An Advanced SLC-buffering for TLC NAND Flash-based Storage

Kirock Kwon, Dong Hyun Kang, and Young Ik Eom

Abstract — **In recent years, almost all consumer devices adopt NAND flash storage as their main storage, and the performance and capacity requirements for the storage components are increasing. To meet the requirements, many researchers and manufacturers have focused on combined SLC-TLC storage, which consists of high-speed single-level cell (SLC) and high-density triple-level cell (TLC) memory. In this paper, we re-design the internal structure of the combined SLC-TLC storage to improve its performance and lifetime. We also add new behavior by employing the I/O characteristics of the file system journaling to efficiently manage the SLC region inside the storage. We implemented our scheme on a real storage platform and compared its performance with those of the conventional techniques. The results of our evaluation show that our scheme improves the storage performance by up to 65% relative to the conventional techniques.**

Index Terms — **file system journaling, NAND flash, combined SLC-TLC storage, SLC buffer, TLC NAND, FTL, Dual mode program**

I. Introduction

Flash storage has become popular as storage media for smartphones, smart TVs, tablet PCs, and other consumer devices because it fulfills almost all requirements for consumer devices including a low power consumption, small size, and shock resistance. In addition, advanced flash technologies, such as multi-level cell (MLC) and triple-level cell (TLC) configurations, have helped remedy the cost constraints of consumer devices by allowing to store two or three bits of data in each cell [1], [2]. Unfortunately, higher density in flash memory has a negative impact on the performance and endurance of the devices because each cell must handle more voltage states [3], [4]. To efficiently address these side effects, some researchers have focused on developing devices composed of both single-level cell (SLC) and MLC storage components, which is named *combined SLC-MLC storage*. They categorized incoming data from the host into hot and cold one to exploit the benefits of the SLC performance and MLC capacity [5]-[7]. In recent years, several manufacturers proposed *combined SLC-TLC storage*, consisting of SLC and TLC storage components, and they addressed the lower performance of the TLC by employing SLC as a cache for incoming data [8]. In other words, data from the host is temporally cached in SLC and is then migrated from the SLC to TLC using the cache policy. However, such an approach significantly decreases the storage performance when the SLC buffer needs to reclaim free space.

Meanwhile, many researchers have studied storage I/O stacks on consumer devices with flash storage, such as eMMCs and SD Cards, and they have proposed several solutions to improve the performance and endurance of flash storage. Some researchers focused on the relationship between the lifetime of data on the file system and the data placement inside the flash storage because the lifetime and placement of data are correlated with the performance and endurance of the flash storage. For example, Kang et al. [9] re-designed the internal structure of the special type of flash storage, multi-streamed solid-state drive (SSD), to separate incoming data into streams according to their life cycle on the file system. Some studies have adopted non-volatile memory (NVM) technologies to independently store frequently updated data, such as journal data or logging data [10]-[12].

In this paper, we also target combined SLC-TLC storage and propose a novel scheme that efficiently improves the performance and endurance of flash storage without any modifications at the host level. The contributions of this paper are as follows:

- We first study the I/O traffic in the journaling mechanism of EXT4 [13] file system and then analyze the I/O characteristics and patterns based on the commercial flash storage.
- We re-design the internal structure of the flash storage and add new internal behavior to reduce the overhead for both garbage collection (GC) and data migration from SLC to TLC.
- We implement our scheme on a real storage platform to compare it to using a conventional approach. The results of the evaluation show that our scheme improves the performance by up to 65% relative to the conventional one.

Manuscript received September 27, 2017; accepted November 15, 2017. Date of publication December 19, 2017. This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2015M3C4A7065696) and this work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. NRF-2017R1A2B3004660). *(Corresponding author: Y. I. Eom.)*

Kirock Kwon is with Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon, South Korea (e-mail: krkwon@skku.edu).

Dong Hyun Kang is with Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon, South Korea (e-mail: kangsu@skku.edu).

Young Ik Eom is with Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon, South Korea (e-mail: yieom@skku.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCE.2017.015070

The rest of this paper is organized as follows. In Section II, we present the background needed to understand our scheme. We describe in detail the design and implementation of our scheme in Section III, and then present our experimental results on a real hardware platform in Section IV. The related work is described in Section V. Finally, Section VI concludes this paper.

II. Background and Motivation

In this section, we classify the NAND flash memories and identify the advantages and disadvantages of each type of memory technology. Then, we study the journaling mechanism of EXT4 file system to understand our scheme in detail. Finally, we briefly describe our motivation.

A. Classification of NAND flash memories

In general, NAND based flash memories can be classified into three program modes: SLC, MLC, and TLC. The mode of flash memory is determined by the number of bits that can be stored in a single cell. As is well-known, a denser the program mode provides a higher capacity in the flash memory at the same manufacturing costs. Unfortunately, a denser program mode is considered to have a negative impact in that it significantly reduces the performance and endurance of the flash memory.

To remedy the disadvantages of using TLC, some manufacturers have started to commercialize hybrid flash storage composed of SLC and TLC program modes [14], [15]. The storage employs flash memory in the SLC mode as a buffer for the TLC to temporarily handle incoming data as fast as possible. Then, all of the data in the SLC buffer is simultaneously migrated to the flash memory of the TLC program mode during idle time. However, this type of storage operates at the performance levels of TLC if there is no free space in the SLC buffer.

