FTL DUO : STRATEGIC WRITE DISPATCHING AND OVER-PROVISION PARTITIONING OVER PAGE- AND HYBRID-MAPPING FTLS

Author

Ting-Chieh Huang

Supervisor

Li-Pin Chang

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
AT
NATIONAL CHIAO TUNG UNIVERSITY
HSINCHU, TAIWAN
JULY 2013

Table of Contents

List of Tables

List of Figures

Abstract

Real-life workload is mixed with long/sequential and small/random bursts, however, to optimize these two patterns in the single FTL makes design become more complicated. In this paper, we proposed a new design which takes advantages of hybrid mapping FTL and page mapping FTL to handle sequential and random pattern respectively. We focus on extracting different types of write patterns from the workload and dispatching appropriate write groups into each suitable FTL. Moreover, we apply overprovision partitioning mechanism for lower overall flash management cost because garbage collection overhead is inversely proportional to its overprovision size. Experimental result shows there are existing optimal partition point in each workload, and significantly reduce write amplification while increasing garbage collection pressure.

Acknowledgements

First of all, I would like to express my deeply gratitude to my advisor, Prof. Li-Pin Chang, for his significantly valuable guidance, encouragement and feedback during my master's degree. Without his advice, this thesis would not have been done so well. Also, I appreciate the efforts of the following two committee members, Prof. Ya-Shu Chen, Jen-Wei Hsieh, for their many beneficial suggestions to my thesis.

Secondly, I would also like to express special thanks to my senior colleagues Wen-Ping Li, Yi-Kang Chang, Tung-Tang Chou, Po-Han Sung, Chen-Yi Wen, junior colleagues Chun-Yu Lin, Pin-Xue Lai, Guan-Chun Chen, Ming-Chi Yan, Po-Hong Chen, Yu-Syun Liou, colleagues Sheng-Min Huang, Chau-Yuan Mao, Cheng-yu Hung, and all members of ESS lab who had provided me assistance in terms of ideas, discussion and other ways.

In addition, I extend my gratitude to my friends Merlin Pan, Chuen-Chu Chang, Tina Wang, and all of other friends who mentally support me and help me pass through challenges.

Finally, my deepest and greatest thanks to my parents for bringing me up and supporting me all along. Without their love and support, I could not have completed this work and achieved what I have today. Thank you.

September 12, 2013

Chapter 1

Introduction

NAND flash memory has been widely used as storage medium for solid stated drives (SSDs). Despite it has several advantages over hard disk drives (HDDs) such as superior random access performance, and shock resistance, there are still some limitations of NAND flash memory. The basic constraint are erase-before-write and write/erase operations which are slower than read operation, those limitations remarkably decrease write performance, so it makes NAND flash memory hard to replace with HDDs directly.

Flash Translation Layers (FTLs) has been designed to overcome inherent problem of NAND flash memory, it can re-write/update existing data out-of-place to avoid frequent flash block erase operations and to keep large mapping table. Block-level mapping is good at handling sequential pattern, and its mapping table size is smaller than Page-level mapping, but due to coarse granularity of mapping unit, small updated pages will cause significant reclaim overhead. However, in order to take advantage of the both page level mapping and block level mapping, log block based hybrid FTL [1] [2] [3] [4] has been proposed. Hybrid-level mapping adopt not only to shrink the mapping table size, but also to enhance the performance better than Block-level mapping. But it still suffers from performance degradation in the random write pattern due to excessive full merge operation will be triggered. We introduce two typical hybrid mapping schemes : BAST and FAST. BAST (Block Associative Sector Translation) [1] adopts block chain for each data blocks, but it easily suffer performance degradation from random workload because log blocks becoming insufficient and incurring a large amount of garbage collection operation which called block thrashing, . FAST (Fully Associative Sector Translation) [2] which use log block by page mapping for incoming updated sectors, however, multi-thread softwares cause small random writes interposing sequential writes, so that sequential pattern can not be easily distributed.

In the real workload, it contains different write access patterns, that is, long/sequential and small/random write burst are co-existing. From those past FTLs, they aimed at optimizing this two different write requests in the single FTL. For instance, FAST [2] use SW log blocks to optimize sequential write pattern, and DAC [5] apply data clustering for differentiating between cold sequential data and small random write data. However, designing a new efficient FTL to handle these two write patterns simultaneously in the individual mapping scheme is too complicated. From our observations, hybrid mapping scheme is good at dealing with long/sequential write, and page mapping scheme cope with small/random write well. Therefore, we exploit these merits of mapping schemes, and take the easy way to deal with these patterns concurrently and efficiently. We propose a new design which composed of write buffer, hybrid level mapping, and page level mapping FTL.

The main objective of this research is to separate long write burst and small random write burst before writing it into FTL, and we use write buffer with group dispatcher for dynamically assigning write burst to each suitable FTL. Hybrid mapping FTL deal with long write burst, and Page mapping FTL manages small random write burst. Our model differs from those convertible FTLs which take care of write requests for distributing write patterns. If only considering characteristics of write requests, it will easily destroy locality of write patterns, a long sequential write burst has been cut in by several small write requests will be promptly classified into non-sequential type request, conversely, we consider write bursts for maintaining the locality of write pattern. Besides, we separate write pattern out of FTL, hence, this model can be deployed easier. Furthermore, because any flash block could only be used by single FTL, and management cost of FTL is inversely proportional to its over-provisioned blocks [6] , we put forward a new policy about partitioning overprovisioned blocks. It adjusts the number of flash blocks from those two FTLs to achieve the lowest overall management cost.

Experiment result show there are existing a best overprovision partition point in each workload, and when we increase garbage collection pressure, the overall flash management cost will decrease significantly.

