A Two-level SLC Cache Hierarchy for Hybrid SSDs

Li Cai[†], Zhibing Sha[†], Jun Li[‡], Jiaojiao Wu[†], Huanhuan Tian[†], Zhigang Cai[†], Jianwei Liao[†*]
[†]*College of Computer and Information Science, Southwest University, Chongqing, China*
[‡]*School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China*

Abstract—**Although high-density NAND flash memory, such as triple-level-cell (TLC) flash memory can offer high density, its lower write performance and endurance compared to single-level-cell (SLC) flash memory are impediments to the proliferation of TLC products. To overcome such disadvantages of TLC flash memory, hybrid architectures, which integrate a portion of SLC chips and employ them as a write cache, are widely adopted in commercial solid-state disks (SSDs). However, it is challenging to optimize the SLC cache, such as the granularity of cached data and the cold/hot data separation. In this paper, we propose supporting two-level hierarchy (i.e. L1 and L2) of SLC cache stores based on varying granularity of cached data. Moreover, we support the segmentation of the L1 and the L2 cache in the SLC region with a dynamic manner, by considering the write size characteristics of user applications. The evaluation results show that our proposal can improve I/O performance by between 12.6% and 25.1%, in contrast to existing cache management schemes for SLC-TLC hybrid storage.**

Index Terms—**Solid-state Drives (SSDs), SLC-TLC Hybrid SSDs, Two-level SLC Cache Hierarchy, I/O Performance.**

I. Introduction

High-density flash memory chips that store multiple bits per cell, such as each cell of Triple-level cell (TLC) flash store three bits, are being considered as SSD components due to their low price per unit [1], [2]. The read/write performance and endurance of TLC flash memory, however, are considerably limited. To address this issue, commercial solid-state disks (SSDs) commonly integrate single-level cell (SLC) chips and TLC chips, so that such hybrid SSDs can hold the vast majority of data in TLC chips and achieve comparable capacity of a pure TLC SSD, meanwhile offering I/O performance and endurance of SLC chips [3]. Specifically, in the SLC-TLC hybrid SSD, the SLC chips serve as a write cache to quickly complete write requests. It carries out garbage collection (GC) to reclaim the cache space when the SLC chips become full, indicating the buffered valid data pages will be migrated from the SLC chips to the TLC chips in hybrid SSDs [4].

In order to further improve the overall write performance of hybrid SSDs, many approaches have been proposed [3], [5], [6]. Specifically, Kwon et al. [5] presented dividing the SLC cache into the hot region and the cold region, and buffering the data pages in the relevant region according to their update hotness. As a result, it can reduce the number of valid page migrations during the GC operations in the SLC cache. Zhang el al. [3] suggested to employ the idle time slots on the parallel units of hybrid SSDs to proactively migrate the data from the SLC cache to the TLC chips, for mitigating the impacts of data migration. Shi et al. [6] proposed an I/O scheduling scheme for hybrid SSDs. It directly flushes the data of large write requests to the QLC region when the current I/O workload is intensive. Besides, it supports dynamically adjusting the size of the SLC cache to avoid frequent GCs.

*Corresponding author: J. Liao works for College of Computer and Information Science, Southwest University, P.R. China, and State Key Laboratory for Novel Software Technology, Nanjing University, P.R. China. E-mail: *liaotoad@gmail.com*.

The applications running at the host send I/O requests in many 4KB **sectors**, but the basic read/write unit of modern SSDs is a **page** of 16KB [7]. The partial page writes (updates), whose write size is smaller than a page will cause internal fragmentation within physical pages, and lead to a waste of the SLC cache space. Consequently, it will cause frequent GC operations in the SLC cache, and then impact the I/O responsiveness of hybrid SSDs [8]–[10]. We argue that how to efficiently address the issues of the granularity adaptivity of cached data and the cold/hot data separation in the SLC cache to better improve the performance of hybrid SSDs, have not yet been systematically explored.