To clearly understand the performance gap in real world systems, we measured the write throughput on the commercial SSD consisting of hybrid flash storage. In this test, we employed FIO microbenchmark [16] to continuously issue write operations to the underlying flash storage. Fig. 1 shows the results of our evaluation. As expected, the performance of the storage suddenly drops at a time of 20s because the storage handles incoming data with the flash storage in the TLC program mode when the SLC buffer becomes full. In addition, the decrease in the performance of the storage does not improve after the performance drop until there is idle time. Therefore, write requests coming from the host after that time are handled 3 times slower compared to those at the initial time.

B. File system journaling

Journaling is one of the common approaches to guarantee data consistency of the file system such as EXT4 and XFS [17]. However, to maintain consistency of the file system, journaling should write the same data twice: once where the data is stored into the journal area, and once where the data is stored into the original area [18]-[20]. In addition, write requests stored in the journal area are always issued with synchronous options, O_SYNC [18], so journaling has a huge impact on storage performance. To address this impact, the file systems provide several journal modes. For example, EXT4 file system offers three kinds of journal modes: write-back, ordered, and data journal mode [18]-[20]. There is a fundamental tradeoff between the journaling overhead and the consistency level. Ordered mode only guarantees the metadata consistency of the file system by writing the metadata of the file system twice. On the other hand, data journal mode writes the metadata and data on the file system twice to fully ensure consistency in the file system.

In order to study the journaling behaviors in further detail, we captured I/O requests from the host to the underlying flash storage. Fig. 2 shows the I/O patterns captured by running the FIO microbenchmark on a real flash storage. In Fig. 2, we can see that journal data blocks are sequentially written to the journal area (shown as black dots in the LBA range 590-592), so we can easily find the journal area in the file system because the journal area is reserved at the time of format and is re-used in a round-robin way.

C. Motivation

If a consumer device uses hybrid flash storage as main storage media, the device cannot avoid the drop in performance shown in Fig. 1. Unfortunately, the device may take a long time to store journal data because write operations incurred by journaling may be handled with the flash storage in the TLC program mode due to a semantic gap between the file system

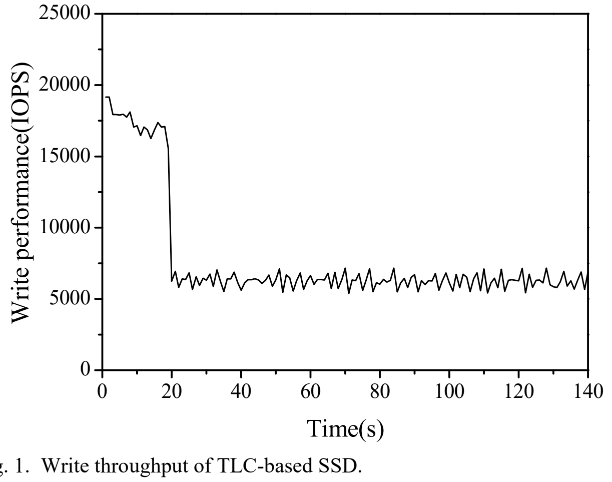

Fig. 1. Write throughput of TLC-based SSD.

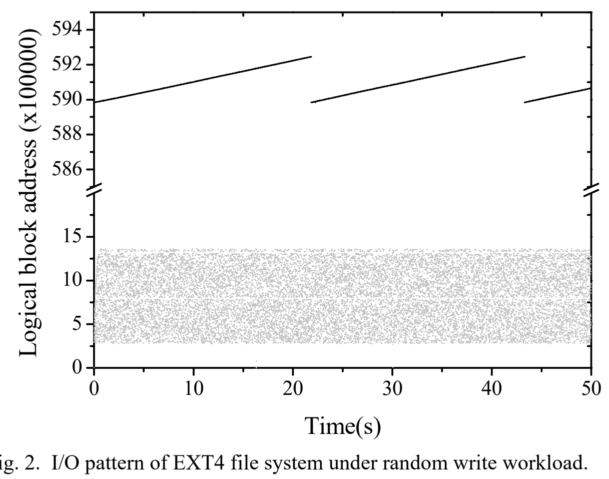

Fig. 2. I/O pattern of EXT4 file system under random write workload.

and the flash storage. This unstable performance has motivated us to re-design the internal structure of the flash storage and to propose our scheme.

III. Design and Implementation

In this section, we first describe the details of our scheme for NAND flash-based storage consisting of an SLC region where data is programmed in the SLC program mode and a TLC region where data is programmed in the TLC program mode. We then present the details of our implementation.

A. *SLC buffering & journal write*

Unlike other approaches using an SLC write buffer, our scheme divides the SLC write buffer into *journal region* and *buffer region*. This is to exploit the write pattern of the file system journaling as explained in Section II.

Our scheme classifies host data into journal data (J_{data}) and normal data (N_{data}) by using journal detector, and it writes them into the journal region or the buffer region according to the classification of the journal detector; J_{data} is the data requested by the journal daemon and N_{data} is the other data.

The file system journaling requests journal writes in a round-robin way. Thereby, J_{data} is written into the journal region sequentially. Since NAND flash memory does not support an in-place update, our scheme needs to append the updated J_{data} to a free block and then invalidates the old J_{data}. Therefore, the old J_{data} can be completely invalidated by the updated J_{data}. Consequently, our scheme can produce completely invalid blocks that can be used as free blocks. In this way, our scheme can constantly write J_{data} to the journal region without any other process, such as garbage collection and data migration.

