International Journal of Hybrid Information Technology
Vol. 9, No.9 (2016), pp. 123-134
http://dx.doi.org/10.14257/ijhit.2016.9.9.13

Investigating of Dynamic Interval of Journaling for more Lurked Discard Regions for Flash Memory Storage

Seung-Ho Lim and Ki-Jin Kim

Division of Computer and Electronic Systems Engineering
Hankuk University of Foreign Studies
slim@hufs.ac.kr, do131104@naver.com

Abstract

In modern storage system, most file system uses journaling operation to preserve data in-safe and recover fast from power loss. However, journaling operations generate excessive IO requests and degrades IO throughput. In addition to that, it also makes a lot of possible-discard regions that could be invalidated physically through discard command. However, due to it short period of time between journal transactions, there occur a lot of tiny lurked discard regions which give much overheads. In this paper, we identify that there exist many tiny lurked discard regions between successive journaling commit operations, and investigate a dynamic interval controlling scheme of journal transactions commit to reduce tiny lurked discard regions, as well as enlarge IO size of upcoming commit transaction. With the help or dynamic controlling of transaction interval, overall IO bandwidth can be enhanced.

Keywords*: NAND Flash Memory, Discard, Trim, Journal, Transaction Interval*

1. Introduction

Recently, the capacity of NAND flash memory chips has become large enough for them to be used as a part of the main storage medium of electronic devices, and this size will continue to increase quickly. The NAND Flash-based storage devices, such as Solid State Drives (SSDs) and embedded Multi-Media Controller (eMMC) cards, have reached the mainstream market as storage solutions [2]. SSDs are rapidly taking the place of traditional mechanical hard drives since they support the same interfaces at higher layers, and eMMC has also reached wide adoption in mobile systems.

Due to the limits of physical characteristics of Flash memory, there are some mismatches between host system and Flash devices. At the view of Flash memory, some area of Flash memory is considered as valid, even if the area does not contain valid data since Flash Translation Layer (FTL) provides logical address to host system and hides physical address, whereas, host system has no idea about the real physical location of data. Even the host system thought that a data is considered as invalid, FTL might recognize the data is being valid. To compensate this mismatch and corresponding overhead, host command, called TRIM or discard[1][3] was proposed as a part of command standards. According to the specification and device driver, discard command consists of LBA and its range, which means that logical region from LBA to (LBA+range) is invalid so the area can be deleted physically. At the aspect of Flash device and its internal usage, GC efficiency is getting higher as physically invalidated area is getting greater. Thus, it is crucial to find out how much discard regions among file system's IO operations. However, too many discard commands degrade overall IO bandwidth since it also occupies bandwidth at the command levels. Moreover, the discard command is used as synchronized manner, which gives more degradation of Bandwidth.

ISSN: 1738-9968 IJHIT

Not only the explicit deleting regions from the user's notification such as file's deletion, but also, there are a lot implicit lurked regions that could be deleted logically as well as physically with the file system area. Specifically, we focus on the file system's journal operations and its generating lurked discard regions. In modern storage system, most file system uses journaling operations to preserve data in-safe and recover fast from power loss. However, journaling operations generate excessive IO requests and degrades IO throughput. In addition to that, it also makes a lot of possible-discard regions that could be invalidated physically through discard command. We call these regions as lurked discard regions since these regions cannot be identified by file system explicitly.

Since file system has no idea about that these can be treated as discard region, these cannot be marked as invalidated physically unless otherwise indicated by others. There is previous work that could identify the lurked discard region during journaling transactions, and eliminate those by generating discard commands [4]. However, due to it short period of time between journal transactions, there occur a lot of tiny lurked discard regions which give much overheads. Since journaling commit operations are one of the most frequent write operations in file system's IO operations, a lot of lurked discard regions are generated among consecutive commit requests. Also, too many journal transactions reduce overall IO bandwidth since it generates many short IO requests.