This paper proposes a two-level SLC cache management scheme, called as *Hierarchy*, to address the aforementioned issues in hybrid SSDs. It borrows the basic principle of tiered storage, and adopts a hierarchy of SLC cache stores based on varying data granularity and access frequency to buffer user data. In summary, this paper makes the following three contributions:

- We present a two-level SLC cache hierarchy approach, called *Hierarchy*. It splits the SLC cache into two layers of the L1 cache and the L2 cache. The L1 cache serves partial page writes (updates) after their data are merged into full page data to alleviate internal fragmentation, while the L2 cache satisfies the full page write requests for ensuring read responsiveness.
- We propose an adaptive SLC cache division scheme to enhance the SLC cache use efficiency. Our approach dynamically adjusts the SLC space for L1 and L2 cache based on the write workloads to both portions of the cache. As a result, our *Hierarchy* proposal can reduce the negative effects of GC operations and improve I/O responsiveness.
- We perform a series of evaluation tests by using disk traces of real-world applications. Our measurements show that our proposal can respectively reduce the average write latency and read latency by 18.7% and 22.1% on average, in contrast to state-of-the-art cache management

techniques for hybrid SSDs.

The rest of this paper is structured as follows: Section II introduces the background knowledge and motivations. Section III describes the design and implementation specifications of *Hierarchy*. The evaluation results and relevant discussions are presented in Section IV. Finally, the paper is concluded in Section V.

II. Background and Motivation

A. SSDs and Hybrid SSDs

NAND flash-based SSDs have been widely used thanks to their features of energy saving and volumetric capacity [11], [12]. To further decrease the per-unit price of SSDs, flash density increases to TLC, indicating a flash cell can store three bits of data [13], [14]. However, the decreases in endurance and I/O performance accompanying the shrinking of manufacturing size are now becoming a major issue to deal with in the domain of high density SSD products [15]–[17].

To address this issue, the hybrid SSD architecture has been proposed, and we employ the SLC-TLC hybrid SSD as the illustration in this paper. The hybrid SSD consists of SLC chips and TLC chips, in which SLC acts as the write cache and serves hot data while TLC chips serve cold data [18], [19]. In general, the new data are first written onto the SLC cache, for quickly responding to the write requests. Then, the buffered data will be later flushed onto the TLC chips if the SLC cache becomes full. In other words, the hot write data will be kept in the SLC cache to boost I/O performance, the TLC chips can offer larger storage capacity [3].

The hybrid architecture, however, typically is equipped with a small SLC cache and manages the SLC cache at the page-level granularity, which leads to low space utilization as certain update requests are much smaller than the size of a page. Then, the SLC cache will be filled up quickly and many inefficient GCs will be triggered. These GC operations, however, may indiscriminately transfer the hot data pages to the TLC flash, leading to the TLC chips being involved in subsequent update requests, which further degrades overall performance.

B. Optimization on Hybrid SSDs

A number of optimization strategies on hybrid SSDs have been proposed to either minimize the GC overhead in the SLC cache or enhance the overall I/O performance.

To be specific, Kwon et al. [5] proposed to split the SLC cache into two parts, including the hot region and the cold region, and then dispatch the hot data to the hot region while other data will be flushed to the cold region. This approach can minimize the number of page migrations in GC processes, but it cannot perform well on utilizing the storage space of the SLC region in the scenarios of many partial page writes, as the buffer unit is the SSD page.

Zhang et al. [3] proposed SPA-SSD for minimizing the negative effects caused by moving the buffered data from the SLC cache to the TLC chips in the hybrid SSD. To this end, SPA-SSD proactively triggers data migration on the parallel units while they are not busy processing I/O requests.

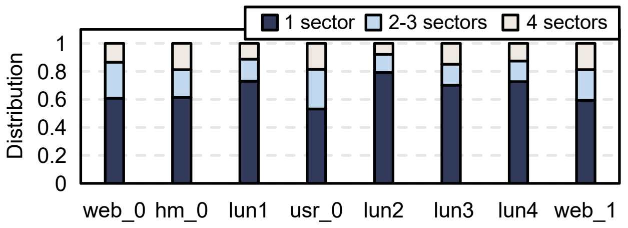


Fig. 1: SLC cache update size distribution after replaying the selected traces in the SLC-TLC hybrid SSD. Note that the cache granularity is the page of 16KB, and the sector is 4KB.