Fig. 3 shows an example comparing our scheme to that using the conventional technique. The physical blocks (PBs) in the figure represent the blocks of the SLC region, and each block consists of four pages. N and J denote normal data and journal data, respectively, and the numbers in parentheses indicate the logical block address (LBA) of the data. For simplicity, the journal area of the file system is assumed to be fixed to LBA 0 to 3 when the storage is formatted. Fig. 3(a) shows a case in which write operations are performed without separating the journal writes and normal writes, which is referred to as *baseline*. As is shown in the figure, some data was already written to PB10 and PB11 according to the pre-condition sequence. After some new data has been written to PB12 according to the write sequence #1, the two pages on PB10 are invalidated by the updated J(0) and J(1), as shown in snapshot 1. In the same way, after another new data is written to PB13 according to the write sequence #2, the two pages on PB11 are also invalidated by the updated J(2) and J(3) on PB13, as shown in snapshot 2. As a result, there is no free block to accommodate subsequent write requests. Therefore, the baseline requires additional operations, such as garbage collection and data migration, to reclaim free space. On the other hand, Fig. 3(b) shows write operations in the case of our scheme. Here, journal data and normal data are written separately in PB10 and PB20 according to the pre-condition sequence, and the journal data is placed in a sequential pattern. As is shown in snapshot 1 and 2, some data are written to PB11 and PB21 according to the write sequences #1 and #2. As a result, all pages of PB10 are invalidated by sequentially updated journal data [i.e., J(0-3)]. Therefore, our scheme can reclaim a free block by erasing PB10. In this way, our scheme can constantly reclaim free blocks for journal writes.

B. *TLC direct write & migration from SLC buffer to TLC region*

Typically, to reclaim space in the SLC buffer, the data written into the SLC buffer must migrate to the TLC region during the idle time. However, if the SLC buffer is exhausted due to a heavy workload without a sufficient idle time to migrate the data, there can be no free space for write requests in the SLC buffer. Therefore, in that case, data is written directly to the TLC region. As a result, the write performance may drastically drop from the SLC performance to the TLC performance. On

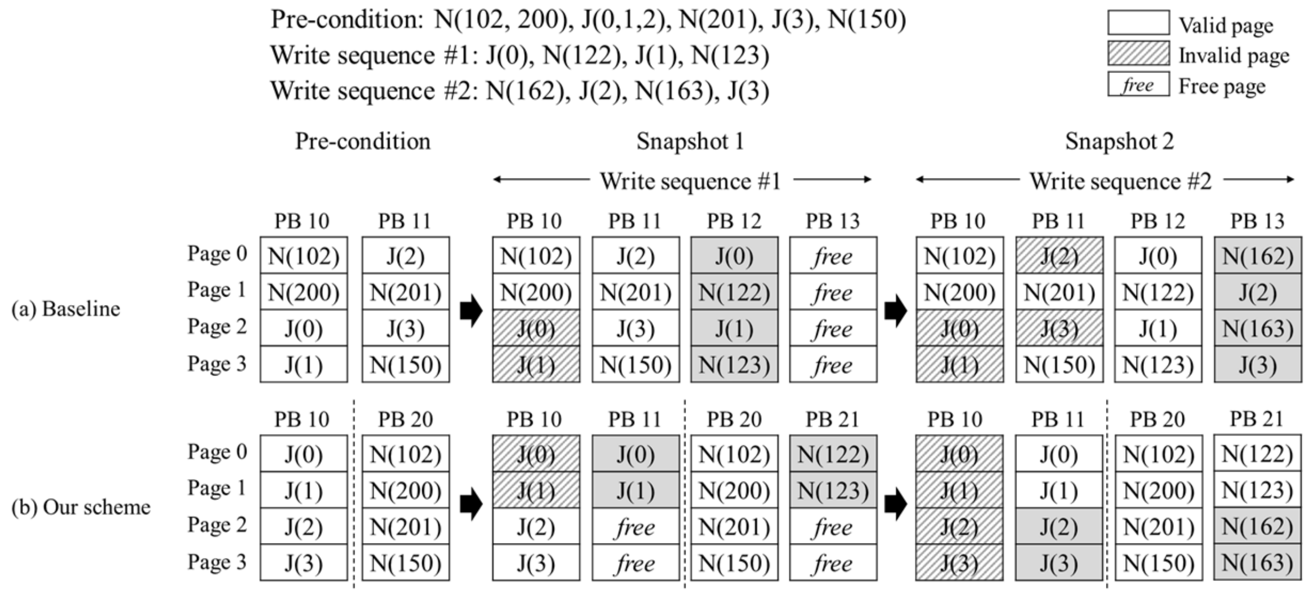

Fig. 3. Example of write operations. (a) and (b) shows the write operation without and with separating journal writes and normal writes, respectively.

the other hand, in the proposed scheme, only N_{data} is written directly to the TLC region because our scheme can constantly write J_{data} to the journal region, as was described before, even when the SLC buffer has been exhausted. Therefore, our scheme guarantees that journal data will constantly be written with a high performance.

