Rapid Recovery by Maximizing Page-Mapping Logs Deactivation

Jung-Hoon KIM\(^{a)}\), Member

SUMMARY As NAND flash-based storage has been settled, a flash translation layer (FTL) has been in charge of mapping data addresses on NAND flash memory. Many FTLs implemented various mapping schemes, but the amount of mapping data depends on the mapping level. However, the FTL should contemplate mapping consistency irrespective of how much mapping data dwell in the storage. Furthermore, the recovery cost by the inconsistency needs to be considered for a faster storage reboot time. This letter proposes a novel method that enhances the consistency for a page-mapping level FTL running a legacy logging policy. Moreover, the recovery cost of page mappings also decreases. The novel method is to adopt a virtually-shrunk segment and deactivate page-mapping logs by assembling and storing the segments. This segment scheme already gave embedded NAND flash-based storage enhance its response time in our previous study. In addition to that improved result, this novel plan maximizes the page-mapping consistency, therefore improves the recovery cost compared with the legacy page-mapping FTL.

key words: NAND flash-based storage, flash translation layer, page-mapping logging policy, consistency, recovery

1. Introduction

Since the NAND flash memory era has begun, NAND flash-based storage has become a popular storage media. As a significant component of that storage, a flash translation layer (FTL)\(^{[1]}\)–\(^{[5]}\) translates data addresses for completing stor-nificant component of that storage, a flash translation layer (FTL) based storage has become a popular storage media. As a sig-1. Introduction

As NAND flash-based storage has been settled, a flash translation layer (FTL) has been in charge of mapping data addresses on NAND flash memory. Many FTLs implemented various mapping schemes, but the amount of mapping data depends on the mapping level. However, the FTL should contemplate mapping consistency irrespective of how much mapping data dwell in the storage. Furthermore, the recovery cost by the inconsistency needs to be considered for a faster storage reboot time. This letter proposes a novel method that enhances the consistency for a page-mapping level FTL running a legacy logging policy. Moreover, the recovery cost of page mappings also decreases. The novel method is to adopt a virtually-shrunk segment and deactivate page-mapping logs by assembling and storing the segments. This segment scheme already gave embedded NAND flash-based storage enhance its response time in our previous study. In addition to that improved result, this novel plan maximizes the page-mapping consistency, therefore improves the recovery cost compared with the legacy page-mapping FTL.

key words: NAND flash-based storage, flash translation layer, page-mapping logging policy, consistency, recovery

1. Introduction

Since the NAND flash memory era has begun, NAND flash-based storage has become a popular storage media. As a significant component of that storage, a flash translation layer (FTL)\(^{[1]}\)–\(^{[5]}\) translates data addresses for completing storage I/O services. As one of the granular FTLs, the FTL running the page-mapping scheme separates the total storage capacity into small page groups. Therefore, the page-level FTL should control its many page mappings and needs to memorize their locations.

The page-level FTL could purposely store the page mappings to NAND flash memory on its storage. Because the volatile memory (e.g., Static Random Access Memory (SRAM) or Dynamic Random Access Memory (DRAM)) as the first reference is used only for caching the page mappings, that storing can protect the page mappings’ consistency on the NAND flash memory. For example, if the storage employs a small volatile memory that cannot hold all page mappings, unloading updated mappings (e.g., victim-page cache eviction\(^{[4]}\)) without the storing breaks the page mappings’ consistency. However, the frequent page-mapping write could cause large-written amounts and hurt the storage latency performance\(^{[6]}\). This situation might appear mainly through embedded NAND flash-based storage devices (e.g., eMMC\(^{[7]}\) and UFS\(^{[8]}\)).

The NAND flash-based storage might engage a large volatile memory, such as solid-state drives (SSDs). If that storage supports to load the entire page mapping table on the volatile memory, the FTL could eliminate numerous page-mapping stores caused by the small cache, therefore minimizes those written overheads. However, since an event of a power loss damages the volatile memory, the FTL should also guarantee the page mappings’ consistency recorded only on the volatile memory. If there were no page-mappings’ writing before the power loss event, the FTL might need to scan the entire NAND flash memory to recover the page mappings\(^{[9]}\)–\(^{[11]}\).

To minimize the mapping recovery costs by the inconsistency,\(^{[12]}\) proposed the periodic mapping writes in a log-structured way while exploiting the FAST FTL characteristics. This research stored newly-generated address mappings adequate for hybrid-mapping FTLs. However, it did not try to manage a massive page mapping table requiring a larger memory space. And,\(^{[13]}\),\(^{[14]}\) insisted on period-ically creating the checkpoints; however, its block-mapping level cannot take advantage of the granular page-mapping level. As for a page-mapping level FTL,\(^{[15]}\) implemented the metadata log structure similar to a journal\(^{[16]}\) of file systems. However, this policy should bear syncing the metadata logs that could cause massive amounts of writes. For the fast recovery,\(^{[17]}\) claimed for updating the page mappings newly written on the NAND flash memory. Though, this proposition did focus on increasing the page-mapping consistency only at the start-up time.