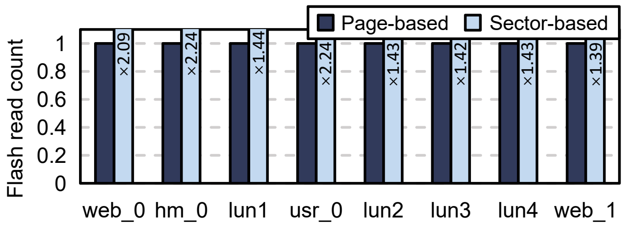


Fig. 2: Normalized numbers of read accesses on the SLC cache for servicing a page-size read request with both page-based and sector-based SLC cache management schemes.

Consequently, it can mitigate the interference between the background data migration and user I/O requests, thus improving I/O performance. Shi et al. [6] proposed scheduling small size and frequently updated data to the SLC cache while directly placing the data of large write requests to the quad-level cell (QLC) SSD if the current workloads are intensive in the SLC-QLC hybrid SSD. As a result, it can relieve the flush intensity on the SLC cache and reduce the negative effects caused by migrating data from the SLC cache to QLC SSDs.

Certain modern SSDs support the flash cells working in both original high-density modes (e.g. QLC and TLC) and low-density modes (e.g. SLC), so that the flash cells with the low-density mode will be used as the cache. For better enhancing I/O performance, Wei et al. [20] employed reinforcement learning to guide not only the conversion between the original high-density mode and low-density modes, but also the data migration from low-density mode cells to high-density mode cells. In addition, they proposed using k-means clustering to identify cold/hot data, and scheduling the hot data to the flash cells with the SLC mode while the cold data to the flash cells with the QLC mode. Consequently, it can ensure real-time responses to the update requests, thereby improving the overall performance of hybrid SSDs.

C. Motivations

In order to quantify the proportion of partial update requests when running real-world applications, we carried out a series of experiments and the experimental settings will be presented in Section IV-A. Figure 1 shows the distribution of varied sizes of update requests when running the selected benchmarks in a conventional SLC-TLC hybrid SSD. As seen, 53.1%-79.2% of update requests completed in the SLC cache are smaller than a sector of 4KB. To satisfy such an update request, it must read the entire original data page on the SLC cache or the TLC chips, and merge the contents with the updated part, and then write the whole page of data to the SLC cache. Though only the update part should be buffered in the SLC cache, it must buffer a whole page of data with page-based cache management, that must cause a waste of the cache space.

An intuitive way is to manage the SLC cache at the sector granularity if partial page updates account for a major proportion of all update requests. However, sector-based cache management generally results in space overhead caused by holding the sector mapping table. Moreover, it will lead to a reduction in read performance, which is caused by reading on multiple flash pages to form the data for responding to a page-size read request. For example, reading 16KB of data, in the worst case, takes up to four times longer if the data is spread across four different flash pages. Figure 2 presents the results of normalized numbers of read accesses on the SLC cache for servicing a page-size read request with both page-based and sector-based SLC cache management schemes. As read, the sector-based scheme entails more flash reads to serve read requests, by between `39.0%` and `124.0%` when compared to the page-based scheme. This fact verifies that managing the SLC cache at the sector granularity will worsen the read performance of hybrid SSDs.

Such observations motivate us to propose a novel SLC cache hierarchy for hybrid SSDs, aiming to improve read responsiveness and minimize the overhead of data migrations between the SLC cache and TLC chips when running applications.

III. Design and Implementation of *Hierarchy*

A. System Architecture

Figure 3 illustrates the high-level overview of *Hierarchy*. The SLC cache slots are distributed across SSD channels, and the slots are labeled as the L1 cache or the L2 cache. Then, it schedules the write request to be absorbed by which part of the SLC cache depending on the write size. To be specific, the full write requests are directly dispatched to the L2 cache, whereas the partial page writes will be merged with others to generate a full data page and then the data page will be flushed to the L1 cache. In addition to merging the partial page writes, we also merge the partial page updates to form a full page write in the L1 cache, even though the original page of data is buffered in the L2 cache.