In this paper, we identify that there exist many lurked discard regions between successive journaling commit operations, and propose a dynamic interval controlling scheme of journal transactions commit to reduce tiny lurked discard regions, as well as enlarge IO size of upcoming commit transaction. In the conventional journaling transactions in the Linux Operating System, the journal interval is fixed to five seconds as default. Although the interval is configured with configuration parameter, the configured interval time is not varied until re-configured. The dynamic interval controlling scheme, developed in this paper, changes journal transactions interval dynamically during runtime by monitoring the size of incoming transaction size and number of lurked discard regions to adjust number of discard commands as well as size of upcoming transaction. With the help or dynamic controlling of transaction interval, overall IO bandwidth can be enhanced.

The remainder of this paper is organized as follows. In Section 2, background and related work is described. The dynamic interval controlling scheme of journal transactions are described in detail in Section 3, and its performance evaluation and comparison are described in Section 4. Section 5 concludes this paper.

2. Background and Related Work

In this section, basic characteristics of NAND flash memory is described, and accordingly, the details of discard command are described. Most of the descriptions are from our previous works [4,5].

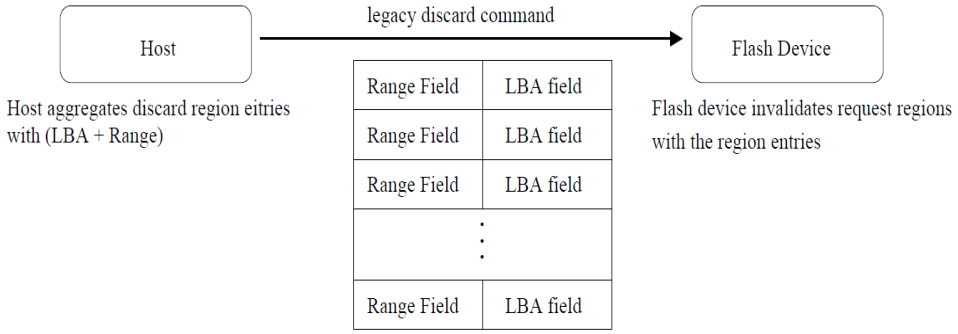

Figure 1. Discard Command Format

NAND flash memory is array of memory cells, in which one cell stores 0 or 1. The bundle of memory cells called 'Page' is the unit of read or program. Likewise, bundle of several 'Page's 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 4KB 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 mismatch between program and erase operation makes addition overhead for Flash memory, such as mapping management between logical address and physical address, and GC (Garbage Collection) [6]. The Flash Translation Layer (FTL) [7][8] is in charge of managing these issues. FTL is key software layer for NAND flash memory, and most of NAND flash devices include it inside their devices. 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. 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. There are two major different points between two storage media of NAND flash-based and traditional rotational-based; the one is there is additional internal overhead for Flash device, such as mapping management between logical address and physical by special software layer called Flash Translation Layer(FTL), and Garbage Collection (GC) processes during runtime writes. The Second difference is that NAND flash-based storage media does not dependent on mechanical aspects any more. The GC efficiency is getting higher as the number of data to be copied is getting smaller.

Inside the Flash device, 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. At the view of Flash memory, some area of Flash memory is considered as valid, even if the area does not contain valid data. FTL provides logical address to host system and hides physical address, whereas, host system has no idea about the real physical location of data. Even the host system thought that a data is considered as invalid, FTL might recognize the data is being valid, which is due to the file system's metadata operation. For example, when a file is deleted by delete operation from user, 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 [3]. According to the specification of recently released ATA 13[3], 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 Figure 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.

3. Controlling of Dynamic Transaction Interval based on the Discards

3.1. Analysis of Lurked Discards In Journal Transactions

In general operating systems, transaction management is done within filesystem, and most of filesystems use form-of-redo logging [10] for file system's atomicity. In our previous work, we identified the lurked discard regions among transaction commits of journaling file system, and proposes invalidation scheme of the lurked discard regions [4]. According to the previous work, there are a lot of multiple copies during the successive transaction commits. As referred the previous work [], the lurked discard regions are made as follows. While committing transactions, other transaction writes are buffered together in a compound transaction. If the new transaction tries to write to the same buffered block, which is being committed, then a new copy of the block is made and committed with new transaction. In this case, the previous committed block is useless since new version is committed. We call those regions as lurked discard regions.