In addition, the amount of migration to the TLC region can be reduced by separating the journal writes and the normal writes. As is shown in Fig. 3(a), the baseline has to migrate all valid data in 12 pages included in 4 blocks [i.e., N(102, 200, 201, 150, 122, 123, 162, 163) and J(0, 1, 2, 3)]. On the other hand, as is shown in Fig. 3(b), our scheme migrates only N_{data} in 8 pages included in 2 blocks [i.e., N(102, 200, 201, 150, 122, 123, 162, 163)]. As a result, our scheme can improve the lifetime of the TLC region by reducing the amount of data migrated there.

C. *SLC buffer on SSD development board*

We implemented our scheme on an SSD development board [21]. Since the board is composed of MLC NAND flash, we also emulated the combined SLC-TLC storage on the platform. Fig. 4 illustrates the write operation path of our scheme using a simplified block diagram.

When the host issues write requests to the board, the *journal detector* distinguishes journal writes by adopting a heuristic method that detects the sequential large loop characteristics of the journal writes [22], and it sets a flag that indicates whether the requests are journal writes or normal writes. The I/O requests issued by the host are queued to the FIFO queue, called *event queue*, along with the flag set by the journal detector, and it is then dequeued for internal operations by the flash translation layer (FTL).

SLC region manager manages the SLC region, such as by conducting address mapping and SLC-to-TLC migration, as was explained in earlier sections.

In order to emulate the combined SLC-TLC storage, we added *NAND mode changer*, which is a special operation in the

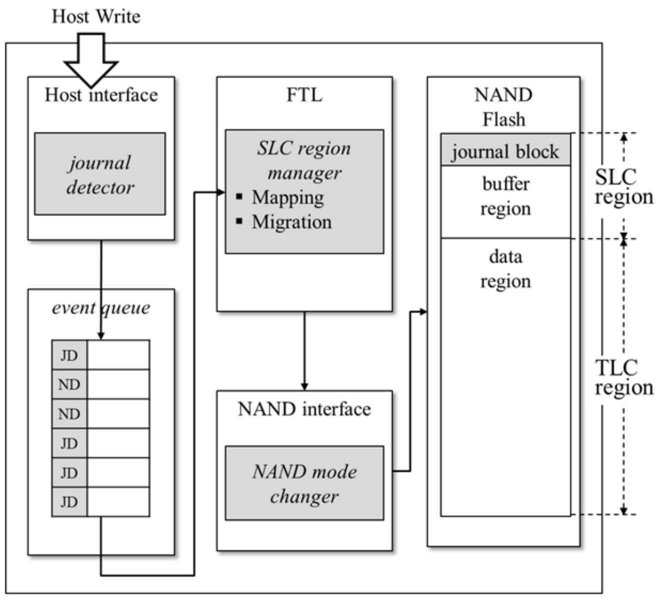

Fig. 4. Simplified write path of our combined SLC-TLC storage.

NAND flash controller, to change the write mode to the SLC-mode or TLC-mode according to the physical block address of the request. We also emulated the SLC-mode and TLC-mode program by adding an appropriate delay to the NAND program operation.

The board has a total capacity of 64GB. As a baseline, we set an SLC region of 640MB and a TLC region for the remainder of the total capacity. Compared to this baseline, our scheme allocated a journal region of 128MB, which is the same size as the journal area of EXT4 file system, on the SLC region.

IV. EXPERIMENTS

In this section, we first describe our experimental environment. Then, we present the results obtained using our development board and compare the performance of our scheme to the conventional one.

A. *Experimental setup*

In order to evaluate the performance of our scheme, we conducted the evaluation on a PC with a 3.40GHz CPU and 8GB RAM. The system ran on Linux kernel version 4.5.4. We compared our scheme to the baseline using EXT4 file system for both ordered and data journaling modes.

We conducted experiments using synthetic and real-world workloads. We used the FIO benchmark to generate synthetic workloads and also used Filebench [23] and Sysbench [24] to generate various real-world workloads.

Our experiments address the following questions:

- How effectively does our scheme manage journal writes compared to the baseline?
- How much can our scheme reduce the latency of the journal writes?
- How effective is our scheme in terms of performance?
- How effective is our scheme in terms of endurance?

B. *Management of journal writes*

In order to answer the first question, we compared the performance by varying the amount of journal writes. We used 8KB random write workloads generated by the FIO benchmark and invoked fsync() [18] call after every 2 to 128 write requests to vary the amount of journal writes. In order to measure both the SLC buffer performance and the TLC direct write performance, we adjusted the amount of writes to twice the size of the SLC region.

Fig. 5 shows the breakdown of the host writes for J_{data} and N_{data}. The y-axis represents the proportion of J_{data} and N_{data}, and the x-axis is the interval of fsync() call. As shown in the figure, the proportion of J_{data} increases as the interval of fsync() call is shortened. In other words, the amount of J_{data} increases with a greater frequency of fsync() calls. This figure also shows that the proportion of the J_{data} is higher in data mode than in ordered mode, and these results can be explained as follows: (1) Every time fsync() call is invoked, the dirty pages of the file given as an argument are flushed to the storage, resulting in journaling. Therefore, the amount of J_{data} increases as fsync() call is issued more frequently. (2) Ordered mode records only metadata in the journal area, whereas data mode records both file data and metadata in the journal area. Therefore, the amount of J_{data} is larger in data mode than in ordered mode.