On the one hand, the data of full write requests will be directly buffered in the L2 cache for ensuring read responsiveness. The cache space will be reclaimed with GC operations, and the data pages in the GC block will be treated as the cold data and flushed to the TLC chips. On the other hand, the data of either partial write request or partial update request will be buffered in the L1 cache after a merging operation, which helps improve cache space utilization and keep hot update data in the L1 cache. In the GC process of the L1 cache, it needs to check the L2 cache and the TLC chips to make sure whether other parts of the same page data exist or not. If yes, *Hierarchy* merges such sectors of data with the original page of data and flushes the latest version to the TLC chips.

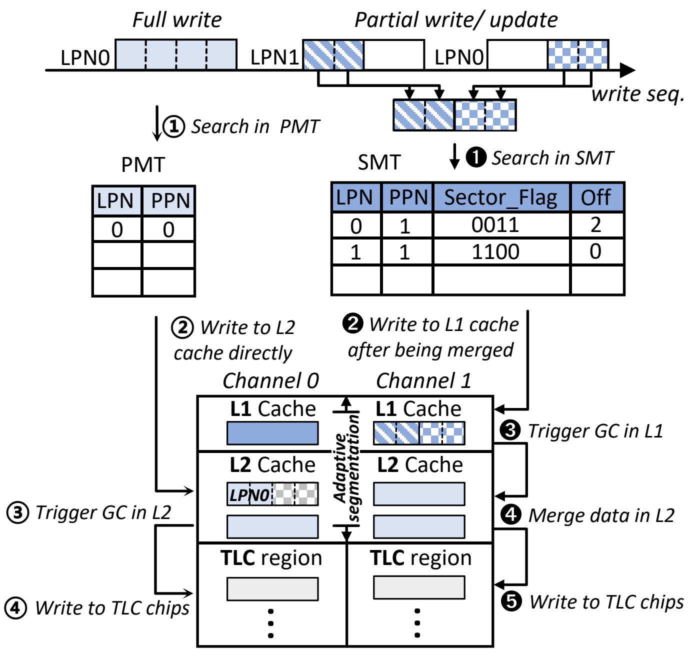


Fig. 3: High-level architectural overview of *Hierarchy*, and the segmentation of L1 and L2 in the SLC cache are adjustable. We assume the hybrid SSD has 2 channels for simplicity.

B. Mapping Management

We suggest that *Hierarchy* keeps hot sectors of data in the L1 cache for improving the cache use efficiency, and maintains the cold sectors of data in the L2 cache or even move them to the TLC region with GC operations. To fulfill address translation in the case of managing data at different granularity, we introduce two level mapping tables, including the sector-level mapping table (SMT) that manages the sector-based data in the L1 cache, and the page-level mapping table (PMT) that deals with the page-based data in the L2 cache and TLC chips. The fields of *Sector_Flag* and *Off* in the SMT entry represent which sectors in the logical page are buffered in the L1 cache and the offset inside the flash page for buffering the data.

The two-level cache design may impact read performance, because the sectors belonging to the same logical page spread across the flash pages in both levels of the cache. To mitigate the performance degradation caused partial page updates, we propose distributing the updated data across different parallelism units. For example, when the partial page update occurs (e.g., last two sectors of data in *LPN0* are updated), our proposal will merge them with the data of other partial page writes (e.g., LPN1) or other partial page updates to generate a new page of data. Then, the page of data will be flushed to the L1 cache slots located at *Channel 1* for avoiding read conflicts, with two new relevant mapping entries in the SMT table.

C. SLC Cache Division

Because user applications may have varying I/O access patterns, the fixed division of the L1 cache and the L2 cache cannot work well in such scenarios. To address this issue,