An instance of lurked discard regions is described in Figure 2. In the Figure, data buffers of block 51 and 52 are the metadata buffers of two consecutive journal transactions of (n-1)th and (n)th, and 51, 52,and 55 are the metata buffers of two consecutive transactions of (n)th and (n+1)th. The consecutive updates of same metadata can be generated frequently due to the frequent updates of the corresponding file. As a result, both buffers are included in both of the transactions. The former logical block number of buffers 51 and 52 are block 1 and 2 in (n-1)th transaction, and the latter logical block numbers of those are allocated to block 5 and 6, respectively in (n)th transaction. Accordingly, the former logical block number of buffers 51, 52, and 55 are block 5, 6, and 3, the latter logical block number s of those are allocated to block 8, 9, and 10 in (n+1)th transaction. It leads that the former committed blocks, 1, 2, 3, 5, and 6 which are former versions of blocks, are logically useless anymore.

To invalidate multiple copies logically, two kinds of list structures, valid list and discard list are used. The valid list manages the list for the blocks that have up-to-date data of journal region, and discard list maintains list of lurked discard regions of the ongoing committed transaction. The blocks that are contained in the discard list are made up to the discard region in a discard command, and the discard command is issued from host system to flash device after the commit of the transaction is completed. For every transaction commit, the discard commands are generated.