The rest of this paper is organized as follows. In section II, we mention background and the related work. In section III, we describe the system overview. In section IV, we explain the implementation issue. Experimental results are discussed in Section V. The last, we summarize the conclusion in Section VI.

Chapter 2

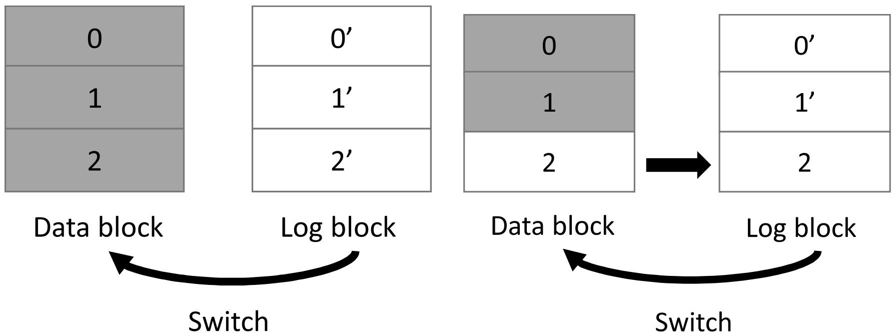
Background and Related Works

	Block mapping	Page mapping	Hybrid mapping
Mapping table size	small	large	medium
GC overhead	high	low	medium/high
Performance	low	high	medium
suitable pattern	sequential	small/random	long/sequentail

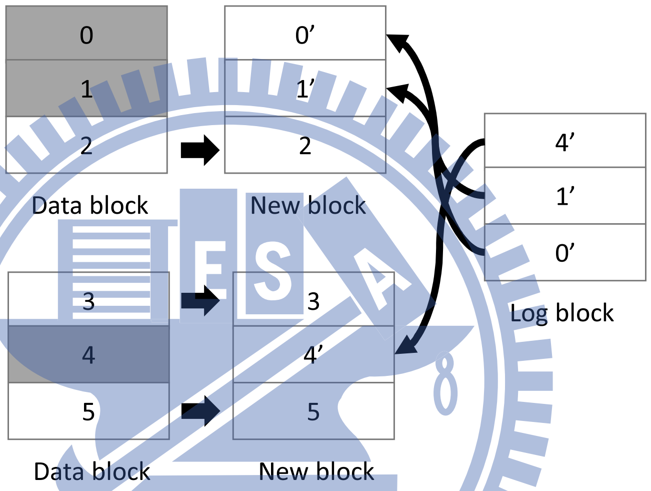
Table 2.1: Characteristic of typical FTLs

NAND flash memory which composed of several blocks, and each block comprised of multiple pages, in addition, a page has spare area, and data area which store mapping information and data respectively. NAND flash memory write unit is a page, but erase unit is a block. However, because of NAND flash memory physical constraints, it can not in-place-update. That is, when updated a page, we needs to find another free page to write incoming data, this called out-of-place updated, and original page is marked as invalid. Therefore, there exists a lot of invalid pages occupied in NAND flash memory. So that system should erase those invalid data pages some period time, for cleaning a free space to store other data, this called garbage collection operation.

Flash translation layer (FTL) is an essential component in SSDs, it emulates SSDs to be a block device for supporting read/write operation from file system directly. Existing FTLs can be classified into three typical designs on the table 2.1. Block level mapping



(a) Switch Merge (b) Partial Merge



(c) Full Merge

Figure 2.1: Merge Operations

only need small portion of RAM for storing mapping information, but each small write will cause expensive cost for updating whole block, so block level mapping suffers from small/random write. Page level mapping is good at dealing small/random write, but it requires considerable memory space for storing mapping table. Assume storing a mapping information of a page needs 4Byte, a 16GB SSD use 16MB to store mapping information, it apparently requires too much memory space. For resolving defects of these two FTLs above, sorts of hybrid mapping FTLs have been proposed. It comprises page level mapping for log blocks and block level mapping for data blocks. Log block which is associated with one or more data blocks serves incoming write requests. When log space is insufficient, garbage collection operation will be activated, it recycles valid pages of log block with corresponding data blocks. Merge operations can be composed of switch merge, partial merge, full merge. Switch merge operation (refer to figure 2.1a) is that we can fill data in entire logical block into a log block, then switch this log block to replace old block directly, that is, erase a block per switch merge incur. Partial merge operation (refer to figure 2.1b) is that write pages into log block in logical sequence but do not fill entire log block, then it needs read old valid pages of this logical block and fill them into log block, so that we can erase original data block, which means if write data in log block is few, it needs to copy more valid pages. Full merge operation (refer to figure 2.1c) is that do not write in logical sequence into log block, so it should copy valid pages of log block and corresponding data block's valid pages into other new block. Then it erase origin data block, and log block for recycling.

However, it also still suffers performance degradation from random pattern because heavy cost of full merge. real-life workload is mixed with sequential and random pattern. In order to handle real-life workload, S. Lee et al. [7] have proposed a new Hybrid mapping FTL which concerns about locality of different access patterns. They determine the locality type by each request size for separating different write pattern, but it make the design of FTL more complicated. D. Park et al. [8] proposed convertible FTL which not only dynamically switches mapping scheme to access pattern by using hot/cold identifier, but also fully exploit the merits of block mapping scheme and page mapping scheme, besides, they use two-level mapping table for reducing address mapping translation overhead. Q. Wei et al. [9] proposed that using a small part of flash memory for buffering write requests in order to generate long/sequential bursts. According to write bursts size, it adaptively dispatches write bursts which large write bursts to block mapping area and small write bursts to page mapping area. H. Kwon et al. [10] present a mechanism to find the optimal utilization of log blocks for lower overall write costs which by recycle wasted space in data block so that dynamically change the size of page mapping area and block mapping area. moreover, they design two methods : fusion and de-fusion for data migration between Page Mapping Area(PMA) as log block space and Block Mapping Area(BMA) as data block space. C. Wang et al. [11] proposed a method which divides log space into sequential area and random area for sequential and random write respectively. By monitoring the ratio of switch merge and ratio of full merge, it adaptively change the size of each log space by workload, which means if performance suffers from insufficient block, blocks will transfer from opposite area. In addition, they use historical access table for predictively transferring pages between areas for avoiding unnecessary data movements in the merge process.