Algorithm 1 The process of servicing a write request

```
1: Input: The $size$, $LPN$ of write, $cur\_ratio$ of $L2$, $ratio_{L2}$, the number of processed requests of $req\_num$, the numbers of full page writes and merge writes of $\omega_L$ and $\omega_S$;
2: Output: Null;
3: $req\_num++$;
4: /*decide using the cache slots in L1 or L2 */
5: if $ratio_{L2} > cur\_ratio$ then
6:     $src = allocate\_L1\_page()$;
7:     $cur\_ratio += 1/CACHE\_SIZE$;
8: else
9:     $src = allocate\_L2\_page()$;
10:    $cur\_ratio -= 1/CACHE\_SIZE$;
11: end if
12: if $size < PAGE\_SIZE$ then
13:    /*merge the partial page write*/
14:    $merge\_to\_full\_page()$;
15:    /*write to non-conflict channel of the L1 cache*/
16:    $ch = find\_none\_conflict\_channel()$;
17:    $write\_to\_L1(ch, src, LPN)$;
18:    $\omega_S++$;
19: else
20:    /*schedule the full page write to L2*/
21:    $write\_to\_L2(src, LPN)$;
22:    $\omega_L++$;
23: end if
24: /*periodical division adjustment*/
25: if $req\_num == window\_size$ then
26:    /*estimation the best division ratio for L2*/
27:    $ratio_{L2} = \omega_L/(\omega_S+\omega_L)$;
28:    /*reset for the new round of adjustment*/
29:    $req\_num = 0$;
30:    $\omega_L = \omega_S = 0$;
31: end if
```

we propose a division method to direct dynamically adjusting the segmentation of the SLC cache. Equation 1 defines the objective function for cache space division.

$$r = \frac{\omega_L}{\omega_L + \omega_S} \tag{1}$$

where r means the proportion of space allocated to the L2 cache, ω_L and ω_S are the numbers of full page writes and the merged writes.

As illustrated, our method considers the number of full page writes and the merge writes occurred in the previous time window, to periodically adjust the segmentation of the L1 cache and the L2 cache, for servicing I/O requests in the following time window.

D. Implementation Details

Algorithm 1 demonstrates the implementation details of the proposed scheme. First, *Hierarchy* decides either L1 or L2 will offer the cache slot for the data, after comparing the optimal ratio with the current ratio of L2 (Lines 5-11). Then, it checks the size of write request to verify whether it is a partial page write or not. If it is a partial page write, *Hierarchy* first merges it with others to form a full page data. Then, it retrieves the L1 cache slots located at the specific channel to avoid read conflict between L1 and L2 (Lines 12-18). We argue that the original whole page might already be in the L2 cache while the updated sectors are buffered in the L1 cache, reading the page of data requires accessing on both L1 and L2. Otherwise, *Hierarchy* flushes the data of the write request to the L2 cache (Lines 19-23). After the write request is satisfied and the accumulated number of write requests approaches the size of the pre-defined time window, *Hierarchy* will update the optimal segmentation of L1 and L2 based on the numbers of partial page writes and full page writes occurred in the last time window for future cache management (Lines 25-31).

TABLE I: Experimental settings of *MQsim*

SSD parameters	
SLC flash layout	Plane-level distribution
(Channel, Chip, Die, Plane)	(4, 1, 1, 4)
SLC/TLC Block num per Plane	64/448
Page num per SLC/TLC block	128/384
Page size	16KB
Sector size	4KB
Total Capacity	44GB
GC trigger threshold	50%
SLC/TLC Program latency	500us/1600us
Erase latency	15ms

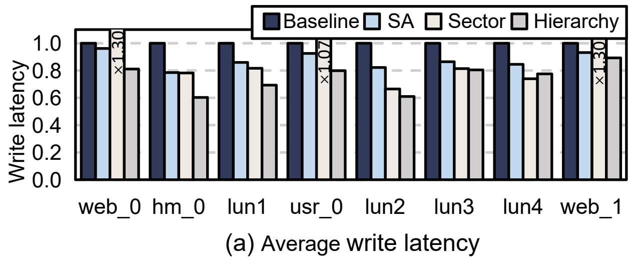
TABLE II: Specifications on traces (ordered by write ratio)

Traces	**Req. #**	**Write R**	**Write SZ**	**Read SZ**
web_0	2,029,945	70.1%	8.59KB	29.9KB
hm_0	3,993,316	64.5%	8.33KB	7.36KB
lun1	749,806	61.5%	8.84KB	25.2KB
usr_0	2,237,889	59.5%	10.20KB	40.9KB
lun2	441,199	59.0%	8.02KB	23.6KB
lun3	821,226	52.9%	9.10KB	24.9KB
lun4	759,483	50.1%	8.77KB	25.6KB
web_1	160,891	45.8%	9.21KB	45.9KB

