Mapping Granularity Adaptive FTL Based on Flash Page Re-programming

Yazhi Feng, Dan Feng, Chenye Yu, Wei Tong‡, Jingning Liu‡,
Wuhan National Lab for Optoelectronics, School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, China
{fengyazhi, dfeng, yuchenye, Tongwei, jnliu}@hust.edu.cn
‡ Co-corresponding authors: Wei Tong (Tongwei@hust.edu.cn) and JingningLiu (jnliu@hust.edu.cn)

***Abstract*—The page size of NAND flash continuously grows as the manufacturing process advances. While larger page can reduce the cost per bit and improve the throughput of NAND flash, it may waste the storage space and data transfer time. Meanwhile, it causes more frequent garbage collections when serving small write requests. To address the issues, we proposed a Mapping Granularity Adaptive FTL (MGA-FTL) based on flash page re-programming feature. MGA-FTL enables a finer granularity NAND flash space management and exploits multiple subpage writes on a single flash page without erase. 2-Level Mapping is introduced to serve requests of different sizes in order to control the overhead of DRAM requirement. Meanwhile, the allocation strategy determines whether different logical pages can be mapped to a single physical page to balance the space utilization and performance. Subpage merging limits the number of associated physical pages to a logical page, which could reduce data fragmentation and improves the performance of read operations. We compared MGA-FTL with some typical FTLs, including page-level mapping FTL and sector-log mapping FTL. Experimental results show that MGA-FTL reduces the I/O response time, write amplification and the number of erasures by 53%, 30% and 40% respectively. Despite the overhead of fine-grained management, MGA-FTL increases no more than 16.5% DRAM requirement compared with a page-level mapping FTL. Unlike the subpage-level mapping, MGA-FTL only needs one third of DRAM space for storing mapping tables.**

I. Introduction

With the advance in flash memory manufacturing process, the storage density of NAND flash chip has been increased in many folds. The basic read and write unit of flash is a flash page, whose size is continuously growing from 512 bytes to 4KB and 8KB. Table shows the evolution of the chip capacity and page size of NAND flash memory [1]. Now some flash chips with 16KB pages have already been shipped. Recently, 3D NAND Flash memories have been introduced to keep the trend of increasing bit density with the potential to scale up to several Terabits. However, 3D flash memory devices face the problem of big blocks [2]. A large block might consist of a large number of pages or the pages with large size. So the page size may continue increasing even larger in the near future. In order to reduce the cost of bit, solid-state drive (SSD) manufacturers are increasingly likely to choose the large-capacity and high-density flash memory chips, which degrades the lifetime and performance when serving small updates.

TABLE I: Evolution of NAND Flash Page Size

	2006	2007	2008	2009	2010	2014
Process	90nm	72nm	50nm	34nm	25nm	16nm
Chip Capacity	2Gb	8Gb	16Gb	32Gb	64Gb	128Gb
Page Size	2KB	2KB	4KB	4KB	8KB	16KB

Flash Translation Layer (FTL) is used in SSDs to emulate the Hard Disk Drive (HDD) interface. Thus, most file systems can be directly used on SSD without any changes. However, the default block size of file systems such as NTFS [3] and ext4 [4] is 4KB. Flash pages are several times the size of File System Blocks (FSBs). For example, if the flash page size is 16KB, it needs four 4KB FSBs to fill up a full page. When updating 4KB data in a flash page, 16KB old data in the entire page need to be read, merged with the updated data and then written to a new flash page. These operations cause a lot of extra page read, write and erase cost which wastes the data transfer time, resulting in significant performance loss and lifetime degradation.

To address the above issues, an intuitive way is to manage all flash pages at the granularity of subpages (the same size of FSBs). However, the subpage mapping table entries will increase several times of page-level mapping and the cost of DRAM is unacceptable, e.g., 1 TB device with 16KB pages and 4KB subpages needs 2GB storage space for mapping tables. On the other hand, the past work of subpage-level mapping [5] only supports to write part of a page, and leads to a low space utilization. We took advantages of the characteristics of the underlying hardware to make full use of the entire flash page.

