IEICE Electronics Express

LETTER

IEICE Electronics Express, Vol.13, No.2, 1–10

Bitmap discard operation for the higher utilization of flash memory storage

Seung-Ho Lim[1a)] **and Woo Hyun Ahn**[2b)]

[1] *Division of Computer and Electronic Systems Engineering, Hankuk University of Foreign Studies, 89 Wangsan-ri, Mohyeon-Myeon, Yongin-si, Korea*

[2] *Department of Computer Science, Kwangwoon University, 447–1 Wolgye-Dong, Nowon-Gu, Seoul, Korea*

a) *slim@hufs.ac.kr*

b) *whahn@kw.ac.kr, Corresponding Author*

Abstract: NAND Flash memory storage is widely used in computing systems. In General, there exists mismatch between logical address and physical address in Flash storage, and these address translations are managed by Flash Translation Layer (FTL). Due to its management, logically invalid data is considered as physically valid at some parts in Flash device, which causes additional overhead. The physically valid area being logically invalid area can be invalidated by TRIM or discard command, however, too many discard commands degrade throughput. In this paper, we propose a bitmap-based discard operation which can decrease the number of runtime discard commands. According to the proposed scheme, hundreds of separated region can be discarded all at one bitmap-based discard command.

Keywords: NAND Flash, FTL, discard, bitmap

Classification: Storage technology

References

[1] A. Ban: U.S. Patent 5,937,425 (1997).

[2] F. Shu and N. Obr: Data set management commands proposal for ATA/ATAPI Command Set-3. T13 Technical Committee, United States: At Attachment (2013).

[3] A. Mathur, M. Cao and S. Bhattacharya: Proc. of the Linux Symposium (2007) 21.

[4] S.-H. Lim: IEICE Electron. Express **10** (2013) 20130339. DOI:10.1587/elex.10.20130339

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse and R. Panigraphy: USENIX ATC (2008) 57.

[6] M. Saxena and M. M. Swift: USENIX ATC (2010) 187.

[7] J. Kim, H. Kim, S. Lee and Y. Won: Intl. Workshop on SSPS (2010) 7.

[8] C. Hyun, J. Choi, D. Lee and S. H. Noh: ACM SOSP (2011) 11.16.

[9] N. Jeremic, G. Muhl, A. Busse and J. Richling: ACM SAC (2012) 1753.

[10] H. Son, Y. Lee, Y. Kim and J.-S. Kim: KIISE TCP **10** (2015) 52.

[11] Memory Technology Devices, Memory Technology Device Overview (2012) http://www.linux-mtd.infradead.org/.

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

[12] Samsung Electronics: Nand flash-memory datasheet (2011) http://www.datasheetcatalog.com/samsungelectronic/41/.
[13] J. Katcher: PostMark: A New File System Benchmark, Technical Report TR3022, Network Appliance Inc., Oct. (1997).

1 Introduction

NAND Flash-based devices, such as Solid State Drives (SSDs) and embedded Multi-Media Controller (eMMC) cards, have reached the mainstream in storage solutions. Inside the Flash device, due to its physical limitations, there is additional internal overhead, such as mapping management between logical address and physical address and Garbage Collection (GC). The Flash Translation Layer (FTL) [1] is in charge of managing these issues. FTL is key software layer for NAND flash memory, and most of NAND flash devices include FTL inside their devices.

FTL provides logical address to host system and hides physical address, whereas, host system has no idea about the real physical location of data. At the view of Flash memory, some areas within it are considered as valid even if the areas do not contain valid data any more. Even the host system thought that a data is considered as invalid, FTL recognize the data is being valid. For instance, when a file is deleted, file system deletes it by just deleting the metadata of the file, leaving data area of the file alive. In this case, the data is considered as valid within flash memory until the region is rewritten by file system. The physical area of logically invalidated area can be invalidated by TRIM or discard command [2]. However, the problem is that the discard command itself is another kind of overhead at the aspect of IO subsystem and device driver, since the command also occupies bandwidth, as well as it is sync operation. For instance in real systems, if Ext4 file system [3] is set to support discard operation in Linux, a lot of discard commands are generated between ongoing read and write requests, which, in turn, degrade bandwidth at that moment.