This letter proposes a novel plan that enhances the page-mapping consistency during the storage run-time. This method controls the page mappings divided by a virtually-shrunk segment and deactivate page-mapping logs by assembling and storing the segments. Through a similar segment scheme, our previous research\(^{[6]}\) let the embedded storage enhance its response time. As another approach, this method extends the page-level FTL that employs the legacy page-mapping logs\(^{[15]}\),\(^{[18]}\) especially on a mas-sive volatile memory. Consequently, the evaluation result shows that the page-level FTL independently adopting the segment maximizes the page mappings’ consistency, therefore improves the recovery cost compared with the legacy page-mapping FTL.

\(^{[1]}\) The author is with the Device Solution Research, Samsung Electronics Co., Ltd., Hwasung, Republic of Korea.
\(^{a)}\) E-mail: peculiarity@gmail.com
DOI: 10.1587/transinf.2020EDL8141
2. Page-Mapping Consistency Enhancement

Figure 1 shows the page-level FTL’s components needed for the page mapping table (PMT) management on the NAND flash-based storage. If the volatile memory caches the entire PMT, the FTL does not need to unload the victim-page mappings and store them to the NAND flash, named a PMT block in Fig. 1. Therefore, the FTL could immediately refer to and update page mappings when serving storage services (e.g., host-data I/Os or garbage collection). However, those mapping changes break the consistency on the PMT block. To record them, the FTL could utilize the page-mapping log structure. That structure includes a logical page number (LPN) and a physical page number (PPN).

Because the page-mapping log structure is different from the PMT structure, repeating to accumulate logs might increase the PMT reconstruction cost. This cost happens commonly after an event of a power loss. To reduce the PMT reconstruction cost, the page-level FTL could write PMTs containing any updated page mappings to the PMT block. If the FTL performs those PMT writes, the consistency of those page mappings evidently could increase.

However, the PMT writes might need to share data-transferring bandwidth with other data, such as host-user data. So, the page-level FTL can confine the PMT data to one NAND page to minimize its data transaction. One NAND page is the least amount to store data to NAND flash memory. This letter’s plan to the page-level FTL engages the virtually-shrunk segment to increase a resolution entirely for the PMT. Therefore, the PMT should be divided into smaller virtually-shrunk segments to speed up the page-mapping consistency. One NAND page that assembles the different segments might increase the number of up-to-date and changed page mappings.

Figure 2 employs the virtually-shrunk segment to deal with the PMT in Fig. 1. Contrary to one NAND page that takes one PMT involving consecutive 4096 LPNs, one PMT write includes eight different segments simultaneously. One segment consists of consecutive page mappings smaller in size than one PMT, claimed as a mapping segment table (MST) in the previous research [6]. If this segment contains page mappings of consecutive 512 LPNs, LPN 0 written to PPN 560 in the page mapping log zone 0 belongs to Segment 0, and LPN 4096, located in PPN 960, comes to Segment 8. Likewise, Segment 1, 4, and 9 also own their consecutive LPN’s page mappings. If most amounts of the page-mapping changed data are in Segment 1, 4, and 9, the consistency swells up while immediately invalidating all the page-mapping logs accumulated previously to the page-mapping log blocks. That collection showed considerable consistency improvement in the evaluation results of enterprise server workloads composed of small random writes.

Because containing the up-to-date and changed page mappings, the PMT writes’ repetition can invalidate all the page-mapping logs. However, the page-mapping log blocks of Fig. 1 only gather the page mapping logs, not considering valid or invalid log data. Deploying two logical-log zones of Fig. 2 as this letter’s novel plan can differentiate the invalid data from all the log data. Figure 2 shows the segments’ write, including Segment 1, 4, and 9, to invalidate all the valid-log data belong to the Segments. After deactivating all valid-log data in the log-zone 0, the page-mapping logs’ write changes from the log-zone 1 to the log-zone 0. This log-zone rotation continues the same by invalidating all the log data in the log-zone 1.