In this study, we present the Mapping Granularity Adaptive Flash Translation Layer (MGA-FTL). We focus on the impact of large flash pages to the I/O performance and lifetime of SSDs. MGA-FTL manages SSDs in different granularities, writes small requests into subpages and serves large requests with entire flash pages. MGA-FTL makes use of the page re-programming feature of flash to achieve higher page utilization. In contrast with other design [6], Subpage merging operations is not necessary in our work and the DRAM requirement of mapping tables is carefully controlled.

The rest of this paper is organized as follows. Section II describes the background. The details of MGA-FTL design are presented in section III and we describe the evaluation in

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

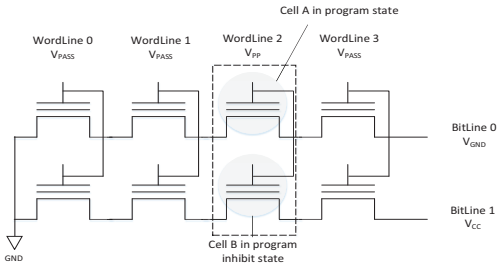


Fig. 1: NAND Array schematic view

section IV. Section V introduces the related work. Finally we drew a conclusion and look to the future work in section VI.

II. Background

A. Flash Re-programming Feature

Floating-gate cells, the storage unit of flash chip, whose logical state depends on the amount of electrons stored in the floating-gate. The logical state of a cell without any electron is '1', and it turns to '0' when the electrons tunneling into its floating-gate.

Figure 1 shows the schematic view of a flash block. In a NAND string, the cells are connected in series and share the same bitline. Control gates are connected through wordlines. Logical pages (indicated as dash-dotted box in the graph) are made up of cells belonging to the same wordline [7]. Four flash pages are shown in the schematic view, each with a capacity of two flash cells. When programming page 2 to state '01' (cell A to '0' and cell B to '1'), a high voltage Vpp (i.e., 9-20V) is applied to cell A's corresponding wordline, so as to inject electrons into the floating gate of cell A. The so-called self-boosting mechanism [8] prevents cell B from undergoing an undesired program (remains '1'). The corresponding bitline is driven to Vcc to boost the potential of the channel in order to reduce the voltage drop across the tunnel oxide and, hence, inhibit the tunneling phenomena.

By leveraging the self-boosting mechanism, small amount of data could be programmed to one subpage of a flash page when serving small requests. The other subpages remain unchanged and could serve the subsequent small requests so the flash page can be written more than once. We used in house platform [9] and Samsung SLC flash chips [10] to verify the effectiveness of this method. The experiment results show that this method works out well within the number of NOP (Number Of partial Programs) referred in the datasheet. NOP indicates the number of a flash page can be re-programmed [10][11]. In order to avoid too much program disturbance and ensure the accuracy of the data, this parameter should be set into some small values, e.g. 4 or 8 for SLC flash chips and 1 for MLC.

NOP is typically 1 for MLC flash chips because a MLC cell stores two bits and each cell stores one Least Significant Bit (LSB) and one Most Significant Bit (MSB). The state transition of LSB pages and MSB pages has more restrictions [12]. [13] proposed an "LLH" method which enables MLC flash to be re-programmed before erase operations. We could use this method to implement programming a page more than