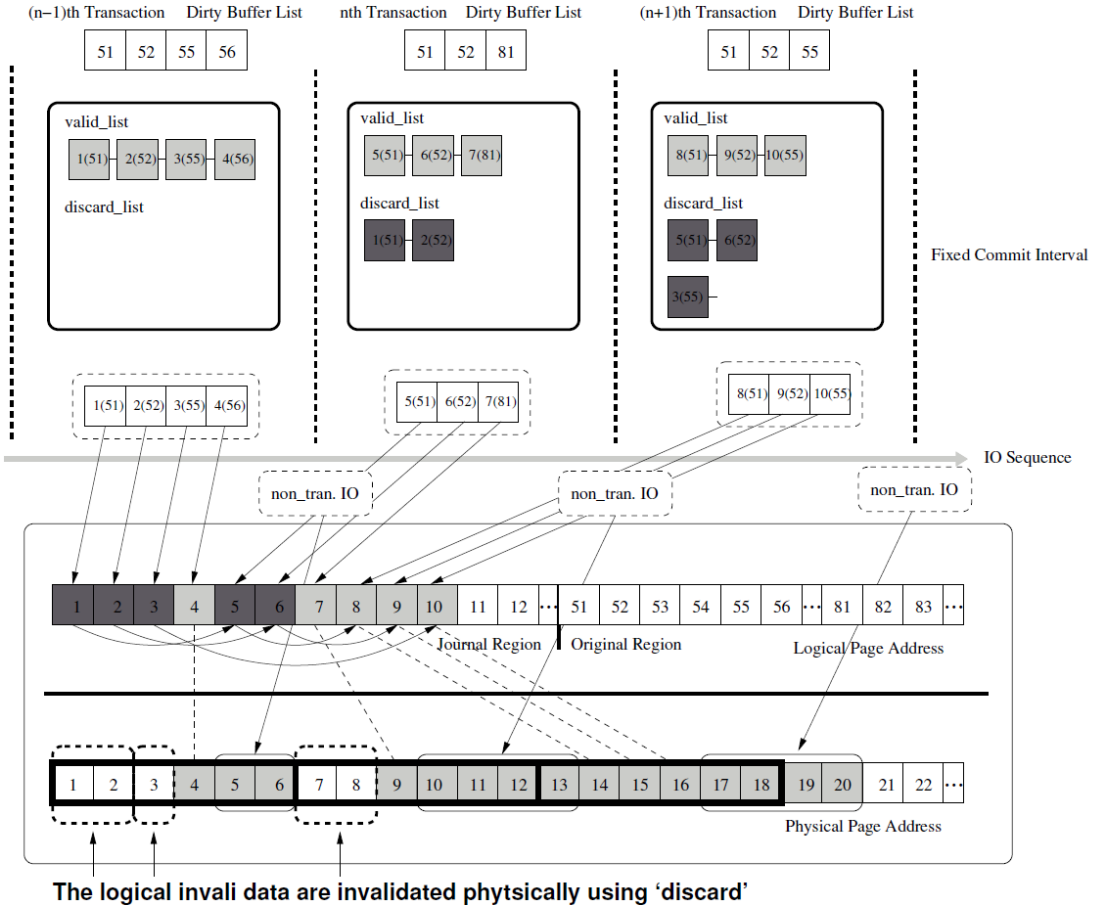

Figure 2. Management Scheme of Lurked Discard Region in File system Journal

3.2. Dynamic Transaction Interval Control Scheme

The discarding of lurked discard regions for journal transactions is beneficial to flash internal usage, and also as a result, beneficial to overall IO bandwidth. However, too many discard commands make adverse effects on IO bandwidth. The number of lurked discard regions for each transaction commit is deeply related to the transaction commit interval. The commit interval is also related to the commit IO size as well. If the transaction commit interval is short, the number of lurked discard regions is diminished, however, the length of commit IO is also diminished due to lack of time for gathering blocks for journal logging. As a result, short interval generates too many short commit IO. On the contrary, while long commit interval can generated large request for transaction

commit, it degrades the abibility of data integraty. Thus, proper commit interval should be decided, or other commit schcme should be considered.

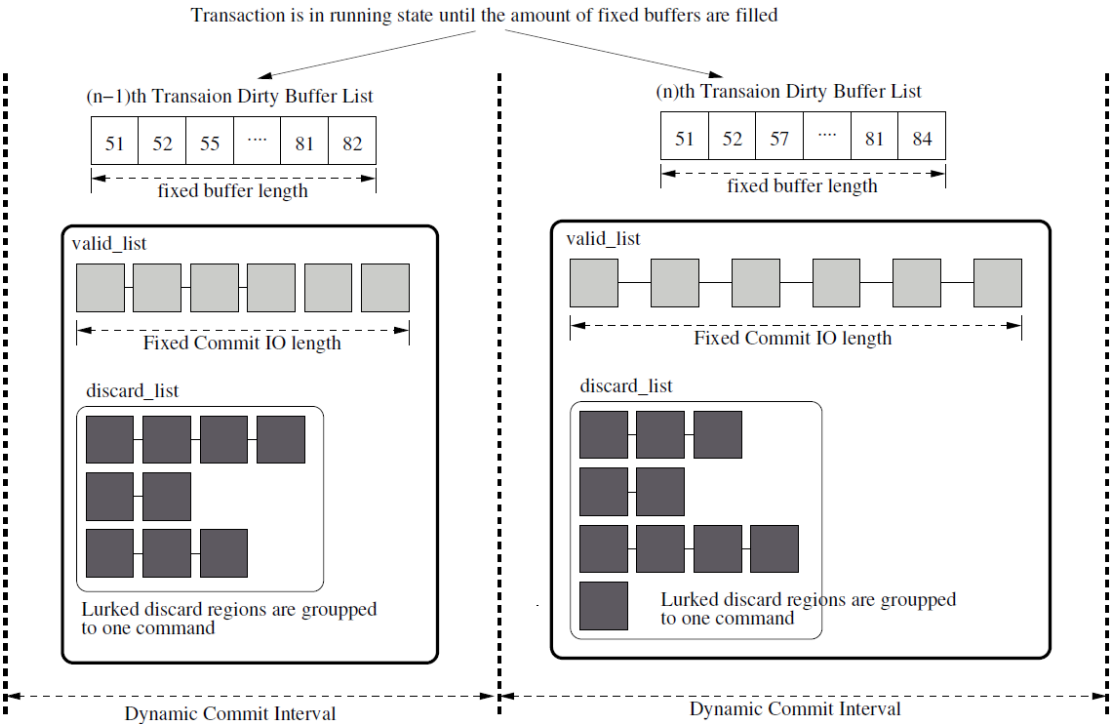

Figure 3. Dynamic Commit Interval Control and its Corresponding Commit IO And Discard Command