2.1 The Difference between Our Work and Related Work

2.1.1 Dispatch Write Pattern

We focus on extracting different types of write patterns from the workload and dispatching appropriate write bursts into each suitable FTL rather than only considering write request for separating pattern. Therefore, we can maintain locality of write pattern by write buffer. In contrast, long write bursts probably be interposed by several small writes in real workload so that its write requests will be dispatched to non-sequential write request. That is, the efficiency of dispatching write pattern will decrease. Although Q. Wei et al. [9] use small portion of flash for producing long/sequential bursts, they do not concern about the relation of garbage collection overhead which is inversely proportional to its overprovision size.

2.1.2 Overprovision Adjustment

We apply adaptive overprovision partitioning mechanism for lower flash management cost, because garbage collection overhead is inversely proportional to its overprovision size. D. Park et al. [8] and Q. Wei et al. [9] focus on keeping the mapping structures compact but ignores the relation between overprovision and garbage collection cost, except H. Kwon et al. [10] and C. Wang et al. [11] take this relation into consideration.

Chapter 3

FTL Duo - Design Concept

In this section, we describe our proposed design - FTL Duo

3.1 System Architecture Overview

From figure 3.1, our architecture has a small size write buffer, write group dispatcher, hybrid-mapping FTL, page-mapping FTL, and overprovision adjustor. The write buffer management policy we use Block-LRU [12], and hybrid-mapping FTL we take FAST as default. Nowadays, High-end SSDs employs with small size write buffer to alleviate write traffic, absorb hot data, on top of that, it can produce long/sequential write burst as possible as it can, so our model takes it into consideration, this will help to maintain its locality.

At first, all of write requests will be written into write buffer, and then write buffer produce write groups which stands for a logical block. Second, write group dispatcher separate large and small group by group dispatch policy. The idea depends on each write pattern is suitable to different FTLs.

We use hybrid mapping FTL to handle with long/sequential bursts, and page mapping FTL to deal with small/random bursts. Through easily revising mapping table and page

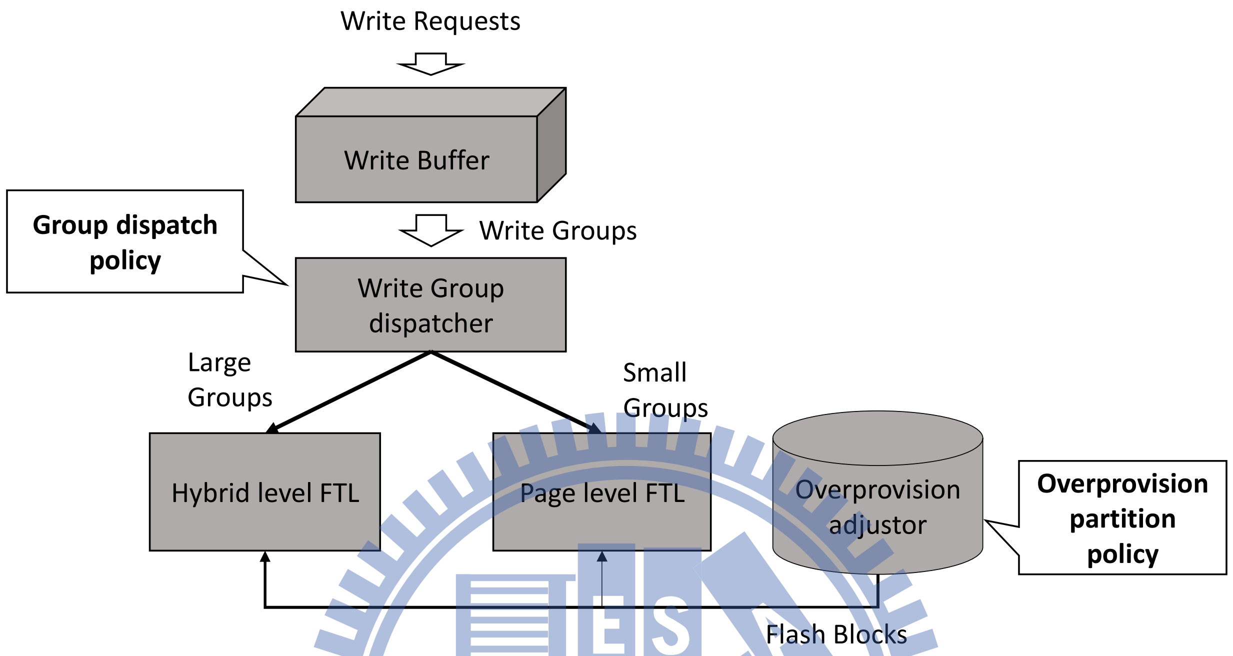


Figure 3.1: System Overview

invalidation mechanism, we can write data between these two FTLs. Notice that, we partition all blocks into two set of blocks for hybrid mapping FTL and page mapping FTL, and System allocates over-provisioned blocks for each FTL, not to partition flash memory into two space physically.