The bitmaps of one logical zone could present if the page-mapping logs are valid. Each zone has its bitmaps that represent which segment is whether up-to-date or not. If any page mappings in a segment change, that page-mapping change adds (or updates) to the page-mapping log table, and the corresponding segment bit of a zone is set, such as LPN 0 of Segment 0 and LPN 4096 of Segment 8. On the other hand, after the segments’ collection write, the corresponding segments’ bits in two zones are unset. Figure 2 also depicts the bit 0s representing the invalidation of Segment 1, 4, and
9. If all segments those bits set before are written, all the page-mapping logs of the log zone 0 expire.

The smaller segment could gather more changed and up-to-date page mappings. However, it needs more space to keep the segment directory table [6] and all the bitmaps of the zones. The segment directory table contains PPNs of valid segments. Because a bitmap represents one segment, an 8-KiB segment needs 2-KiB space for 128-GiB storage. Instead, a 2-KiB segment requires 8-KiB space. However, the smaller segment could increase the page-mapping invalidation ratio. In the evaluation results, the least segment remarkably nullifies the logs and improves the page-mapping consistency.

3. Evaluation

3.1 Experimental Setup

The experiment measures the amount of NAND page write (i.e., programming) to invalidate page-mapping logs. The evaluation utilizes a trace-driven FTL simulator that can handle the block-level I/O requests commonly used in a host file system. To compare the results, each page-level FTL adopts its segment, such as 2 KiB, 4 KiB, or 8 KiB. Each is called 8S, 4S, and 2S, respectively, used as labels in the evaluation result’s graph. The 0S case means that it is the same as one 16-KiB NAND page following the NAND flash memory specification [19]. This case implements a legacy page-level FTL. All the page-level FTLs implement 128-GiB NAND flash-based storage with a 4-KiB page-mapping level.

At first, the experiment utilizes synthesized address patterns giving variations for page mappings changed. Those patterns are data writes composed of sequential and random address patterns. Lastly, the enterprise server’s workloads are used. They were captured from enterprise storage traffics on the enterprise virtual desktop infrastructure (VDI) [20]. The page-level FTLs allocate 16-KiB volatile memory space for the page-mapping log table. That table size is identical to one NAND page of the NAND flash memory.

3.2 Synthesized Pattern Analysis

(1) Sequential Write Results

Sequential data writes fill up the entire storage with 16-KiB chunk requests. After data writes, the page-level FTL updates the page mappings of the PMT and adds the page-mapping changes to the page-mapping log table. The changes contain LPNs and PPNs where the page-level FTL writes to the NAND flash. Because the first filling writes cannot invalidate all the logs of the first zone, the sequential data writes twice as much as the entire capacity.

Since one PMT can include all the page mappings changed by the sequential data writes, gathering the segments set in the zone does not affect to increase the page-mapping consistency. As a result, the write amounts to invalidate the logs are nearly same among all the page-level FTLs. However, because of the segment directory table increase, each segment case increases the total write amount by −0.7%, 0.9%, and 4.0% of the legacy page-level FTL. Figure 3 compares the sequential write results for each segment.

(2) Random Write Results

The second experiment evaluates random writes to inspect the effect of segment collection. The chunk size is small 4 KiB, and these small-random addresses are composed of a fraction of the entire LPN. These random patterns might have less possibility to update only one PMT. The x-axis of Fig. 4 indicates the fraction ratio from 5% to 30%. The total amount of write requests is the total storage capacity. Figure 4 shows the data write amounts that were needed to invalidate the page-mapping logs. Those amounts include the write amount of the segment directory. Because different addresses are less in the lower ratios, those written amounts decrease naturally. All the page-level FTL results show this decreasing tendency. Because of the small-random writes, the page mapping changed could have a longer distance in the lower fraction ratio. This kind of distance gives a higher chance to gather the segments. The
page-level FTL with the smaller segment needs the smallest amount to recover the page mapping consistency. The 8S case only requires 21.8% of the legacy page-level FTL as the result of the 5% fraction ratio.

3.3 Real-Device Workload Result Analysis

The final experiment deploys the block-level write requests of real-device SSDs. They are I/O traces logged from three SSD devices of the enterprise VDI servers. The x-axis of Fig. 5 describes each trace with a logical unit number (LUN). The evaluation assumes that the event of a power cycle occurs once a day.

If the power cycle happens, the page-level FTL would complete the PMT reconstruction at the start-up by using both the PMT blocks and the page-mapping log blocks. Because the page-mapping log blocks might contain more up-to-date page mappings than the PMT blocks, the FTL should exploit those page-mapping logs to reconstruct PMTs. By the way, if the FTL has stored those page mappings to the PMT blocks, the FTL could minimize this kind of recovery overhead. This log deactivation operation might work through special commands, such as a shutdown or a power-off notification command [8], [21], [22]. The evaluation results calculate the deactivation cost by reflecting the bitmaps’ sets of two zones. Clearing one bitmap set at least requires one number of NAND write.