Instead of use of commit with fixed interval time, we develop dynamic controlling scheme of transaction commit interval based on the the amount of blocks for each transactions to be committed. As shown in the Figure 5, each tranaction is in running state to gather fixed amount of updated buffers, not fixed amount of time. The buffer size is fixed with pre-defined length for buffering the update blocks during transaction running state. After the the buffer is filled with update blocks in th transaction running, the transaction state is moved to commit states. The behavior of trsnactions comit state is same as that of fixed commit interval scheme, that is, it checks the valid list of the transactions by comparing each number of buffered block with the block number of entries in the valid list. If it is matched, the block in the valid list is move to discard list, and block in the buffered list is inserted into valid list. After the checking all the blocks in the buffer, the blocks are flush to journal log. Then the discard command are generated with the regions of lurked discard lists. Since a discard command can group several discard regions, almost all the lurked discard regions can be clustered into one discard command. Lastly, the dicard command is transferred to flash device. As shown in the Figure 5, the dynamic interval controlling scheme can make a relatively larger size of commit IO than that of fixed interval scheme since it guarantee a pre-defined fixed number of blocks to be flushed to journal log at a transaction commit. In addition to that, is can accumulate more dispersed lurked discard regions than that of fixed interval scheme, and these are clustered into a discard command, which reduce discard command management overhead. As a result, It increases overall IO bandwidth.

4. Evaluation

Table 1. 4 Different File System's IO Configurations

	Conf. 1	Conf. 2	Conf. 3	Conf. 4
Base files	3000	2000	500	200
# transactions	10000(many)	5000(mid)	3000(mid)	1000(small)
File’s size	1KB ~ 10KB(small)	50KB ~100KB(mid)	500KB ~ 1MB(mid)	1MB ~ 2MB(large)
User Block sizes	Read = 512 bytes, write = 512 bytes			
Biases	Read / append = 5, create/delete = 5			

With the lurked discard regions in the journal transaction operations, we have analyzed the distribution of discard commands for the file system IO workloads that includes journal transaction workloads, by generating file system’s IO operations with create/read/write/delete, as IO request distribution varies. Postmark benchmark [11] is used for 4 different configurations, as described in Table 1. 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 configuration running, we have four types of experiments with regards to discard commands, nodicard, home discard, journal discard, and over40 discard, which means discards are not generated, discards are generated only home region, discards are generated in both of home and journal region, and discards are generated only for the range is over 40 file systems’ blocks, respectively. During the experiments, request size, elapsed time, request sequence, and bandwidth are profiled for discard command.

The Figures 4-7 depicts request distribution for generated discard commands for each configuration. For each configuration, the generated discard command and its discard range is plotted. From the figures, we identify that discard range for each discard is getting larger and sporadic as file size increases and number of files decreases. On the contrary, the discards with small range are generated frequently for configuration 1 and 2. We identify that the discards of small range give bad effects for other generic requests, which results in bandwidth reduction, while discards for large range give benefit for Flash device, which results in IO bandwidth increment, from the bandwidth results. The discard for journal region of file system, the discard range makes much more traffic for file system’s journaling operations.