In this paper, we propose a bitmap-based discard operation for higher NAND flash storage utilization by reducing the number of runtime discard commands. In the bitmap-based scheme, the discarded area is represented as a bitmap. In the bitmap, one bit represents one page in the Flash storage, which is similar to data bitmap in Ext4 file system layout. The advantages of bitmap-based scheme are two; the first is that it can reduce discard commands dramatically by using bitmap information. The second is that several distant discard regions can be invalidated with one discard command. According to the proposed scheme, hundreds of separated region can be discarded all at one discard command.

2 Background

In this section, the background information for Flash memory is described, and the standard command structure of ATA TRIM command, i.e., discard command is explained.

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

IEICE Electronics Express, Vol.13, No.2, 1–10

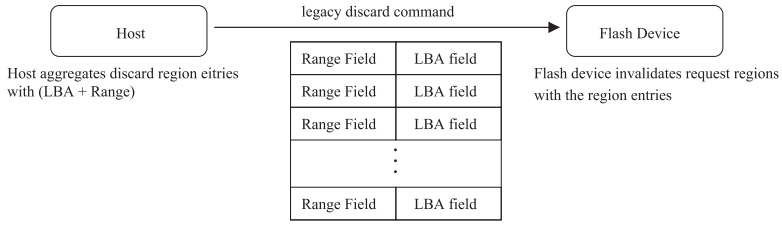


Fig. 1. The legacy discard command operations.

NAND flash memory is array of memory cells, and the bundle of memory cells called *Page* is the unit of read or program (write). Likewise, bundle of several pages called *Block* is the unit of erase. The read and program command are related with data transfer between host and Flash device, while, the erase command has no data transfer between host and Flash device. Typically, the size of one Page is 4 KB and doubles as manufacturing process advances and a Block is composed of 64 or 128 Pages. The erase operation for each cell should be preceded by program operation. The role of FTL is address translation between logical address of file system and physical address of flash memory itself. FTL performs out-of-place updates which in turn help to hide the erase operation. If the number of free pages is insufficient for write operations, free pages should be made by 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. GC makes available free region for later write requests. During GC operation, valid data should be copied from victim block to available other region. The GC efficiency is getting higher as the number of data to be copied is getting smaller. The detailed management of FTL is described in [4].

According to the specification of recently released ATA 13 [2], discard command consists of requests of invalided logical region specified by LBA and range, which means that the logical region from LBA to (LBA+range) is requested to be invalidated so the requested area can be deleted physically. According to the specification, one or more (LBA+range) discarded range can be aggregated in one discard command, as shown in the Fig. 1. Typically one sector-command can have up to 64 discard regions. When the discard command is transferred from host to flash device, FTL updates its mapping table with the address received from host. The procedure of invalidation for the requested region within FTL is as follows. At first, FTL finds its logical addresses ranging from LBA to (LBA+range), and sets the corresponding physical address of the requested logical address as invalid, i.e., 0xFFFFFFFF for example.

Although the number of entries to be invalidated for one discard command can be more than one, as the specification notes, the aggregation of the regions would not be properly done as just that the specification ordered in real system implementation such as Linux device driver, which is maybe from the command generation and timing issues between file system operations and block layer operations, as well as the notation format of discard region. For example, if there are a lot of distinct discard regions having single block, is makes many entries of discard command. It in turn generates many discard commands. Moreover, discard

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

IEICE Electronics Express, Vol.13, No.2, 1–10

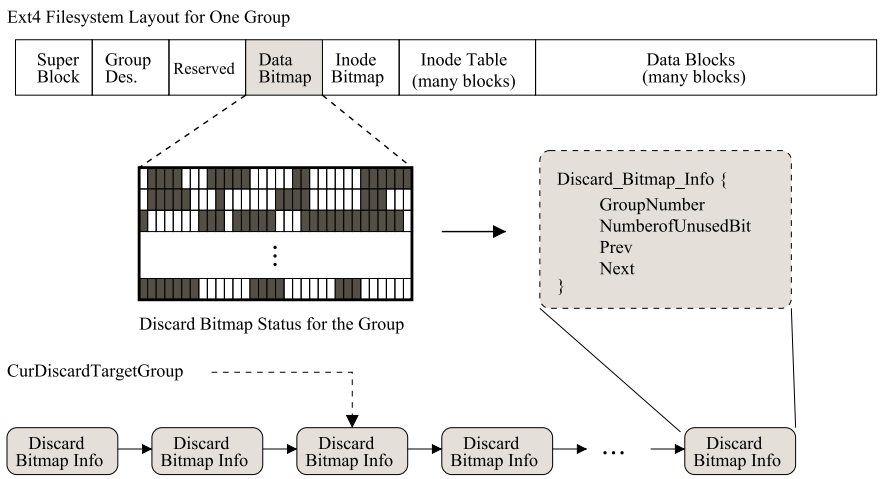


Fig. 2. Discard bitmap structure and its management.