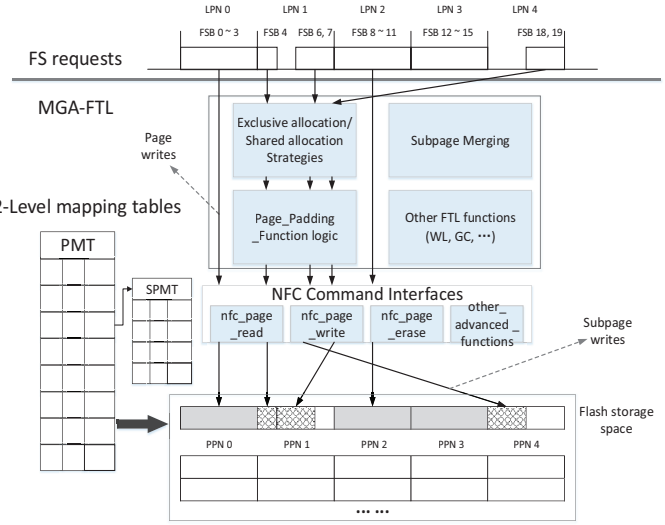


Fig. 2: The architecture of MGA-FTL

once at the granularity of subpage before erasing a MLC block. The principle for "LLH" method is to ensure the cells' threshold voltage increasing linearly. Similar to the MLC flash, theoretically, we might support re-programming on TLC or other flash whose cells store more than three bits.

III. The System Design

To improve the utilization of large flash pages, we proposed a novel FTL design called Mapping Granularity Adaptive FTL by taking advantage of flash page re-programming feature. Figure 2 shows the overview of MGA-FTL. On the basis of normal FTL, MGA-FTL uses 2-Level mapping tables and exclusive and shared allocation strategies, and supports subpage write and merging. 2-level mapping tables effectively control the overhead of DRAM requirement. The allocation strategies balance the space utilization and performance. Subpage writes pad the subpage with masks to program a flash page. Though subpage merging is not a necessary operation, considering reducing data fragmentation and the improvement of read operations, subpage merging is introduced in MGA-FTL. We also extended the states of flash pages to describe subpage read/write operations more precisely. And we redefined the transition between each states and made a new state transition diagram.

A. 2-Level Mapping Tables

MGA-FTL is a page-level and subpage-level hybrid mapping FTL. It serves small requests with subpage-level mapping and other requests with page-level mapping. Page-level Mapping Table (PMT) is the first level mapping table and SubPage-level Mapping Table (SPMT) is the second level mapping table. MGA-FTL does not allocate the second level mapping table until the size of request smaller than a flash page arrives. 2-level mapping tables could largely reduce the DRAM requirement compared with only using subpage-level mapping and ensure its memory footprint would not exceed too much than page-level mapping.

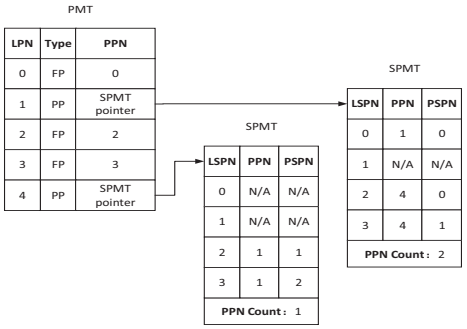


Fig. 3: 2-Level Mapping Tables

Figure 3 shows structure of the 2-level mapping tables. We assume every flash page contains 4 subpages. FP and PP in the type field stand for ”full page” and ”partial page” to which a logical page maps. LPN and PPN represent logical page number and physical page number. Similarly, LSPN and PSPN denote the Logical SubPage Number and Physical SubPage Number within a flash page respectively. In this case, the logical page 0, 2 and 3 are mapped to a flash page directly without SPMT, and logical page 1 and 4 are mapped to the subpages in different flash pages with SPMT. SPMT, the second level mapping table, stores the Physical Page Number (PPN) and the corresponding Physical SubPage Number (PSPN) to keep the mapping information completely. Besides, SPMT keeps the number of physical pages associated with the logical page, as a reference when choosing the subpages to merge.

B. Implementing Subpage Writes

As describe in section II, by leveraging the program inhibit feature, we can achieve writing one subpage as follows. 1) Write data to the corresponding subpage which has not been used. 2) Fill up the rest of the page with ‘1’ as the mask to maintain the corresponding data unchanged. To implement filling mask method, for general consideration, instead of modifying the read/write logic of NAND Flash Controllers (NFCs), we extended firmware codes for the SSD controller. We added function (page_padding) before calling NFCs’ interfaces (page_read/page_write). In this way, hosts only need to transfer the subpage sized data to SSD since “page_padding” will fill up the rest of flash page with 0xFF. This way could eliminate the extra data transfer. As Figure 4 shows, a flash page consists of four subpages. At first, all the subpages are in free state. After programming the subpage 0 in the flash page, there are still three free subpages available. The rest subpages are able to serve the other requests since the flash page can be written more than once. The page would not be erased until all the subpages become invalid, being merged or its block selected as GC victims.