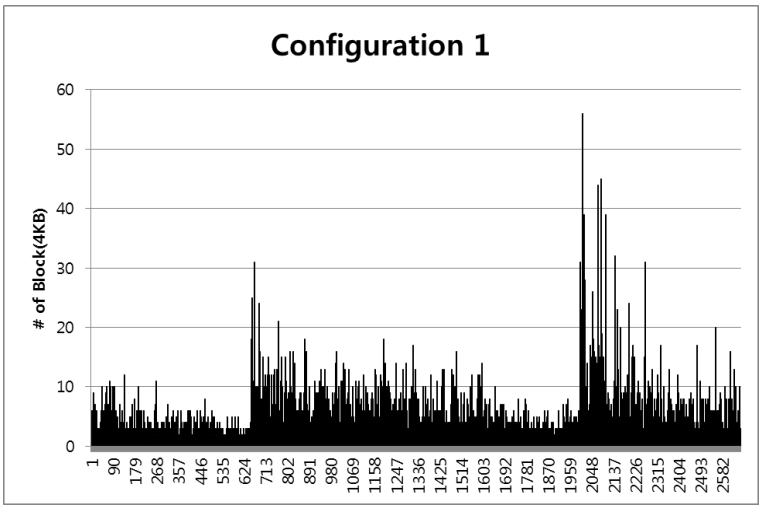

Figure 4. Discard Distribution for Configuration1

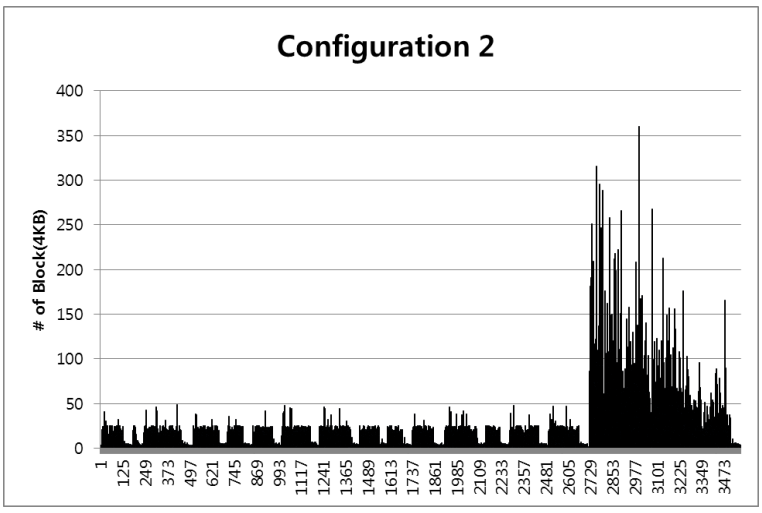

Figure 5. Discard Distribution for Configuration 2

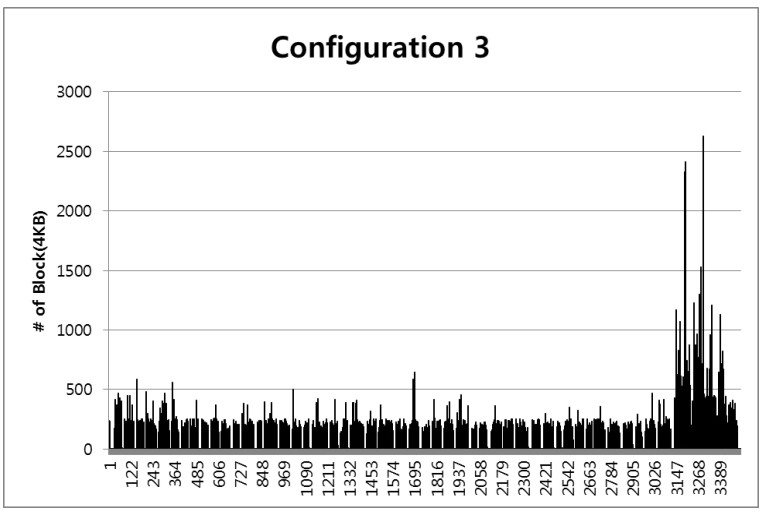

Figure 6. Discard Distribution for Configuration 3

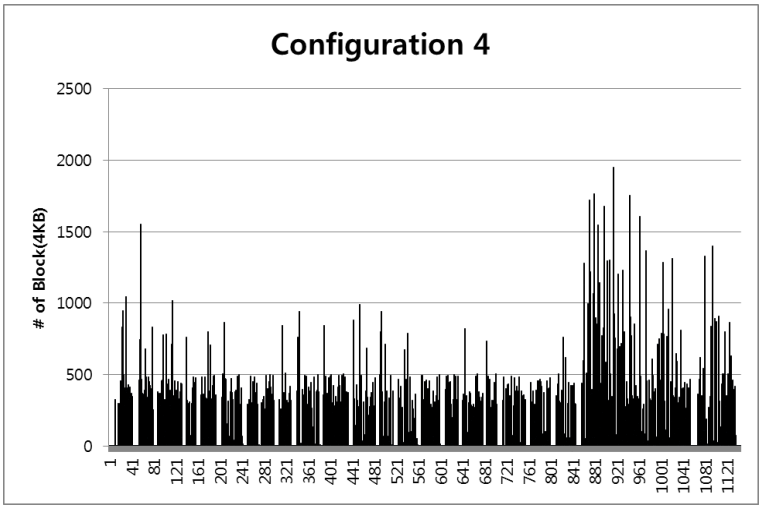

Figure 7. Discard Distribution for Configuration 4

To view the advantages of dynamic interval controlling of transaction, we have preliminary experiments with same experimental environment described above. The experiments are preliminary since we only set the buffering time dynamically with only regards to the numbers of lurked discard regions. In the journal configuration, default fixed interval is used for comparison with the dynamic interval controls, *i.e.*, five seconds for transaction interval. For the dynamic interval controlling, we have set the buffering interval to gather a set of sixty-four discard regions which is standard discard command parameter for aggregation of the discard rnages for preliminary experiments. The benchmark settings are same with above experiments, that is, there are four configurations with Postmark benchmakr. For each benchmark running, read and write throughputs are measured for both of fixed interval and dynamic interval. The Figure 8 depicts the prelinarity experimental results for dynamic interval controlling with comparison with fixed interval controlling. As shown in the results, the dynamic interval controlling outperforms fixed interval controlling for both of read and write throughput, as we intended. In the experiments, we identify that average interval time for dynamic interval controlling is three times longer than that of fixed interval controlling. The long interval