Moreover, we concern the relation that garbage collection cost is inversely proportional to its overprovision size. By overprovision partition policy, overprovision adjustor shifts over-provisioned blocks for lower overall flash management overhead.

3.2 Group Dispatching Policy

We consider two factors about group size and group updated interval. A evicted group from the write buffer is defined as a logical block, group size stands for how many dirty sectors a group has, moreover, let group updated interval denote the interval which between two flushed group by the same logical block. The reason we take group updated interval

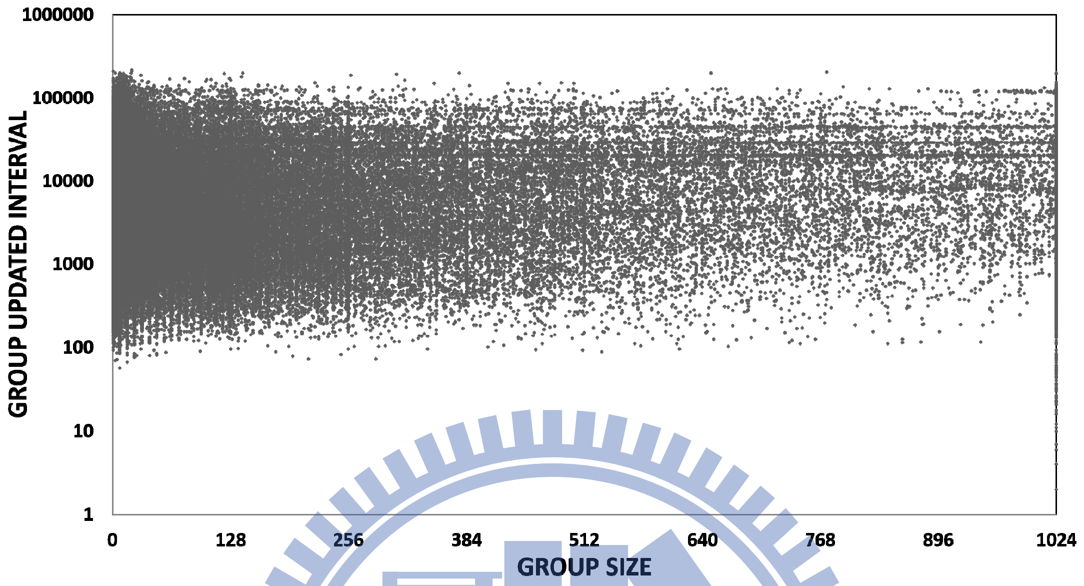
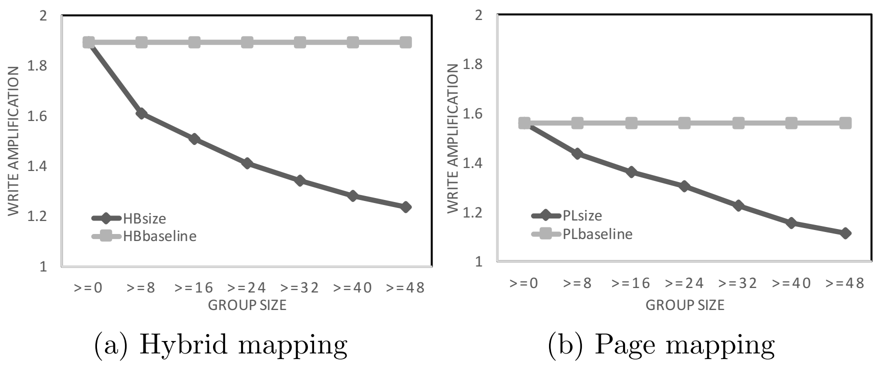


Figure 3.2: The distribution of group size and group updated interval

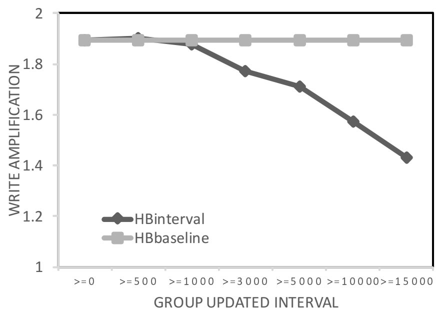
into consideration is that we assume that the data of short updated interval groups can be absorb by log block. If group updated interval is too long, the written data from the same logical block probably already recycle by full merge, that is, log block cannot absorb the cost of full merge.

Figure 3.2 shows the distribution between group size and group updated interval of all evicted groups. In the group size factor, it has no evident distribution, on the other hand, it centralized between ten thousand to one hundred thousand by group updated interval factor. We discuss the effect by this two factors below.