command itself is sync operation, so it gives more overhead of ongoing data transfer commands. The bitmap-based discard region expression is suitable for aggregation of distinct discard regions, so the discard command format is re-designed as a bitmap manner in this paper to aggregate distinct discard regions efficiently. Although there are several previous works for TRIM technique to support operating system levels or flash device levels [6, 7, 8, 9, 10], there were little work for re-designing TRIM, i.e., discard, request format for higher discard usage. The proposed bitmap-based discard command scheme is enhanced version of original discard command specification in the aspect of how to aggregate the dispersed discard regions.

3 Bitmap-based discard management

To reduce generating discard commands during run-time, we consider bitmap-based expression for the discarded region and transferring the discard-bitmap information within discard commands. The overall bitmap-based discard management is described in Fig. 2. To show its feasibility, the bitmap-discard management system is designed with the Ext4 file system [3], however, other file systems also can be applied similarly, since key point is how to generate discard bitmap information to aggregate the dispersed discard regions.

When Ext4 file system is formatted, the layout of file system is divided by several groups according to the capacity of the file system, and each group has layout shown in the Fig. 2. In the Ext4 file system group layout, there exists bitmap information for data block used, called *Data Bitmap*. In the Data bitmap, one bit is set to 1 when the corresponding data block is used, or vice versa. For simple description, let us assume that a file system block is match with a Flash memory page. The data bitmap can be used for discarding of the unused blocks, i.e., page, with opposite values, which means that every unused block in the file system can be invalidated in Flash devices. Thus, in the bitmap-based discard system, we generate discard information called *discard bitmap* with opposite values of data bitmap. For each file system group, there exist one discard bitmap in corresponding with one data bitmap. Thus, if the size of discard bitmap is 4 KB, it represents 32768 pages