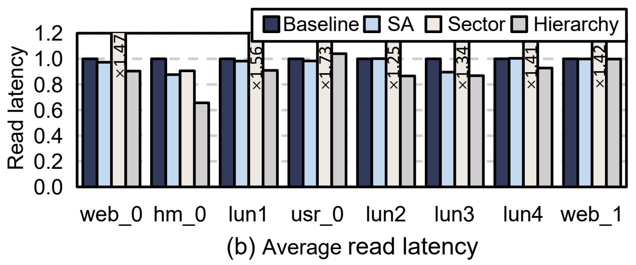
IV. Experiments and Discussions

A. Experimental Settings

We evaluated the effectiveness of our proposed method by using the trace-driven SSD simulator of *MQSim* [21]. Table I lists the experiment settings in our evaluation tests, and the GC trigger threshold is set to 50.0% in both SLC chips and TLC chips by referring to [6]. We selected eight commonly used block I/O traces of real-world applications in the tests. Among them, four traces of *hm_0*, *usr_0*, *web_0* and *web_1* are from the block I/O trace collection of Microsoft Research Cambridge [22]. Another four I/O traces are collected from the application of an enterprise virtual desktop infrastructure (VDI) [23], including *additional-2016021616-LUN1*, *additional-2016021617-LUN1*, *additional-03-2016021615-LUN3*, and *additional-03-*



(a) Average write latency



(b) Average read latency

Fig. 4: Write and read performance comparison after replaying the selected block I/O traces.

2016021616-LUN3, labelled as *lun1* to *lun4*. Table II presents the details of the selected traces.

Apart from our proposal of *Hierarchy*, another three comparison counterparts are used in our evaluation:

- ***Baseline***, which is the design adopted by most hybrid SSDs, and uses the SLC cache as the write cache to absorb all write requests at the granularity of page.
- ***SA*** [8], which merges partial write requests to generate full page writes, for reducing the number of flash writes and improving cache space utilization. Note that *SA* adopts page-level mapping, and each PPN may be mapped with multiple LPNs.
- ***Sector*** [10], which is a sector-level mapping scheme. It merges partial write requests with similar hotness to full page write requests, and enables sectors in a logical page spread across most four flash pages. We argue that *Sector* causes memory overhead for holding the fine-granularity mapping table.

Note that we employ these schemes to manage the SLC cache, and one-shot programming is used in the TLC chips [3]. The proportion of the L1 cache is limited not larger than 0.5 to ensure the L2 cache has more capacity. Besides, the number of I/O requests in each time window is set to 1024 write requests by referring to [24].

B. Performance Results

To validate the effectiveness of our proposed *Hierarchy*, we used three primary performance measures in our experiments: (a) *write and read latency*, (b) *data migration analysis*, and (c) *SLC cache division statistics*.

1) Write and Read Latency: I/O response time is the critical performance indicator to reflect SSD performance, and Figure 4 reports the normalized results of I/O response time after running the selected block traces. We can observe that the proposed *Hierarchy* scheme outperforms other comparison counterparts, in terms of average read latency and write latency. More exactly, *Hierarchy* reduces the average write latency by 25.1%, 12.6%, and 18.9%, in contrast to *Baseline*, *SA*, and *Sector* respectively. Meanwhile, it shows that *Hierarchy* can reduce the average read latency by 10.4%, 6.8%, and 49.3%, when compared to the selected comparison counterparts.

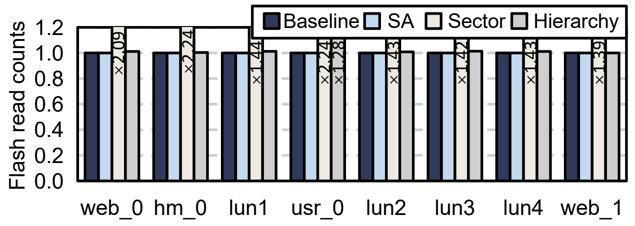


Fig. 5: Normalized numbers of reads on the SLC cache.