time results in better IO throughput. It is from that the size of transaction commit IO for dynamic interval controlling is greater than that of fixed interval controlling, as well as ranges for lurked discard regions for dynamic interval controlling is larger than that of fixed interval controlling. For the further works, the various journal buffering configurations to apply dynamic interval controlling will be considered, which includes set of various buffering sizes, set of lurked discard regions, and combination of time interval and buffering size, and so on.

5. Conclusion

In modern storage system, most file system uses journaling operations to preserve data in-safe and recover fast from power loss. However, journaling operations generate excessive IO requests and degrades IO throughput. In addition to that, it also makes a lot of possible-discard regions that could be invalidated physically through discard command. However, due to it short period of time between journal transactions, there occur a lot of tiny lurked discard regions which give much overheads. In this paper, we identify that there exist many tiny lurked discard regions between successive journaling commit operations, and propose a dynamic interval controlling scheme of journal transactions commit to reduce tiny lurked

discard regions, as well as enlarge IO size of upcoming commit transaction. The dynamic interval controlling scheme changes journal transactions interval dynamically during runtime by monitoring the size of incoming transaction and number of lurked discard regions to adjust number of discard commands as well as size of upcoming transaction. With the help or dynamic controlling of transaction interval, overall IO bandwidth can be enhanced.