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

IEICE Electronics Express, Vol.13, No.2, 1–10

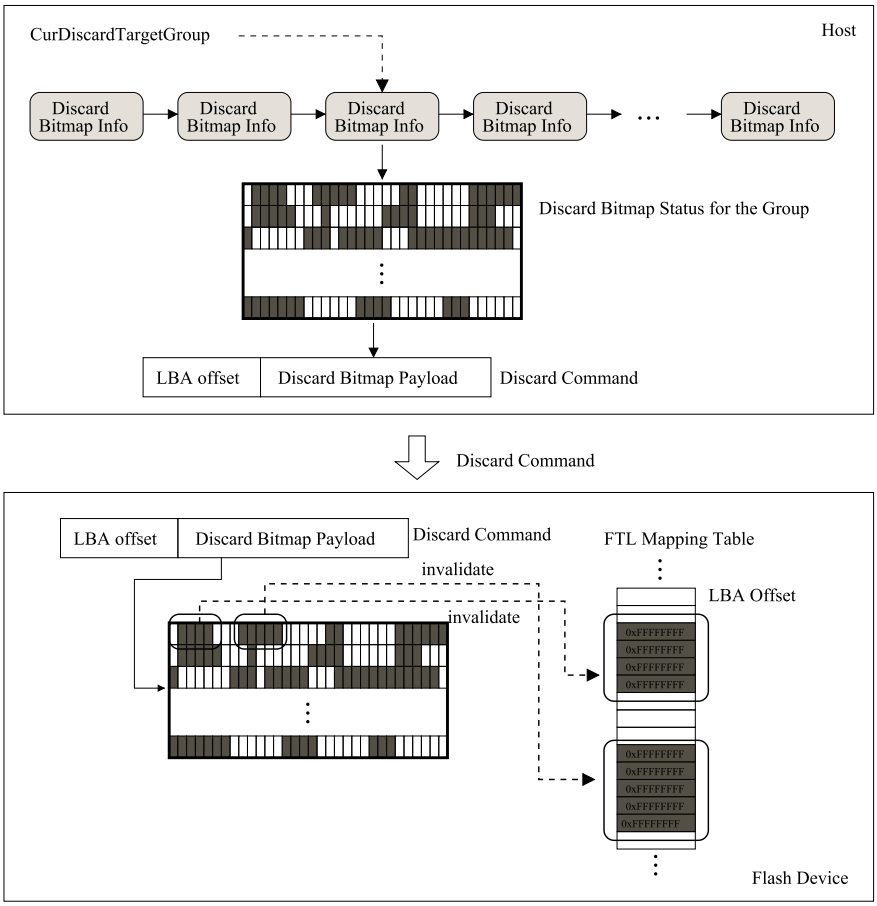


Fig. 3. Bitmap-based discard command transfer and invalidation of the discard area in FTL.

to be checked as discarded or not. Ideally, maximum 32768 logical pages can be invalidated with one discard command, and hundreds of dispersed discarded regions can be aggregated in one discard bitmap in average.

The discard bitmap information is managed by *Discard_Bitmap_Info* structure, which contains the *GroupNumber*, *NumberofUnusedBit*, *Prev*, and *Next* pointer. The GroupNumber and NumberofUnusedBit represent group number of the discard bitmap and number of unused blocks in this group, respectively. The Prev and Next pointers are used for linked list management of *Discard_Bitmap_Info*. When a file is deleted, the corresponding blocks is cleared in data bitmap. Likewise, the corresponding bit is set to 1 in discard bitmap and the corresponding *Discard_Bitmap_Info* is updated. If the group is set to have checked bits, the *Discard_Bitmap_Info* is added at the end of the linked list of existing *Discard_Bitmap_Info* list. The linked list maintains possible discard bitmaps for discard command operations. In general in Ext4, discarded regions are generated after a file is deleted, and these regions are sent to flash device during the Ext4 journal commit operation as bitmap-discard commands. During the journal commit operation, discard bitmap information having discard regions is made to discard command and issued to lower layer. When discard command is issued, the discard bitmap information is encapsulation in the command and transferred from host to Flash device. Since the current targeted discard bitmap is pointed by *CurDiscardTargetGroup*, the *CurDiscard-*

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

IEICE Electronics Express, Vol.13, No.2, 1–10

Table I. File systems' IO configurations with Postmark

	Conf. 1	Conf. 2	Conf. 3	Conf. 4
Base files	10000	10000	5000	2000
# of Transactions	1000000	200000	10000	20000
File Size	10 KB–50 KB	50 K–100 KB	100 KB–512 KB	512 KB–1 MB
User Block Size	Read = 512 bytes, Write = 512 bytes			
Biases	Read/Append = 5, Create/Delete = 5			

TargetGroup points to next discard_bitmap_Info in the linked list, which will be transferred in next round of discard command after the current targeted discard bitmap is transferred.

The bitmap-based discard command transferring and the related invalidation for the requested discard region within Flash device is described in Fig. 3. As shown in the figure, the bitmap-based discard command contains payload data which represents discard bitmap and its starting logical address. When flash device receives the command having payload data, it does invalidation jobs with the requested invalidation data. The invalidation procedure mainly can be done by only updating the mapping table of FTL within Flash device otherwise command has additional issue indicated. Since almost Flash device manages FTL's mapping table in DRAM memory inside it, the invalidation of discard region can be done by just memory operations within the Flash device, but is not related with Flash operations such as reading, programming or erasing some pages or blocks. Thus, for almost all discard commands, FTL just invalidates by setting invalid mark, such as 0xFFFFFFFF, to the map of requested region, which reside in main memory.

Updating the mapping table may differ from FTLs since FTLs have different mapping management algorithm each other. However, main theme is similar, so the invalidation scheme can be applied in similar manner. In our system, we consider FTL having page-level mapping table management [4], since our implementation is based on page-level mapping management FTL. Generally, mapping table for page-level FTL is a large index-based table, where each entry of the table represents physical address of the corresponding logical address. For instance, if the size of page is 4 KB, each entry of the mapping table indicates 4 KB physical page of corresponding logical address. Therefore the mapping table can be described at the right bottom in the Fig. 3, in which each entry of the mapping table contains physical address of the corresponding logical index value. With this mapping table, the invalidating algorithm is as follows. FTL identifies discarded logical address along with the LBA offset and discard bitmap information by checking bit by bit value. In the discard-bitmap information, one bit represents one logical page address, so if a bit is checked, i.e., set to 1, the corresponding logical address is willing to be invalidated by the host. If a bit is set to 1, FTL finds the table entry with the logical address, and updates its entry values as invalid by setting 0xFFFFFFFF. As shown in the Fig. 3, the logical address in the page-level mapping table is invalidated for each set bit. If we look into the discard bitmap information in the figure, the first LBA range from bit 2 to bit 5 is requested to be