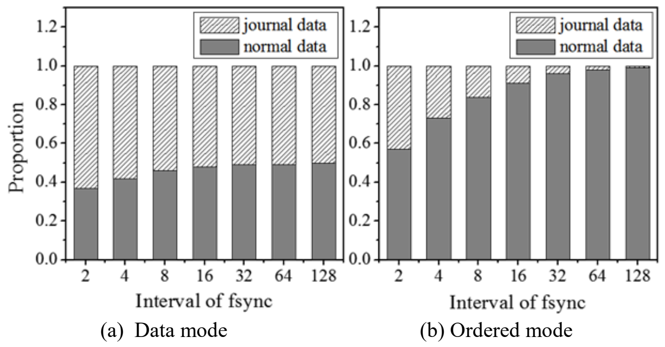

Fig. 5. Breakdown of host writes by varying the interval of fsync() calls.

Fig. 6 compares the performance of our scheme with that of the baseline. This figure shows an improvement in performance in both modes as the interval of fsync() call is shortened. Overall, our scheme can improve the performance in data mode and ordered mode by up to 102% and 97%, respectively. This is because the frequent fsync() calls increase the amount of J_{data} that will be written to the high-speed SLC region.

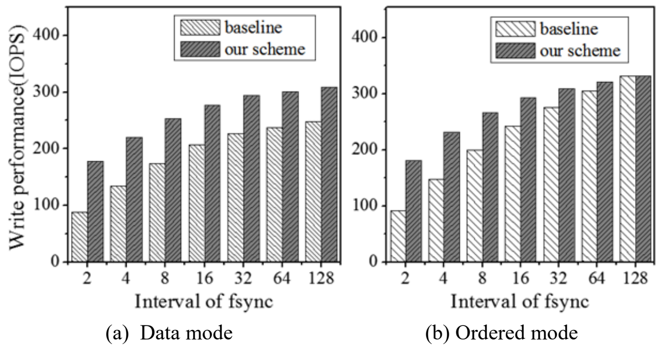

Fig. 6. Performance comparison by varying the interval of fsync() calls.

We also conducted performance profiling to improve our understanding. Fig. 7 compares the performance profile of our scheme to that of the baseline: the y-axis is a random write performance under an FIO workload and the x-axis is the time in seconds. In this figure, there are two periods that have a significant performance gap: (i) SLC buffering period and (ii) TLC direct write period. During the SLC buffering period, the data is written to the SLC buffer, which shows a high performance. When the SLC buffer is full, the data is written to the TLC region, resulting in a drop in performance. As shown in the figure, our scheme can increase the SLC buffering period by 169% and 84% in data mode and ordered mode, respectively. These results clearly show that the free blocks for the journal data can be constantly reclaimed by placing the journal data together on the journal region. In addition, our scheme provides a good effect on the TLC direct write period. When the SLC buffer is full, both journal data and normal data are written to the TLC region in the case of the baseline, whereas our scheme can constantly write the journal data to the journal region in the SLC region. As a result, compared to the baseline, our scheme can improve the average performance of the TLC direct write period by 43% and 45% in data mode and ordered mode, respectively.

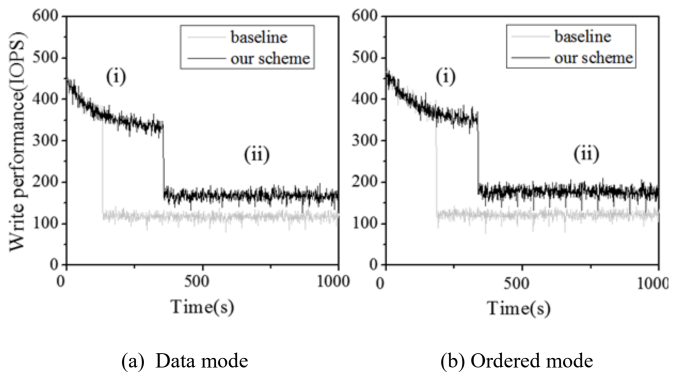

Fig. 7. Throughput for baseline and our scheme under FIO random write workload.

C. Journal write latency

We measured the latency of the journal writes to answer the second question. Fig. 8 compares the latency of the journal writes under the FIO random write workload. Fig. 8(a) and (b) show cumulative distribution functions (CDFs) of the journal write latency. The y-axis represents the latency, and the x-axis is the cumulative percentage. As shown in the figure, our scheme has a lower journal write latency compared to the baseline. Since the journal writes are performed with the synchronous option, the latency of the journal writes is directly affected by the performance of the storage. Therefore, these results can be clearly explained as follows: (1) in the case of baseline, the journal data is first written to the SLC buffer and is then written directly to the TLC region after the SLC buffer is full, whereas (2) our scheme can reduce the latency by constantly writing the journal data to the journal region in the SLC region because the free blocks for the journal writes can be constantly reclaimed.