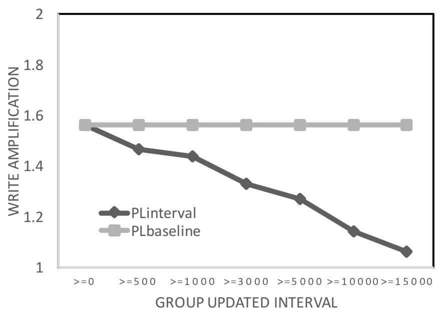


(a) Hybrid mapping (b) Page mapping

Figure 3.3: Remove small size groups



(a) Hybrid mapping



(b) Page mapping

Figure 3.4: Remove short updated interval groups

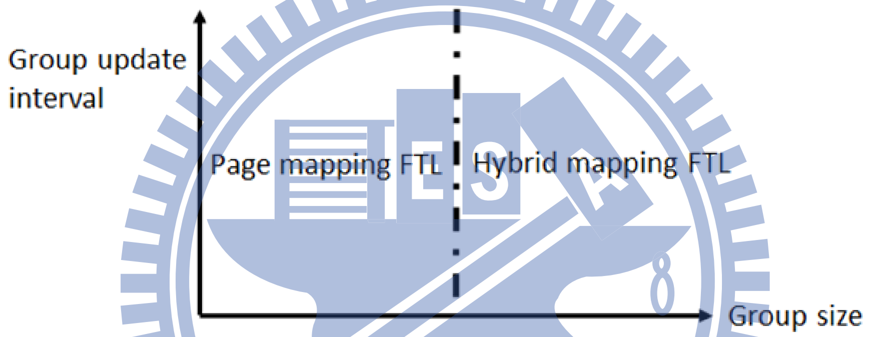


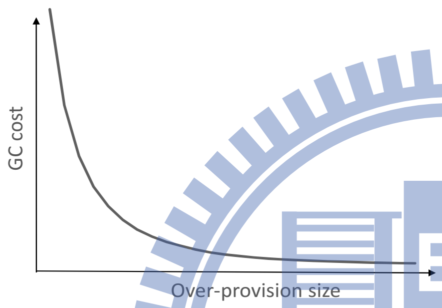
Figure 3.5: Write Group Classification Policy

Let HBBaseline and PLBaseline denote the baseline which all groups be written into hybrid mapping FTL/page mapping FTL ,and write amplification which defined as the average of actual number of flash page writes per user page writes which regarded as write cost. First, from group size prospect (refer to figure 3.3), we remove small size groups from workload comparing hybrid mapping FTL to page mapping FTL. It significantly decrease in hybrid mapping FTL, but it diminishes more slowly in page mapping FTL, which means small size group is not suitable for hybrid mapping FTL.

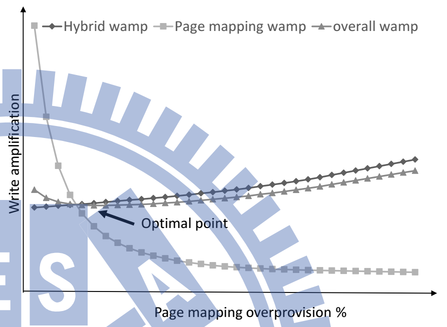
Second, from group updated interval prospect (refer to figure 3.4), we remove short updated interval groups from workload comparing hybrid mapping FTL to page mapping FTL. The decline of write amplification slope are almost identical, that means short updated interval group put either hybrid mapping FTL or page mapping FTL cause no difference.

Summary by our observation, small size groups are suitable to page mapping FTL, and large size groups are better to hybrid mapping FTL. Updated interval factor is not a valuable reference for dispatching write bursts between these two FTLs.

3.3 Overprovision Adjustment Policy



(a) GC cost v.s. Overprovision in the single FTL



(b) GC cost v.s. Overprovision between the two FTLs

Figure 3.6: GC cost is inversely proportional to overprovision size

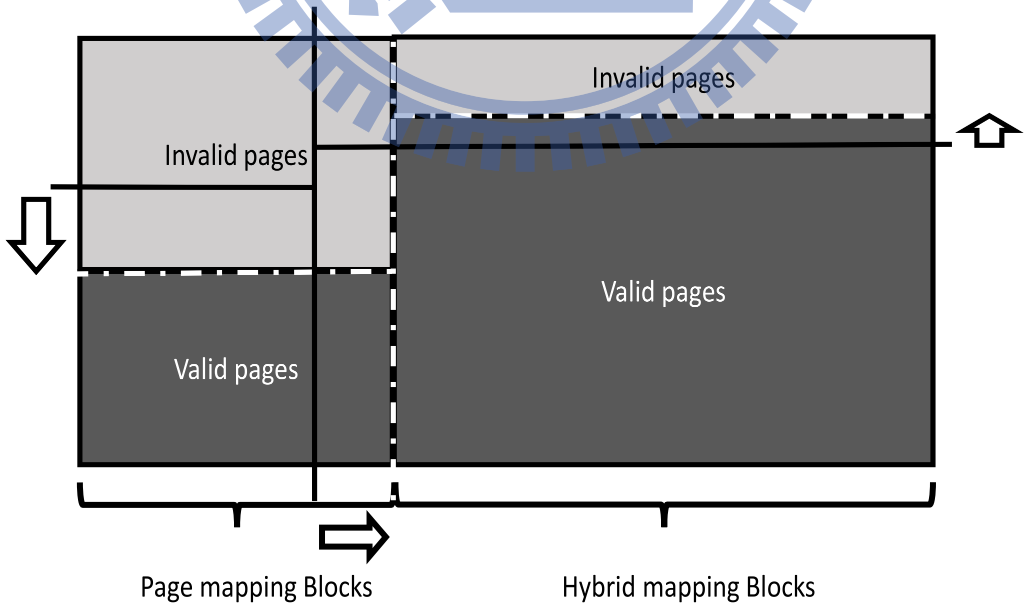


Figure 3.7: The concept of moving overprovisioned block between two FTLs

As we mentioned, garbage collection overhead is inversely proportional to its overprovision size for single FTL (refer to figure 3.6a). By our design, overprovision space is shared with two FTLs, there must be existing an optimal overprovision partition point. (refer to figure 3.6b) The graph shows when we allocate more over-provisioned blocks to page mapping FTL, the write amplification of page mapping FTL will significantly decrease, that is at the time, we allocate less over-provisioned blocks to hybrid mapping FTL, the write amplification of hybrid mapping FTL will slightly increase consequently.