C. Flash Page States Transition

Generally, the states of flash pages include free, valid and invalid. Because MGA-FTL uses a subpage as the smaller access unit, these three states cannot describe the states of flash page completely. Thus, we added a new state called

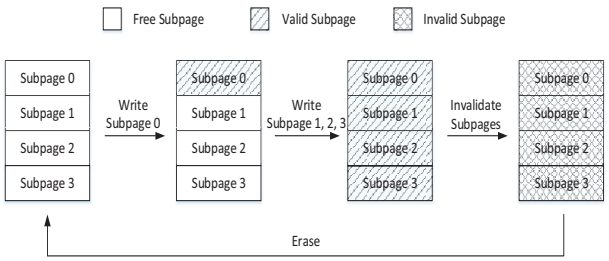


Fig. 4: Subpage Programming

partially valid (PV) to indicate a flash page with data stored in some of its subpages. So MGA-FTL maintains 4 flash page states and figure 5 shows the transitions between these states. The transitions between fully free (FF), fully valid (FV) and fully invalid (FI) states are similar to common page-level mapping. When writing or updating a subpage in some page in FF or FV state, the state of this page changes to PV. When merging subpages, if the page is chosen as the victim, the valid subpages within it are moved to another page called target page. After merging operation, the victim page changes from PV state to FI state and waits to be erased. When a target page gathers all the subpages of a logical page, it changes to FV state.

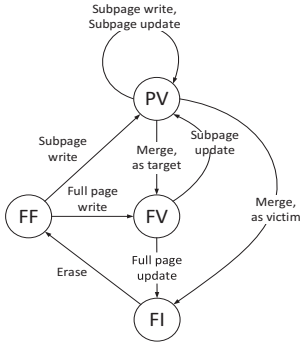


Fig. 5: Physical Page States Transition

D. Allocation Algorithm

We introduce two different allocation strategies, Exclusive Allocation (EA) and Shared Allocation (SA) In MGA-FTL. EA allocates the same physical page for the requests which belong to the same logical page, i.e., the physical page is exclusive to a certain logical page. The advantage of EA is reducing overhead of subpages merging but with low space utilization. SA allocates available subpages for the requests, no matter whether they belong to the same logical page, so the space utilization of shared allocation is higher. SA tends to use partial free space in PV state pages, while EA tends to consume FF state pages. MGA-FTL integrates both allocation strategies, and determines which to use dynamically. We define an Allocation Threshold (AT) to compare to the ratio of FF state pages to all the flash pages. When the ratio is higher than AT, EA is applied for lower frequency of subpage merging. Otherwise, we use SA to increase the space utilization and postpone the garbage collection.

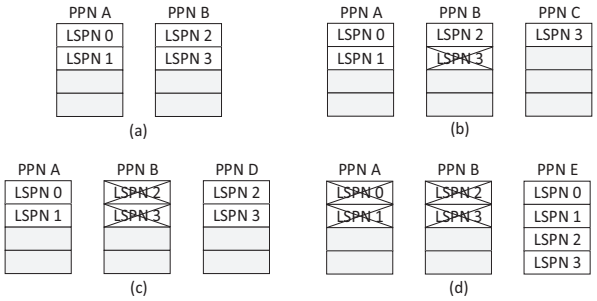


Fig. 6: Subpage Merging

E. Subpage Merging

Subpage allocation brings a problem that the logical subpages in one logical page may be distributed to different physical pages, resulting in a fragmentation of flash storage space. When the host reads a logical page, it needs to read multiple physical pages, thereby lowering the read performance of the SSD. To improve the read performance in this situation, MGA-FTL limits the number of physical pages (PPN Count) associated to a logical page, and the number of read operations of a logical page (Read Count) is taken into account as a threshold to trigger subpage merging. The basic idea is that the more read operations a logical page has, the less physical pages it should be associated. For simplicity to describe, we set the boundary between Read Count and PPN Count as follows:

if $Read\ Count == 0$; **then** $PPN\ Count <= 4$;
if $0 < Read\ Count <= 2$; **then** $PPN\ Count <= 3$;
if $2 < Read\ Count <= 5$; **then** $PPN\ Count <= 2$;
if $Read\ Count > 5$; **then** $PPN\ Count == 1$;