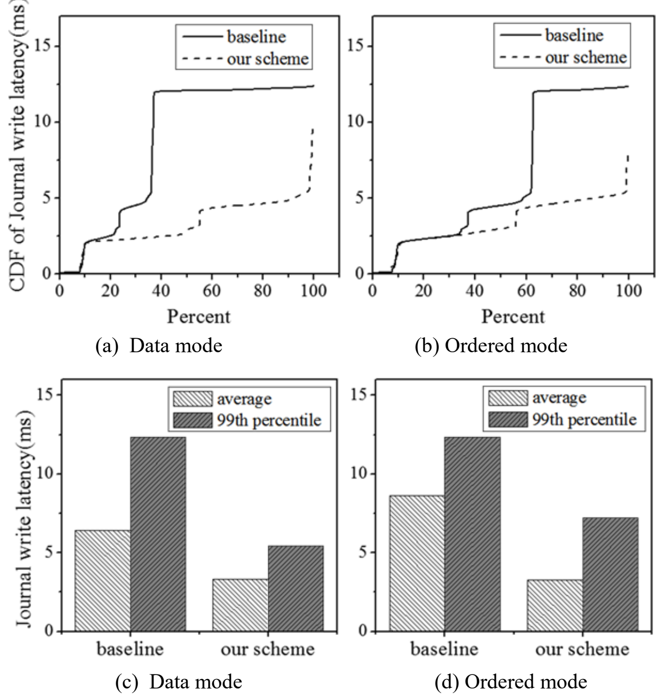

Fig. 8. Comparison of journal write latency under FIO random write workload. (a) and (b) shows latency CDFs, and (c) and (d) shows the average and 99th percentile latencies.

As shown in Fig. 8(c) and (d), our scheme can reduce the average and the 99th percentile latency by 48% and 56% in data mode, and by 62% and 41% in ordered mode, respectively.

D. Performance results

Fig. 9 and Fig. 10 answer the third question. We used Filebench and Sysbench to compare the performance for various real-world workloads. The characteristics of these workloads are summarized in Table 1. Fig. 9 shows the performance of our scheme compared to the baseline under the Filebench workloads. Our scheme performed better than the baseline under all workloads. In particular, the improvement in performance of the varmail workload is the greatest, which is 65% and 41% in data mode and ordered mode, respectively, because the varmail workload issues fsync() call more frequently than others, and more J_{data} is thereby written to the storage (see Table 1). Fig. 10 shows the performance of OLTP workload using Sysbench. Data and ordered modes improved the performance by 40% and 16%, respectively.

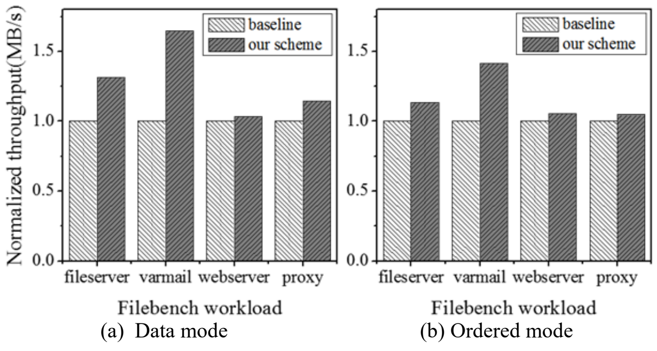

Fig. 9. Performance comparison under the Filebench workloads.

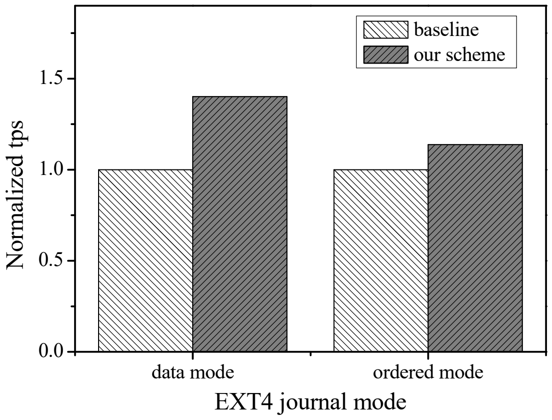

Fig. 10. Performance comparison under the OLTP workload.

E. Endurance

Our scheme can improve the endurance of the storage by reducing the amount of writes to the TLC region. Data is transferred to the TLC region in two ways: migration form the SLC region to the TLC region and a direct write to the TLC region. Therefore, the amount of data written to the TLC region is the sum of the amount of data migrated from the SLC region and written by TLC direct writes.

TABLE I
WORKLOAD CHARACTERISTICS

Workload	R : W ratio	JD:ND ratio	
		Data mode	Ordered mode
Fileserver	1 : 2	1.2 : 1	1: 34
Varmail	1 : 1	5.8 : 1	1: 1.2
Webserver	10 : 1	1.1 : 1	1: 660
Proxyserver	5 : 1	3.6 : 1	1 : 5.3
OLTP	2.8 : 1	1.1 : 1	1 : 15

In order to answer the last question, we measured the amount of data written to the TLC region. As shown in Fig. 11, our scheme can dramatically reduce the amount of writes to the TLC region. The reasons for this result are as follows: (1) our scheme can reduce the data migration from the SLC region to the TLC region by dividing the SLC region into the journal region and the buffer region; (2) our scheme can reduce the TLC direct writes by constantly writing J_{data} to the journal region in the SLC region. In summary, since all J_{data} is absorbed in the journal region, J_{data} is not transferred to the TLC region. As a result, our scheme efficiently improves the lifetime of the storage.

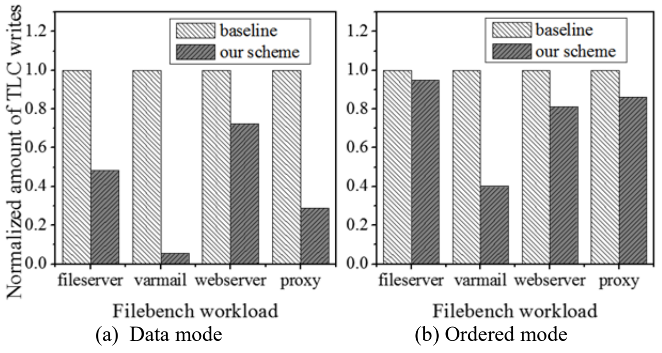

Fig. 11. Comparison of the amount of data transferred to the TLC region under the Filebench workloads.