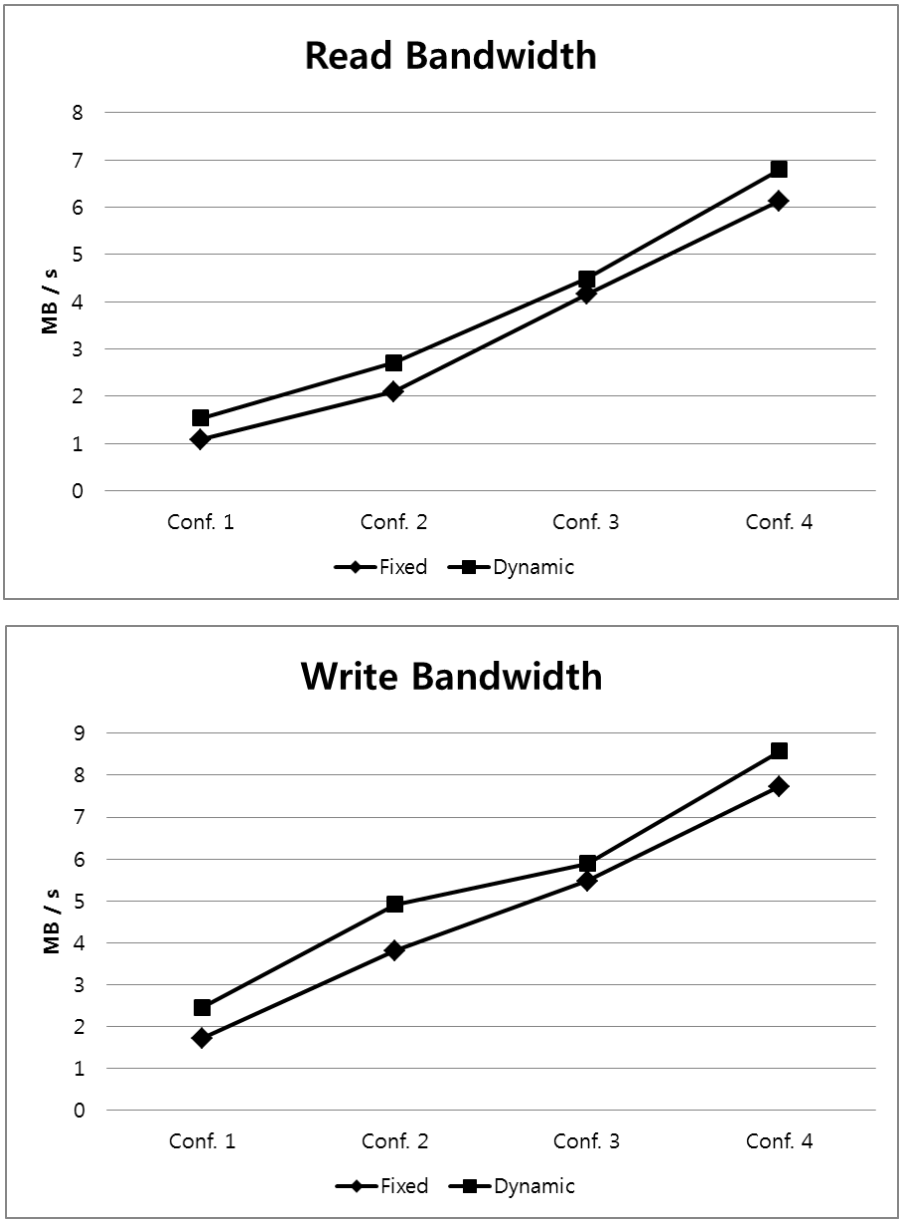

Figure 8. Preliminary Experimental Results for Dynamic Transaction Interval Controlling; Write Bandwidth and Read Bandwidth

Acknowledgements

This work was supported by Hankuk University of Foreign Studies Research Fund. The preliminary version of this work was appeared in CES-CUBE 2016.

References

[1] Intel Corporation, "Intel® High Performance Solid State Drive - Advantages of TRIM". www.intel.com, **(2010)**.
[2] JEDEC, "Embedded Multimedia Card Electrical Standard", **(2013)**.
[3] F. Shu, "Data set management commands proposal for ata8-acs2", In T13 Technical Committee, United States: At Attachment: e07154r1, **(2007)**.
[4] S.-H. Lim and Y.-S. Jeong, "Journaling deduplication with invalidation scheme for flash storage-based smart systems", Journal of Systems Architecture, vol.50, Issue 8, **(2014)**, pp 684-692.
[5] Y. Lee, L. Barolli and S.-H. Lim, "Mapping granularity and performance tradeoffs for solid state drive", Journal of Supercomputing, vol. 65, no. 2, **(2013)**, pp 507-523.
[6] Micron, "Garage Collection in Single-Level Cell NAND Flash Memory", Technical Note, TN-2960.
[7] A. Ban, "Flash file system", United States Patent, no.5, 404, 485, **(1995)**.
[8] Intel Corporation, "Understanding the flash translation layer (FTL) specification", http://developer.intel.com/.
[9] 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, Issue 1, **(2012)**.
[10] M. C. Avantika Mathur, and S. Bhattacharaya, "The new ext4 filesystem: current status and future plans", In Proceedings of the Linux Symposium, **(2007)**, pp. 21–33.
[11] J. Katcher, "PostMark: A New File System Benchmark", Technical Report TR3022, Network Appliance Inc., **(1997)**.

Authors

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 Division of Computer and Electronic Systems Engineering at Hankuk University of Foreign Studies. His research interests include operating systems, embedded systems, and flash storage systems.