If a logical page has not been read ever, its subpages can be written to multiple physical pages. As the number of read operations increases, the number of associated physical pages reduces. When the PPN Count reaches the merging threshold, the system will only allow one physical page stores the logical page by merging its logical subpages, even if it will cause extra space waste. As shown in figure 6 (a), assuming there is one logical page, and it is divided to four subpages, LSPN 0-3, located in two physical pages, PPN A and PPN B. Now the file system updates LSPN 3. If Read Count corresponding to this logical page is 0, we can follow the way shown in figure 6 (b), write LSPN 3 to a subpage which is in the PPN C and retain LSPN 2 in the PPN B, and then update the PPN Count of this logical page to 3. If Read Count of this logical page is greater than or equal to 4, PPN Count should be less than 3. As figure 6 (c) shows, we need to write both LSPN 3 and LSPN 2 to subpages in PPN D, which does not change PPN Count, but invokes an extra subpage migration. If Read Count of this logical page is 10, as shown in figure 6 (d), we need to guarantee all the subpages attached to this logical page are copied to a free physical page. Though the merging operation results in 3 additional subpage migration, it will benefit the read performance of this logical page afterwards.

IV. Evaluation

In this section we first present the experiment environment and then apply a variety of real workloads to evaluate MGA-FTL with different configurations. For comparison, other two existing FTLs are implemented in our system as well.

TABLE II: Workloads Analysis

Workloads	Avg.Size (Read/Write) (4KB)	Read/Write Ratio (%)	Annotation
CFS	23.90/81.99	90.6/9.4	Read-time communication
RADIUS	210.67/22.51	8.1/91.9	Verification server
Ads	63.80/11.28	86.7/13.3	Ads data service
Trade	3.23/5.91	18.8/81.2	Online trade
Synth-A	1/1	40.8/59.2	100% 4KB request
Synth-B	6.58/5.38	45.5/54.5	50% 4KB request

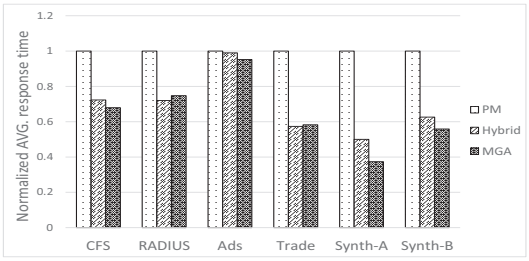
A. Environment Setup

We choose 4 representative real workloads presented by the storage networking industry association (SNIA) and Microsoft for evaluation. In addition, we synthesized two workloads, Synth-A and Synth-B, whose 4KB size requests account for 100% and 50% respectively. The workload details are listed in Table II.

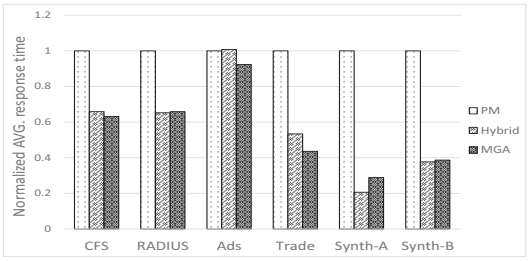
We implement MGA-FTL, page-level mapping, and sector-log hybrid mapping in an event driven SSD simulator. The code for the simulator and FTLs is available online [14]. For convenience, we use MGA, PM and Hybrid for short to stand for these 3 FTLs in the following diagrams. We set 4KB as the size of file system block and flash subpage, and we use 2 different flash page sizes, 8KB and 16KB, with 2 and 4 subpages respectively. And we assume a block has 128 pages and a plane consists of 512 blocks. Other parameters of I/O performance refer to Samsung flash chip datasheet [10].