The segment collection could increase the invalidating ratio for the page-mapping logs in their log blocks. In real-device workload traces holding many small random writes, the smaller segments show better results. The page-level FTL assembling eight segments in one NAND page brings the best result displayed on Fig. 5. As a result, that segmented-collection effect reduces the deactivation cost by 43.7%, 79.1%, and 45.9% of the legacy page-level FTL in LUN0, LUN1, and LUN2, respectively.

4. Conclusion

Every FTL organizes its mapping data to complete storage I/O services. However, mapping data amounts could be enormous in close observation of page-mapping changes, and those mapping changes could have any distances. In this letter, the virtually-shrunk segment enables the page-level FTL to extend its page-mapping management. Gathering the smaller different segments might invalidate the more page-mapping logs previously written by the page-mapping log structure. Consequently, our novel plan to the page-level FTL improves the page-mapping recovery cost while enhancing the page mapping consistency.

References

[1] J. Kim, J.-M. Kim, S.-H. Noh, S.-L. Min, and Y. Cho, “A space-efficient flash translation layer for compactflash systems,” IEEE Trans. Consum. Electron., vol.48, no.2, pp.366–375, Aug. 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, July 2007.
[3] D. Jung, J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, “Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme,” ACM Transactions on Embedded Computing Systems (TECS), vol.9, no.4, pp.1–41, 2010.
[4] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings,” In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLS’09), 2009.
[5] 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.5-6, pp.332–343, May 2009.
[6] J.-H. Kim and Y.-S. Lee, “Exploiting exclusive higher resolution to enhance response time of embedded flash storage,” Proceedings of the 15th International Conference on Software Technologies, pp.470–477, 2020.
[7] K. Whitaker, “Embedded multimedia card (eMMC) eMMC/card product standard, high capacity, including reliable write, boot, and sleep modes,” 2015.
[8] J. Standard, “Universal flash storage (UFS) card extension standard,” 2018.
[9] G. Goodson and R. Iyer, “Design tradeoffs in a flash translation layer,” Proceedings of Workshop on the Use of Emerging Storage and Memory Technologies, Citeseer, 2010.
[10] D. Ma, J. Feng, and G. Li, “LazyFTL: a page-level flash translation layer optimized for flash memory,” Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp.1–12, ACM, June 2011.
[11] C.-H. Wu and H.-H. Lin, “Timing analysis of system initialization and crash recovery for a segment-based flash translation layer,” ACM Transactions on Design Automation of Electronic Systems (TODAES), vol.17, no.2, pp.1–21, 2012.
[12] S. Moon, S.-P. Lim, D.-J. Park, and S.-W. Lee, “Crash recovery in FAST FTL,” IFIP International Workshop on Software Technologies for Embedded and Ubiquitous Systems, pp.13–22, Springer, 2010.
[13] C. Zhang, Y. Wang, T. Wang, R. Chen, D. Liu, and Z. Shao, “Deterministic crash recovery for NAND flash based storage systems,” Proceedings of the 51st Annual Design Automation Conference, DAC’14, New York, NY, USA, pp.1–6, ACM, June 2014.
[14] H.A. Khouzani and C. Yang, “Towards a scalable and write-free multi-version checkpointing scheme in solid state drives,” 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp.37–48, IEEE, 2016.
[15] S.-H. Lim, “Implementation of metadata logging and power loss recovery for page-mapping FTL,” IEICE Electron. Express, vol.10,
[16] S.C. Tweedie et al., “Journaling the linux ext2fs filesystem,” The Fourth Annual Linux Expo, Durham, North Carolina, 1998.

[17] R. Mativenga, P. Hamandawana, T.-S. Chung, and J. Kim, “FTRM: A cache-based fault tolerant recovery mechanism for multi-channel flash devices,” Electronics, vol.9, no.10, p.1581, 2020.

[18] K. Sun, S. Back, J. Choi, D. Lee, S.H. Noh, and S.L. Min, “LTFTL: lightweight time-shift flash translation layer for flash memory based embedded storage,” Proceedings of the 8th ACM international conference on Embedded software, pp.51–58, 2008.

[19] Samsung, “Samsung V-NAND technology,” 2014.

[20] C. Lee, T. Kumanono, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, “Understanding storage traffic characteristics on enterprise virtual desktop infrastructure,” Proceedings of the 10th ACM International Systems and Storage Conference, SYSTOR ’17, p.13, ACM, May 2017.

[21] K. Marks, “An nvme express tutorial,” Flash Memory Summit, 2013.

[22] H. Vuong, “Mobile memory technology roadmap,” JEDEC Mobile Forum, 2013.