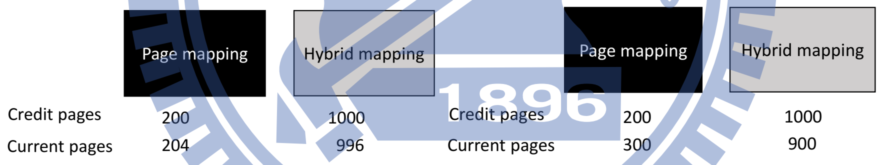
The example of moving spare blocks among two FTLs described below.

From figure 3.7, system dispatch a large amount of long/sequential data into hybrid mapping FTL, and a small amount of small/random data into page mapping FTL. The full line represents the portion of invalid pages and valid pages before shifting over-provisioned block. Assume the write cost of hybrid mapping FTL, we shift blocks from hybrid mapping FTL to page mapping FTL, the dotted line shows after shifting blocks. Consequently, the ratio of average valid pages in page mapping FTL is significant decreasing, it means that write costs decrease considerably in page mapping FTL, but the ratio of average valid pages in hybrid mapping FTL is slight increasing, consequently, overall write cost is decreasing. Therefore, we use this relation to find the best partition point between two FTLs for reducing overall flash management overhead.

Chapter 4

Implementation Issue

4.1 Credit-based Overprovision Control policy



(a) Under-allocated credit pages (b) Over-allocated credit pages

Figure 4.1: Credit-based Overprovision Control

Because that immediate block movement unnecessarily trigger garbage collection, and for maintaining credit pages number of each FTL, we proposed Credit-based overprovision control policy to delay the block shifting and postpone garbage collection. Therefore, we can reduce additional garbage collections effectively.

Let a credit page denote a spare page that an FTL can have, and it not only can be allocated or not, but also can either be free or containing invalid data. Current pages stands for how many allocated spare pages in the FTL right now.

There are two situations when we need a free block : Under-allocated or Over-allocated (refer to figure 4.1). Under-allocated means FTL is under tolerance and FTL has enough spare pages, so it can do garbage collection first rather than get back credit pages, hence, we can delay to shift spare blocks. Over-allocated is that FTL lend/borrow too many credit pages, it force to get back credit pages first, therefore, it postpone doing garbage collection. From the example above, in the figure 4.1a, hybrid mapping FTL only lend four credit pages to page mapping FTL, when hybrid mapping FTL needs a free block, it will do garbage collection by itself if necessary. In the figure4.1b, hybrid mapping FTL lend one hundred credit pages to page mapping FTL, so that when it needs free blocks, it should get credit pages from page mapping FTL first.

4.2 Adaptive Overprovision adjustment policy

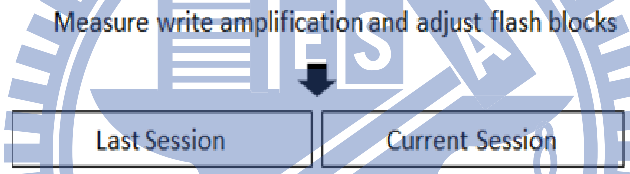


Figure 4.2: Dynamically overprovision adjustment policy

In adjustment policy, we proposed a adaptive overprovision adjustment policy for shifting over-provisioned blocks in the workload. The policy is to detect the write amplification in each session to move blocks, each session is defined as a fixed write data amount. If write amplification of page mapping is large than hybrid mappings, then shifts blocks from hybrid mapping FTL to page mapping FTL. Conversely, if write amplification of hybrid mapping is large than page mappings, then shifts blocks from page mapping FTL to hybrid mapping FTL. (refer to figure 4.2

Chapter 5

Experiments Results

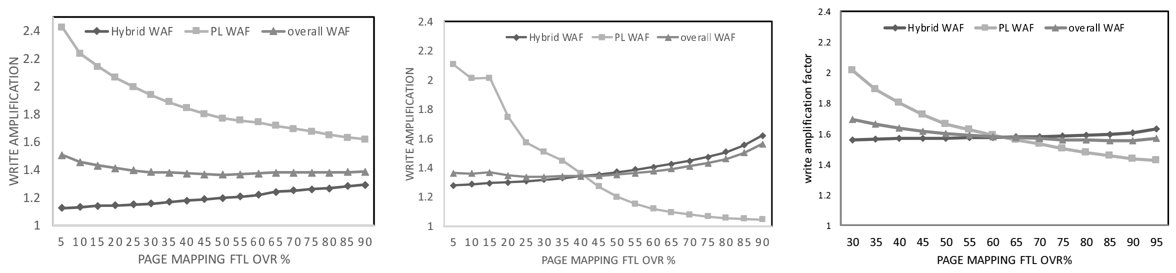
5.1 Experimental Environment

Workload	Operating System	File system	Total Written (GB)
Notebook	Win XP	NTFS	27
PC	Win XP	NTFS	81
Ubuntu	Ubuntu 9	Ext 4	55

Table 5.1: Characteristic of the experimental platform

We implement FTL duo which comprised of hybrid mapping FTL (FAST), page mapping FTL, Write buffer (Block-LRU) on simulator, and use three different real-life workloads which collected from different hosts refer to table 5.1. From our observation behind, group dispatch policy is considering group size factor which dispatching large size write groups to Hybrid mapping FTL and small size write groups to page mapping FTL. The following experiments we selected the best group size threshold for each workload as default. Experimental configuration denote page/block size, overprovision size, buffer size in order such as 4k512k5%16M. We expect that theres existing a best partition point in each workload.