B. Experiment Results

1) Response Time: MGA-FTL allocates subpages for the request smaller than a flash page, reducing the data transfer between the host and the device. Figure 7 shows the average response time in different page sizes. MGA-FTL and Hybrid both reduce the response time compared to page-level mapping. Especially, MGA-FTL reduces the average response time by 72% dramatically for the synthesized workload Synth-A. For the real workload of small random write workload Trade, the average response time is reduced up to 53%. Hybrid is close to MGA-FTL in most cases, as Hybrid also divides the space of solid state disk into fixed subpage-level mapping area. However, when the free space of this subpage-level mapping area is getting low, Hybrid needs more subpages merge operations, leading to a significant performance degradation. Figure 8 shows the number of subpages transferred between the host and the device in workload Trade. At the third request in the sequence, MGA-FTL transferred 7 subpages distributed in 4 flash pages. With page-level mapping, writing 4 pages needs to transfer 8 or 16 subpages when page size is 8KB or 16KB. Thus, MGA-FTL reduces 1 or 9 subpages transfer respectively. In conclusion, with large flash pages and smaller



(a) 8KB Page Avg. Response Time



(b) 16KB Page Avg. Response Time

Fig. 7: Average Response Time

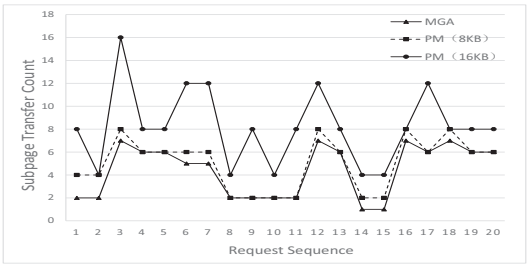


Fig. 8: Subpage Transfer Count

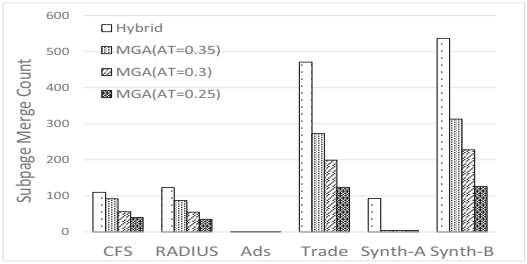


Fig. 10: Subpage Merge Count

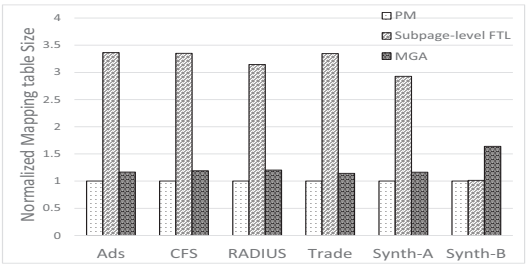


Fig. 11: Mapping Tables Count

host request, MGA-FTL could improves the performance of SSD obviously.

2) Space Utilization: MGA-FTL uses the free subpages to respond to small requests. Thus, the utilization rate of the storage space of the SSD is improved compared with page-level mapping. The higher SSD space utilization ratio is, the less block erase operations there will be. Figure 9 illustrates the erase number corresponding to 8KB flash page and 16KB flash page respectively. For the real workload Trade which mainly consists of small requests, MGA-FTL reduces 40% flash erase operations due to the effective reduction of write amplification by subpage access. When using the 8KB/16KB Flash page, MGA-FTL reduces write amplification by 7.42% and 30.83%. It can be seen that with the larger flash page size and the higher proportion of small requests, MGA-FTL can reduce more erasure operations and achieve higher space utilization. Therefore, the lifetime of SSD is extended.