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

invalid, so the corresponding table entries are invalid by setting invalidation value. The next three bits are not requested, so the next three table entries are not invalidated. After that, next five consecutive LBA range is requested to be invalid, so the corresponding table entries are set to be invalidated, and so on. As a result, the invalidated physical regions are considered as that these do not have valid data, so these areas will be used for possible free area at later use. Also, the area will not be copied as valid during GC.

4 Evaluation

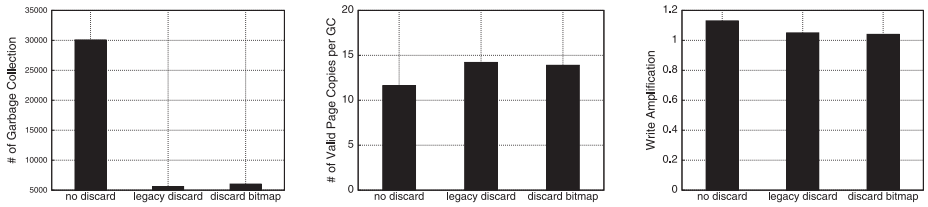
The bitmap-based discard management scheme is simulated in Flash device simulator with FTL and NAND simulated device driver in Linux MTD layer [11]. For the evaluation, we have setup simulation environment with simple embedded system evaluation board with NAND simulator whose physical characteristics is referenced from Samsung datasheet [12]. We have compared the impact of bitmap discard operations with legacy discard command and non-discard system by generating file system IO operations with create/read/write/delete, as IO request distribution varies with Postmark benchmark [13], which is used for 4 different configurations, as described in Table I. In the workload generation, the file size increases and number of IOs decreases, as number of configuration increases. The ranges of file size and number of IOs can represent web or email server, documentation, common works, and multimedia workloads, for conf. 1, 2, 3, and 4, respectively.

For each benchmark configuration, we have three types of experiments with regards to no discard, legacy discard commands, and bitmap discard commands. The no discard means discard commands are not used, and the legacy discard means discard commands are generated as aggregated manner up to 64, as indicated in the specification. The bitmap discard is the scheme proposed in this paper. During the experiments, flash internal performance metrics such as GC count, the number of page copies per GC, and write amplification are estimated, and the results are depicted in Fig. 4. The GC count and the number of page copies per GC are the meaning as the names represent, and write amplification represents the ratio between the amount of logical writes and physical writes, which implies how much the internal overhead is generated.

At first, the number of GC occurred during benchmark tracing is dramatically reduced when for both legacy discard and discard bitmap commands are used, as shown in Fig. 4(a). It implies that discard command itself makes more free flash blocks at running time than no discard, by invalidating many of whole the blocks. For the number of page copies per GC, we can identify that no discard gives best result than others. It seems like discard commands give more overhead than no discard for this metric. However, it is the result as we expected to have more page copies overhead for discard commands. Since GC is rarely occurred for both of legacy discard and bitmap discard, these may have more possibilities to have more valid pages than no discard when GC is performed. Although the number of valid page copies per GC is larger, overall valid page copies during system runtime is less for discard commands, which can be identified by result for write amplification.

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

IEICE Electronics Express, Vol.13, No.2, 1–10



(a) # of Garbage Collection (b) # of Copies per GC (c) Write Amplification

Fig. 4. Flash internal performance metrics and results

Since write amplification represents internal overhead factor by showing the ratio between the amount of host's logical request and the amount of internal writes, it is better to give lower value. As shown in the Fig. 4(c), discard command schemes give lower write amplification than no discard, which means discard command schemes give lower internal overhead than no discard.