Different from *Baseline* and *SA*, our proposed *Hierarchy* approach merges the data of hot update sectors to full pages, so that it can save the SLC cache space for enabling more hot data to be buffered in the SLC cache. Compared with *Sector*, our proposal of *Hierarchy* allow the hot/cold update data of a logical page to be distributed across L1 and L2 caches in different parallelism units, facilitating faster accesses when handling large read requests, thus enhancing read performance.

Another noticeable information is about *Sector* introduces more read latency by 42.3%, in contrast to other schemes. This is because *Sector* manages the cached data at the granularity of sector in the SLC cache, and page-size read requests may be serviced by accessing multiple flash pages. Figure 5 presents normalized numbers of reads on the SLC cache for serving the same number of user read requests, and the results verify the *Sector* scheme causes the largest number of read accesses on the SLC cache.

2) Data Migrations: GC operations occurred in the SLC cache migrate the valid pages to the TLC chips. Each GC operation consists of multiple data migrations, and each of them corresponds to a pair of read and write on the flash memory, which can obstruct enqueued user I/O requests, and cause I/O performance degradation in hybrid SSDs. We record the number of reads (i.e. GC reads) and the total write volume of data migrations caused by GC operations after running the selected benchmarks, and the results are presented in Figure 6.

As read, *Hierarchy* cuts down the number of GC reads by 16.8%, 25.9% on average, in contrast to *Baseline* and *Sector*, with no difference compared to *SA*. This is because *Hierarchy* keeps the data of hot sectors in the L1 cache with subpage merging, indicating high cache space utilization that can minimize the GC overhead. Besides, it shows that *Sector* yields the worst in terms of *GC reads*. This is because *Sector* manages the entire SLC chips at the sector level, while the TLC chips are managed at the page level. This difference can result in up to four GC reads for a valid page migration.

The measure of write volume of data migrations is another indicator of SLC cache management schemes, as the SSD endurance can be measured by the total write volume. As shown in Figure 6(b), our *Hierarchy* approach yields the smallest

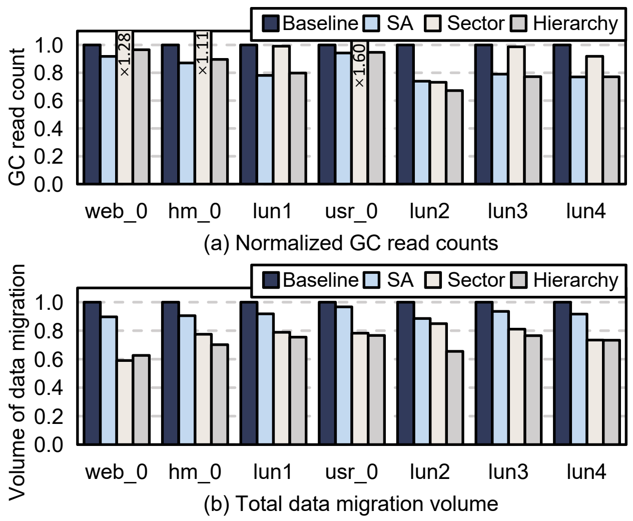


Fig. 6: The number of GC read and total data volume of data migrations when running the benchmarks. Note that the trace of *web_1* is a read-dominant workload so that data migration is not triggered.

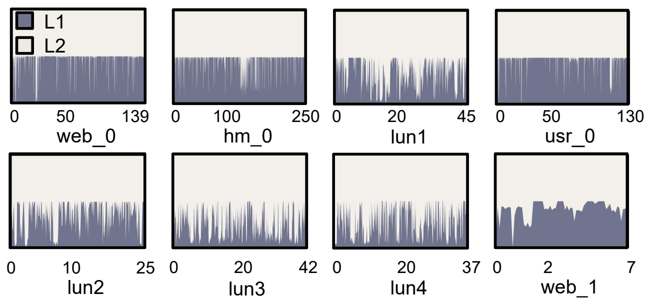


Fig. 7: The proportion of SLC cache occupied by the L1 cache and the L2 cache. In which, the X-axis represents the number of processed write requests (in unit of 1024×10^1).

number of data migration volume after running all selected benchmarks, corresponding to a reduction in write volume by 28.5%, 20.3%, and 4.7% in contrast to *Baseline*, *SA*, and *Sector*. This is because *Hierarchy* dispatches the data of hot update sectors to the L1 cache, while the remaining data of cold sectors are kept in the L2 cache or in the TLC region. Thus, it achieves hot/cold separation and reduces the volume of data migrations in GCs of the SLC cache.