V. RELATED WORK

Over the past decade, with the development of multi-level cell technology, the price of NAND flash storage has continuously been decreased, and the market share of the MLC/TLC NAND storage has continuously been increased. This is because of two reasons: first, flash technology has significantly been improved by the manufacturer, and second, there has been much research to overcome the disadvantages of using MLC/TLC technology, such as the lower performance and lifetime. In this section, we present the related works that have been proposed to overcome the disadvantages of MLC/TLC technology.

Several studies have been performed on FTL and the file system level to overcome the low performance and low lifetime of MLC NAND storage [5]-[7]. Im et al. [5] and Murugan et al. [6] conducted FTL-level research that selectively writes hot data to the SLC region. Lee et al. [7] proposed a file system for combined SLC-MLC storage that uses the SLC region as a write buffer. These techniques focused on data classification based on hotness and efficiently improved the overall storage performance by writing frequently-updated data into the SLC region. Recently, various studies [25]-[27] proposed techniques suitable for TLC NAND storage to alleviate its lower performance and endurance compared to that of MLC NAND storage. Yao et al. [25] and Yang et al. [26] proposed FTLs for combined SLC-TLC storage to improve the performance and lifetime. Yao et al. proposed a workload-aware FTL to improve the lifetime of the storage by balancing the lifetime of the SLC and TLC region. Yang et al. proposed a utilization-aware self-tuning design to improve the performance by dynamically adjusting the size of the SLC and TLC region. The previous version of this paper [27] also improved the lifetime of combined SLC-TLC storage by eliminating unnecessary data migrations caused by the semantic gap between the log-structured file system and the storage. This paper improves upon previous schemes to improve both performance and lifetime of the storage by exploiting the characteristics of the file system journaling.

Other researchers focused on host I/O behavior to improve the performance of the system and storage. Jeong et al. [28] studied the synchronousness of I/O and suggested QASIO scheme to improve the responsiveness of the file system operations. Kim et al. [29] focused on I/O priority inversion on the I/O path and introduced a new I/O prioritization scheme to improve the system performance. Kang et al. [9] proposed multi-streamed SSD that places data with similar lifetime together to reduce garbage collection overhead. The multi-streamed SSD differs from ours in that it requires explicit information regarding the data lifetime from the host. In order to reduce the overhead of journaling, some researchers have adopted non-volatile memory (NVM), which has fast and byte-addressable characteristics [10]-[12]. For example, Lee et al. [10] used NVM as the union of the buffer cache and journal area. They proposed in-place commit scheme to reduce the number of storage accesses by making pages in the buffer cache persistent. Kim et al. [11] proposed delta journaling scheme for the NVM-NAND hybrid architectures. They effectively reduced journaling overhead by storing journal data as a compressed delta in the NVM cache. These studies are different from this paper because they require S/W or H/W changes in the host. They are also not optimized for the TLC storage discussed in this paper.

VI. Conclusion

In this paper, we present an efficient scheme to improve the performance and lifetime of combined SLC-TLC storage. Specifically, we reserved dedicated space for the journal data on the high-speed SLC region to employ the I/O characteristics of the file system journaling. We separated the journal data from the host writes to reduce the overhead of the garbage collection and data migration, and we wrote the journal data on the reserved SLC region to reduce its synchronous journal write latency. We implemented our scheme on a real storage platform and conducted a series of experiments with various workloads. The experimental results show that our scheme significantly improves the performance and lifetime of the storage.

References

[1] S.-H. Shin, D.-K. Shim, J.-Y. Jeong, O.-S. Kwon, S.-Y. Yoon, M.-H. Choi, T.-Y. Kim, H.-W. Park, H.-J. Yoon, Y.-S. Song, Y.-H. Choi, S.-W. Shim, Y.-L. Ahn, K.-T. Park, J.-M. Han, K.-H. Kyung, and Y.-H. Jun, "A new 3-bit programming algorithm using SLC-to-TLC migration for 8 MB/s high performance TLC NAND flash memory," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, Jun. 2012, pp. 132-133.

[2] K. Park, M. Kang, D. Kim, S. Hwang, B. Choi, Y. Lee, C. Kim, and K. Kim, "A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLC NAND flash memories," *IEEE J. Solid-State Circuits*, vol. 43, no. 4, pp. 919-928, Apr. 2008.

[3] J. Jeong, S. S. Hahn, S. Lee, J. Kim, J. Jeong, S. S. Hahn, S. Lee, and J. Kim, "Lifetime improvement of NAND flash-based storage systems using dynamic program and erase scaling," in *Proc. USENIX FAST*, Santa Clara, CA, Feb. 2014, pp. 61-74.

[4] A. Maislos, "A New Era in Embedded Flash Memory," in *Flash Memory Summit 2011*, Santa Clara, CA, Aug. 2011.

[5] S. Im and D. Shin. "ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer," *J. Systems Architecture*, vol. 56, no. 12, pp. 641-653, Dec. 2010.

[6] M. Murugan and D. Du, "Hybrot: Towards improved performance in hybrid SLC-MLC devices," in *Proc. IEEE 20th MASCOTS*, Arlington, VA, Aug. 2012, pp. 481-484.