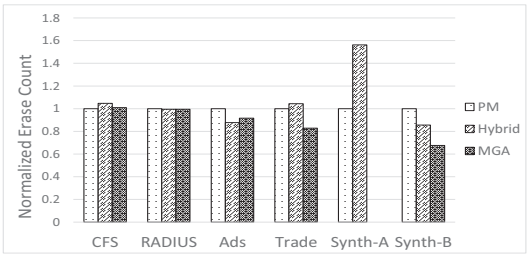
3) Subpage Merging: MGA-FTL and Hybrid both involve the operations of subpage merging. This causes the extra read and write overhead. MGA-FTL uses Allocation Threshold (AT) to balance Exclusive Allocation and Shared Allocation. Since the area for subpage-level mapping is limited in Hybrid FTL, the shrinking of this area leads to merging operations with high overhead.

Figure 10 compares the subpage merge number between MGA-FTL and Hybrid when using 16KB pages. To test the impact of AT on subpage merging, we set 3 values for AT, 0.25, 0.3 and 0.35. Because these AT values are in the range of GC start and stop thresholds, MGA-FTL can switch between EA and SA with different free page ratios. Test results show that the subpage merge number of MGA-FTL is less than Hybrid FTL, which is consistent with the analysis above. So we conclude that higher AT can improve space utilization and postpone garbage collection, but brings more merging overhead. However, this overhead is less than that of Hybrid FTL and we can get improved read performance at the cost of it.

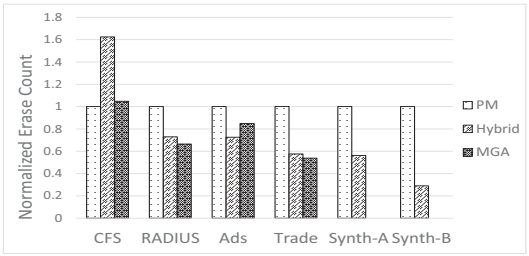
4) DRAM Overhead: figure 11 shows the size of mapping tables of different FTLs. The size of mapping tables is normalized to that of PM, which is determined by capacity of SSDs. Though the fine-grained subpage mapping will inevitably cause more memory space, 2-level mapping tables enables DRAM requirement will not increase dramatically compared with page-level mapping. Memory usage of MGA increases no more than 16.5% compared with page-level mapping. For the same capacity, subpage-level mapping tables take up 3.X times the memory of MGA-FTL because of the large number of subpage table entries.

V. Related Work

Several different methods to solve the unmatched problem between write units and requests have been proposed. Kim [5] suggests a subpage programming scheme to extend the lifetime of NAND flash memory. Without leveraging re-programming feature, the low space utilization causes more erase operations and degrades performance. Another method using a finer-grained address mapping approach is sector-log mapping [6]. This method manages a small part of NAND flash memory in SSDs with sector-level mapping and the rest of the flash area with page granularity. The disadvantage of the sector-log ad-



(a) 8KB Page Erase Count



(b) 16KB Page Erase Count

Fig. 9: Erase Count

dress mapping table is its large DRAM memory footprint and sector data merging overhead. Other similar approaches are Hybrid-FTL [15][16] and Super-page strategies [17]. Hybrid-FTL also faces the challenge of merging operations. And the super-page's efficiency depends on request locality and DRAM buffer capacity.

There have been several studies using of WOM codes to reduce erasures [18][19] recently. However, the WOM codes have some disadvantages such as the complexity of the algorithms and the computational and space overhead [20]. This paper presents a more practical way that re-programs a flash page at the granularity of subpage.

VI. Conclusion

The problem that the request size from a file system is not aligned with the underlying flash page size degrades random access performance, shortens the lifetime of flash memory chips and causes the waste of data transfer time and storage space. We proposed a finer-grained access optimization strategy based on the re-programming characteristics of flash memory. We design a mapping granularity adaptive flash translation layer to manage the subpage granularity access. The results show that the proposed FTL can reduce the I/O response time, write amplification and the number of erasures by 53%, 30% and 40% respectively. The finer-grained subpage mapping and more complex FTL inevitably cause more memory space, which is carefully controlled and evaluated. Since MLC and TLC flash chips gain more and more popularity, we will modify MGA-FTL to adapt additional constraints on MLC flash reprogramming in the future work.