5.2 Find the best partition point



(a) NB workload (b) PC workload (c) Ubuntu workload

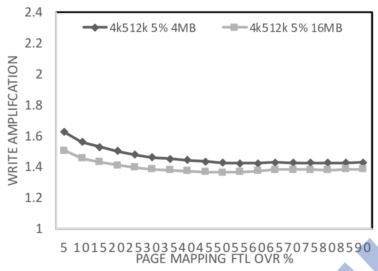
Figure 5.1: Best partition point in different workload

For finding the best partition point, we shift over-provisioned blocks between hybrid mapping FTL and page mapping FTL, then we can get the lowest overall write amplification point, and its partition point is the best partition point. Page mapping overprovision size % stands for assigning how much percentage of total overprovision size to page mapping FTL, besides, Hybrid mapping FTL overprovision size is (1 - page mapping overprovision size %). The figure 5.1 shows there exists the best overprovision partition point between hybrid mapping FTL and page mapping FTL in these three real workload which experimental setting configured in 4k512k5%16M. In NB workload, there exists best partition point in 50% page mapping overprovision size which write amplification is 1.365. In PC workload, there exists best partition point in 25% page mapping overprovision size which write amplification is 1.335. In Ubuntu workload, there exists best partition point in 85% page mapping overprovision size which write amplification is 1.554.

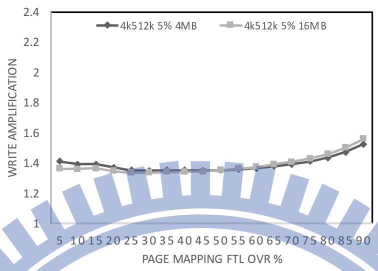
5.3 The effect of increasing garbage collection pressure

For finding the effect of increasing garbage collection pressure, we change three different factors about garbage collection cost : buffer size, total overprovision size, and page/block size. And we also shift over-provisioned blocks between hybrid mapping FTL and page mapping FTL to measure overall write amplification variance. We expect that shifting over-provisioned blocks has more obvious effect while garbage collection pressure is increasing.

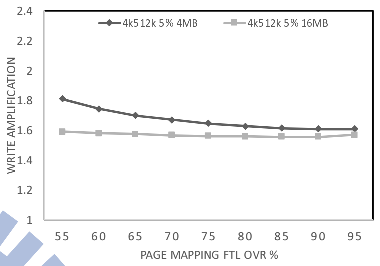
5.3.1 Different buffer size



(a) NB workload



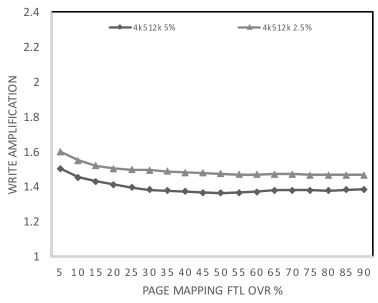
(b) PC workload



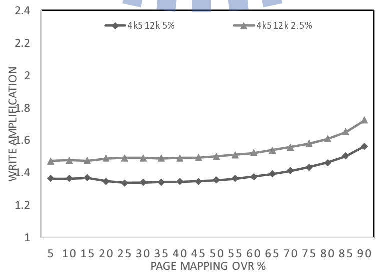
(c) Ubuntu workload

Figure 5.2: Adjust buffer size

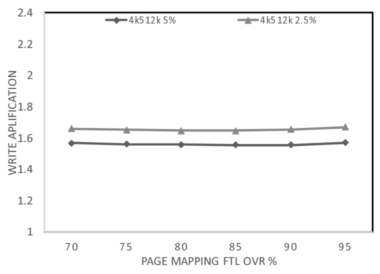
Here we adjust different buffer size in fixed configuration 4k512k5%. The figure 5.2 shows in 4 MB buffer size has more benefit than 16 MB in NB workload. Also in Ubuntu workload, the figure shows the curve in 4 MB buffer size bends more obviously than 16 MB in NB workload. However, in the PC workload, There are slightly different between different write buffer sizes in PC workload, it probably means PC workload contains less random pattern than two other workload.



(a) NB workload

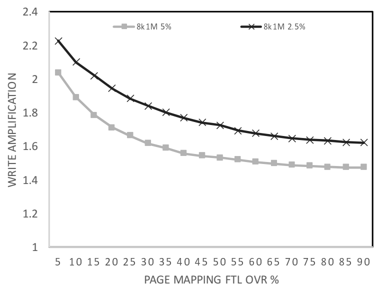


(b) PC workload

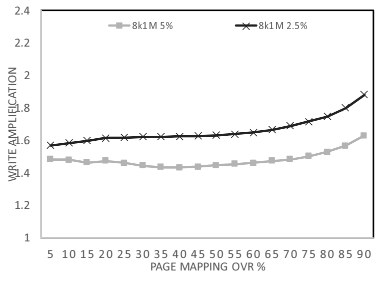


(c) Ubuntu workload

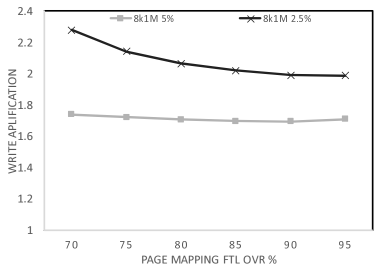
Figure 5.3: Adjust overprovision in 4k512k16M



(a) NB workload



(b) PC workload



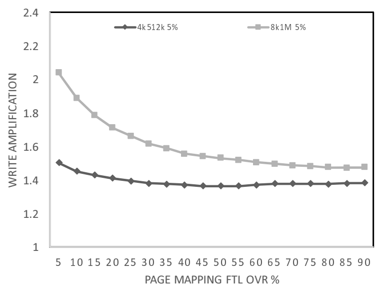
(c) Ubuntu workload

Figure 5.4: Adjust overprovision in 8k1m16M

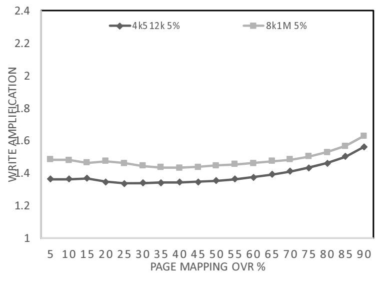
5.3.2 Different overprovision size

Here we adjust different overprovision size in fixed configuration. In 4k512k16M configuration, the figure 5.3 shows 5 % overprovision size and 2.5 % overprovision size do not appear apparently difference, which means the garbage collection pressure probably is not so high, therefore, we configure in 8k1m16M to measure write amplification in different overprovision size. The figure 5.4 shows there exists 2.5% overprovision size has more effect than 5% overprovision size by shifting over-provisioned blocks. That is, in small overprovision size has more impact by shifting over-provisioned block.