[7] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, "FlexFS: A flexible flash file system for MLC NAND flash memory," in *Proc. USENIX ATC*, San Diego, CA, Jun. 2009, pp. 115-128.

[8] Samsung, "Samsung solid state drive TurboWrite technology white paper," Samsung Electronics Co, Tech. Rep., 2013.

[9] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive," in *Proc. 6th USENIX HotStorage*, Philadelphia, PA, Jun. 2014, pp. 13-17.

[10] E. Lee, H. Bahn, and S. H. Noh. "Unioning of the buffer cache and journaling layers with non-volatile memory," in *Proc. USENIX FAST*, San Jose, CA, Feb. 2013, pp. 73-80.

[11] J. Kim, C. Min, and Y. I. Eom, "Reducing excessive journaling overhead with small-sized NVRAM for mobile devices," *IEEE Trans. Consumer Electronics,* vol. 60, no. 2, pp. 217-224, Jun. 2014.

[12] D. H. Kang, and Y. I. Eom, "FSLRU: A page cache algorithm for mobile devices with hybrid memory architecture," *IEEE Trans. Consumer Electronics,* vol. 62, no. 2, pp. 136-143, May 2016.

[13] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: Current status and future plans," in *Proc. Ottawa Linux Symposium*, Ottawa, Canada, Jul. 2007, vol. 2, pp. 21-34.

[14] Samsung 850 EVO SSD. [Online]. Available: http://www.samsung.com/semiconductor/minisite/ssd/product/consumer/850evo.html, 2015.

[15] SanDisk X300 SSD. [Online] Available: http://downloads.sandisk.com/downloads/datasheet/x300-datasheet.pdf.

[16] Flexible I/O Tester. [Online]. Available: https://github.com/axboe/fio.

[17] A. Sweeny, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the XFS file system," in *Proc. USENIX ATC*, San Diego, CA, Jan. 1996, pp. 1-14.

[18] D. Bovet and M. Cesati, *Understanding the Linux Kernel*, 3rd ed., O'Reilly Media, Sebastopol, CA, Nov. 2005.

[19] R. Love, *Linux Kernel Development*, 3rd ed., Addison-Wesley, 2010.

[20] EXT4 Documentation. [Online] Available: https://www.kernel.org/doc/Documentation/filesystems/ext4.txt.

[21] The OpenSSD project. [Online]. Available: http://www.openssd-project.org/wiki/The_OpenSSD_Project/.

[22] E. Lee, H. Bahn, M. Jeong, S. Kim, J. Yeom, S. Yoo, S. H. Noh, and K.G. Shin, "Reducing journaling harm on virtualized I/O systems," in *Proc. 9th ACM SYSTOR*, Haifa, Israel, Jun. 2016, pp. 15-20.

[23] Filebench. [Online]. Available: http://filebench.sourceforge.net/.

[24] A. Kopytov, Sysbench Manual. [Online] Available: http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf.

[25] D. Liu, L. Yao, L. Long, Z. Shao, and Y. Guan, "A workload-aware flash translation layer enhancing performance and lifespan of TLC/SLC dual-mode flash memory in embedded systems," *Microprocess. Microsyst.*, vol. 52, pp. 343-354, Jul. 2017.

[26] M.-C. Yang, Y.-H. Chang, C.-W. Tsao and C.-Y. Liu, "Utilization-aware self-tuning design for TLC flash storage devices," *IEEE Trans. VLSI Systems,* vol. 24, no. 10, pp. 3132-3144, Oct. 2016.

[27] K. Kwon, D. H. Kang, J. Park, and Y. I. Eom, "An advanced TRIM command for extending lifetime of TLC NAND flash-based storage," in *Proc. IEEE ICCE*, Las Vegas, NV, Jan. 2014, pp. 424-425.

[28] D. Jeong, Y. Lee, and J.-S. Kim, "Boosting quasi-asynchronous I/O for better responsiveness in mobile devices," in *Proc. USENIX FAST*, Santa Clara, CA, Feb. 2015, pp. 191-202.

[29] S. Kim, H. Km, J. Lee, and J. Jeong, "Enlightening the I/O path: A holistic approach for application performance," in *Proc. USENIX FAST*, Santa Clara, CA, Feb. 2017, pp. 345-358.

Kirock Kwon received the B.S. and M.S. degree in Department of Electronics Engineering from Kyoungbook National University, Korea, in 2000 and 2002, respectively. He is currently a Ph.D. candidate in the Department of Semiconductor Systems Engineering at Sungkyunkwan University. His research interests include file and storage systems, operating systems, and embedded systems.

Dong Hyun Kang received the B.S. degree in Department of Computer Engineering from Korea Polytechnic University, Korea in 2007 and the M.S. degree in College of Information and Communication Engineering from Sungkyunkwan University, Korea in 2010. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at Sungkyunkwan University. His research interests include file and storage systems, operating systems, and embedded systems.

Young Ik Eom received his B.S., M.S., and Ph.D. degrees in Computer Science and Statistics from Seoul National University, Korea in 1983, 1985, and 1991, respectively. He was a visiting scholar in the Department of Information and Computer Science at the University of California, Irvine, from Sep. 2000 to Aug. 2001. Since 1993, he has been a professor at Sungkyunkwan University in Korea. His research interests include virtualization, operating systems, storage systems, cloud systems, and UI/UX systems.