-H. Chang, P.-L. Wu, T.-W. Kuo and S.-H. Hung, "An adaptive file-system-oriented FTL mechanism for flash-memory storage systems", ACM Transactions on Embedded Computing Systems, vol. 11, no. 1, (**2012**).
[16] Y. Lee, L. Barolli and S.-H. Lim, "Mapping granularity and performance tradeoffs for solid state drive. The Journal of Supercomputing, vol. 65, no. 2, (**2013**), pp. 507-523.
[17] B. Martin, "IOzone for filesystem performance benchmarking", Linux.com, (**2008**).

Author

Seung-Ho Lim, he received BS, MS, and Ph.D. degrees in the Division of Electrical Engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2001, 2003, and 2008, respectively. He worked in the memory division of Samsung Electronics Co. Ltd from 2008 to 2010, where he was involved in developing a high performance SSD (Solid State Disk) for server storage systems. He is currently a professor in the Department of Digital Information Engineering at Hankuk University of Foreign Studies. His research interests include operating systems, embedded systems, and flash storage systems..