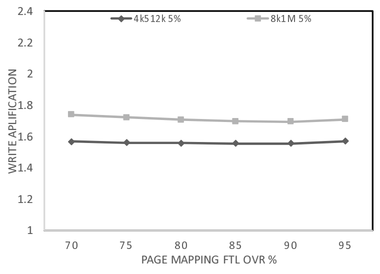
5.3.3 Different page/block size



(a) NB workload



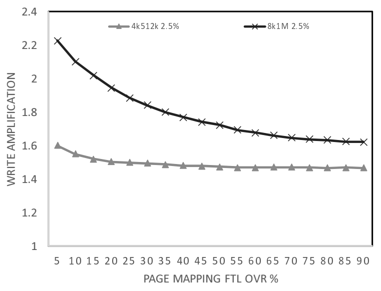
(b) PC workload



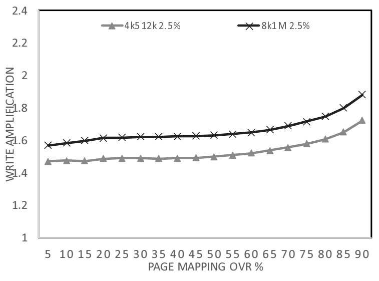
(c) Ubuntu workload

Figure 5.5: Adjust page/block size in 16M5%

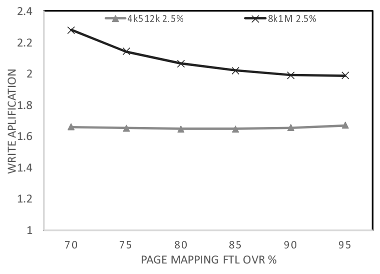
Here we adjust different page/block size in fixed configuration. The figure 5.5 shows in 5%16M configuration, different page/block size does not show significant difference, same as we mentioned, probably garbage collection pressure is not high, hence, we lower



(a) NB workload



(b) PC workload



(c) Ubuntu workload

Figure 5.6: Adjust page/block size in 16M2.5%

configuration in 2.5%16M refer to figure 5.6, to measure measure write amplification in different page/block size. The figure shows there exists 8k1m page/block size has more effect than 4k512k page/block size by shifting over-provisioned block. That is, in large page/block size has more influence by shifting over-provisioned block.

5.4 Summary on Our Experimental Results

From experimental results, we can get the best partiton point in each different workload by shifting overprovision blocks between two FTLs, moreover, our approach particularly useful when the garbage collection pressure is high in which smaller total overprovision size, larger block size, and more random write patterns.

Chapter 6

Conclusion

We present a new design called FTL duo which comprises of write buffer, page mapping FTL and hybrid mapping FTL. The basic idea is to exploit write buffer to separate large write bursts from random writes and dispatch different write patterns to different FTLs. We observe that large size groups are suitable to hybrid mapping FTL and small size groups are better to page mapping FTL, however, updated interval factor is not so useful for distribute write pattern for these two FTLs. Moreover, our design apply overprovision adjustment policy which is adjusting overprovision space helps lower garbage collection overhead and improve performance effectively. Experiment shows our approach particularly useful when the garbage-collection pressure is high.

Bibliography

[1] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," *Consumer Electronics, IEEE Transactions on*, vol. 48, no. 2, pp. 366–375, 2002.

[2] 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 (TECS)*, vol. 6, no. 3, p. 18, 2007.

[3] 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 (TECS)*, vol. 7, no. 4, p. 38, 2008.

[4] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," *Journal of Systems Architecture*, vol. 55, no. 5, pp. 332–343, 2009.

[5] M.-L. Chiang, P. C. Lee, and R.-C. Chang, "Using data clustering to improve cleaning performance for flash memory," vol. 29, no. 3. London, New York, Wiley Interscience [etc.], 1999, pp. 267–290.

[6] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, p. 10.

[7] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "Last: locality-aware sector translation for nand flash memory-based storage systems," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 6, pp. 36–42, 2008.

[8] D. Park, B. Debnath, and D. Du, "Cftl: a convertible flash translation layer adaptive to data access patterns," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1. ACM, 2010, pp. 365–366.

[9] Q. Wei, B. Gong, S. Pathak, B. Veeravalli, L. Zeng, and K. Okada, "Waftl: A workload adaptive flash translation layer with data partition," in *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on.* IEEE, 2011, pp. 1–12.

[10] H. Kwon, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Janus-ftl: finding the optimal point on the spectrum between page and block mapping schemes," in *Proceedings of the tenth ACM international conference on Embedded software.* ACM, 2010, pp. 169–178.

[11] C. Wang and W.-F. Wong, "Adapt: Efficient workload-sensitive flash management based on adaptation, prediction and aggregation," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on.* IEEE, 2012, pp. 1–12.

[12] H. Kim and S. Ahn, "Bplru: A buffer management scheme for improving random writes in flash storage." in *FAST*, vol. 8, 2008, pp. 1–14.