Acknowledgment

This work was supported by the National High Technology Research and Development Program (863 Program) No.2015AA016701, No.2015AA015301; NSFC No.61303046, No. 61402189, No.61472153; State Key Laboratory of Computer Architecture, No.CARCH201505; Wuhan Applied Basic Research Project (No.2015010101010004); This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

References

[1] J. Cooke, "Nand 201: An update on the continued evolution of nand flash," *Micron*, 2011.

[2] M. C. Yang and Y. M. Chang, "Garbage collection and wear leveling for flash memory: Past and future," in *International Conference on Smart Computing*, 2014, pp. 66–73.

[3] J.-w. JU and L.-y. WANG, "Analysis of ntfs file system," *Computer Engineering and Design*, vol. 22, p. 033, 2007.

[4] M. Cao, S. Bhattacharya, and T. Ts'o, "Ext4: The next generation of ext2/3 filesystem." in *LSF*, 2007.

[5] J. H. Kim, S. H. Kim, and J. S. Kim, "Subpage programming for extending the lifetime of nand flash memory," in *Design, Automation & Test in Europe Conference & Exhibition*, 2015, pp. 555–560.

[6] S. Jin, J. Kim, J. Kim, J. Huh, and S. Maeng, "Sector log: fine-grained storage management for solid state drives," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 360–367.

[7] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND flash memories*. Springer Science & Business Media, 2010.

[8] K.-D. Suh, B.-H. Suh, and Y.-H. Lim, "A 3.3 v 32 mb nand flash memory with incremental step pulse programming scheme," *Solid-State Circuits, IEEE Journal of*, vol. 30, no. 11, pp. 1149–1156, 1995.

[9] Z. Li, S. Zhang, J. Liu, and W. Tong, "A software-defined fusion storage system for pcm and nand flash," in *Non-Volatile Memory System and Applications Symposium*, 2015.

[10] S. Electronic, "K9xxg08uxa datasheet," 2006-01, http://www.samsung.com.

[11] E. Micron, "64gb, 128gb, 256gb, 512gb asynchronous/synchronous nand features," 2012.

[12] J. Lee and D. Shin, "Adaptive paired page prebackup scheme for mlc nand flash memory," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 7, pp. 1110–1114, 2014.

[13] F. Margaglia, G. Yadgar, E. Yaakobi, Y. Li, A. Schuster, and A. Brinkmann, "The devil is in the details: Implementing flash page reuse with wom codes," in *14th USENIX Conference on File and Storage Technologies FAST*, 2016.

[14] C. Yu and Y. Feng, "SSD Simulator for multi FTL and Hardware Config," 2016, https://github.com/aadkl91/SSD-Simulator-for-multi-FTL-and-Hardware-Config.

[15] S. J. Kwon, H.-J. Cho, and T.-S. Chung, "Hybrid associative flash translation layer for the performance optimization of chip-level parallel flash memory," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, p. 13, 2013.

[16] S. W. Lee, D. J. Park, T. S. Chung, D. H. Lee, S. Park, and H. J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *Acm Transactions on Embedded Computing Systems*, vol. 6, no. 3, pp. 150–151, 2007.

[17] A. M. Caulfield, L. M. Grupp, and S. Swanson, "Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications," *ACM Sigplan Notices*, vol. 44, no. 3, pp. 217–228, 2009.

[18] G. Yadgar, E. Yaakobi, and A. Schuster, "Write once, get 50% free: Saving ssd erase costs using wom codes," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 257–271.

[19] S. Odeh and Y. Cassuto, "Nand flash architectures reducing write amplification through multi-write codes," in *MASS Storage Systems and Technologies*, 2014, pp. 1 – 10.

[20] D. Burshtein and A. Strugatski, "Polar write once memory codes," *Information Theory, IEEE Transactions on*, vol. 59, no. 8, pp. 5088–5101, 2013.