If we compare two discard schemes for the metrics of the number of valid page copies per GC and write amplification, these give similar results. It is also reasonable results as we expected. Since the overall requested discard ranges from benchmarks might be similar for both discard schemes with the same benchmark traces, so the invaliding regions within Flash device are similar. Among the results, the bitmap-based discard scheme has slightly worse than that of legacy discard scheme for write amplification factor, even though it is depreciable difference. In our opinion, the results are from the differences for the amount of invalid area per discard command and some timing issues. That is, the bitmap discard has larger discarded regions than legacy discard for each discard command, however, there is some timing issue for generating discard command and transferring from host to Flash device. In order to include more discard regions in one command for the bitmap-based discard scheme, it is required more gap between commands than that of legacy scheme. As a result, write amplification factor would decrease. However, the decrease is so much since timing delay is so much to give critical to give crucial degrading of write amplification.

To show the advantages of bitmap discard over legacy discard, we counted the number of discard commands during benchmark tracing. The number of discard commands generated during each configuration of benchmark is depicted in Fig. 5. Also, the estimated number of discard blocks, average number of discard ranges per discard command, and average number of discard blocks per command are summarized in Table II. As shown in the figure, the bitmap discard scheme could reduce the generation of discard commands largely for Conf. 1 and 2 of Postmark benchmark traces, and the discard commands are also reduced for Conf. 3 about half, while bitmap discard generated slightly more commands than legacy discard for Conf. 4. Since there are a lot of dense file operations for small and mid-sized file operations, the bitmap discard can aggregate dispersed discard regions much greatly than legacy discard, which results in much reduced discard commands generation as shown in the figure for Conf. 1, 2, and 3. On the contrary, as the discard range for each discarded region increase, the ability of aggregation for bitmap discard is getting lower, which results in more command generations, however the gap between these two is not significant and tolerable in comparison

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

IEICE Electronics Express, Vol.13, No.2, 1–10

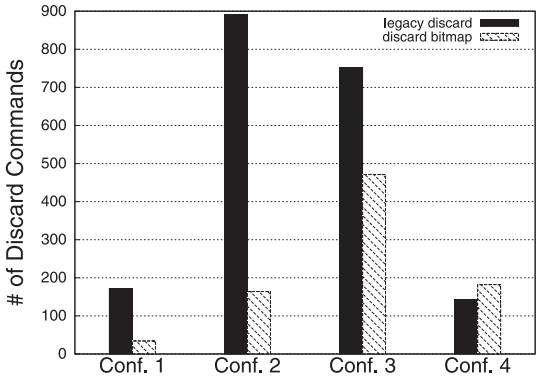


Fig. 5. The estimated number of discard commands comparison between legacy discard and bitmap discard during each Postmark benchmark trace.

Table II. The estimated number of discard blocks, average number of discard ranges per discard command, and average number of discard blocks per command are summarized.

	Total # of Dis. Blocks		Avg. # of Dis. Ranges per Cmd.		Avg. # of Dis. Blocks per Cmd.	
	Legacy	Bitmap	Legacy	Bitmap	Legacy	Bitmap
Conf. 1	385507	364710	64	322	2254	10727
Conf. 2	1429324	1433431	64	347	2320	12625
Conf. 3	3202362	3227040	64	102	6202	9966
Conf. 4	1985790	1940750	64	49	9379	6971

with the great reduction of former results. As s summary, bitmap discard can aggregate more dispersed discard regions than legacy discard, which would result in throughput increase. Although the overall throughput for normal read and write operations are not experimented, we could guess that the throughput increase in proportional to decrease of the number of ongoing discard commands. For the future work, the throughput gains with bitmap-discard command will be investigated.

5 Conclusions

When NAND Flash memory is used for storage device, there is mismatch between logical address and physical address, even at some situation, logically invalid data is considered as physically valid in Flash device. The physical valid area and logical invalid area can be invalidated by TRIM or discard command, however, too many discard command itself degrades write throughput. We propose a bitmap-based discard operation for higher NAND flash storage utilization, which can decrease the number of runtime discard commands. In the bitmap-based scheme, the discarded area is represented as a bitmap and transferred via discard command by modifying the interface specification so that it could aggregate a lot of LBA Range Entries to one command. Thus, one discard command can invalid a lot of scattered discarded region. At the experimental results, we showed that the

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016

proposed bitmap-based discard operations could decrease the number of ongoing discard commands over the legacy discard command operation. The throughput gains with bitmap-discard command will be investigated for the further work.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2015R1A5A7036384).

DOI: 10.1587/elex.12.20150976
Received November 17, 2015
Accepted December 8, 2015
Publicized December 22, 2015
Copyedited January 25, 2016