3) SLC Cache Division Statistics: Figure 7 illustrates the results of cache division status when running selected benchmarks at different time windows. As seen, our *Hierarchy* method dynamically adjusts the SLC cache division for both L1 and L2 cache, according to the I/O features of the currently running benchmarks. As a result, the overall I/O latency can be noticeably reduced, which had been previously discussed in Section IV-B1.

C. Space and Time Overhead

There is no additional space overhead caused by *Baseline* and *SA*, because they employ conventional page-level mapping tables. The main memory overhead of our proposals is caused by keeping the sector-level mapping table (SMT) for managing the L1 cache. By referring back to the Figure 3, in addition to the LPN and PPN, it uses a four-bit *sector_flag* to represent which sectors are stored in the L1 cache, and two-bit offset to indicate the start position in the flash page. To avoid huge space overhead, we limit the max size of the L1 cache to half of the SLC cache. Thus, SMT contains 262,144 items, and each item takes up 5.5 bytes, and a total of $262,144 \times 5.5 \div 1024^2$ = 1.375MB of memory space is used to store SMT, while the *Sector* requires **twice** as much memory to manage the entire SLC cache at the sector level. Besides, SMT adopts the hash function to lookup item, so that we argue that the time overhead of *Hierarchy* is acceptable, even running on the computing-limited platform.

V. Conclusion

This paper has proposed a two-level SLC cache hierarchy scheme, that divides the SLC cache into two layers of the L1 cache and the L2 cache. The L1 cache serves partial page writes after their data are merged into full page data to enhance cache space utilization, while the L2 cache satisfies the full page writes to ensure read responsiveness. Moreover, we present an experiential method to periodically determine the preferred segmentation of the L1 cache and the L2 cache inside the SLC cache. Our experimental results demonstrate that our proposal can noticeably improve the read performance by 22.1% on average, and reduce the volume of data migrations during GCs by up to 28.5%, in contrast to state-of-the-art cache management schemes for hybrid SSDs.

Acknowledgment

This work was partially supported by "The Chongqing Graduate Research and Innovation Project (No. CYS240155)", and "the Opening Project of State Key Laboratory for and Novel Software Technology (No. KFKT2024B47)".

References

[1] Yu D, et al. Differential Evolution Algorithm with Asymmetric Coding for Solving the Reliability Problem of 3D-TLC CT Flash-Memory Storage Systems. In ***TCAD***, 2021.
[2] Kang M, et al. PR-SSD: Maximizing partial read potential by exploiting compression and channel-level parallelism. In ***TC***, 2022.
[3] Zhang W, et al. SPA-SSD: Exploit heterogeneity and parallelism of 3D SLC-TLC hybrid SSD to improve write performance. In ***ICCD***, 2019.
[4] Luo L, et al. Revisiting TRIM On High-Density Flash-Based Hybrid Storage Systems. In ***TCAD***, 2023.
[5] Kwon K, et al. An advanced SLC-buffering for TLC NAND flash-based storage. In ***TCE***, 2017.
[6] Shi L, et al. Understanding and optimizing hybrid ssd with high-density and low-cost flash memory. In ***ICCD***, 2021.
[7] Fareed I, et al. Update frequency-directed subpage management for mitigating garbage collection and dram overheads. In ***TCAD***, 2023.
[8] Kang M, et al. Subpage-aware solid state drive for improving lifetime and performance. In ***TC***, 2018.
[9] Fareed I, et al. PAPA: Partial page-aware page allocation in TLC flash SSD for performance enhancement. In ***MSST***, 2020.
[10] Fareed I, et al. Leveraging intra-page update diversity for mitigating write amplification in SSDs. In ***ICS***, 2020.
[11] Kwak J, et al.Cosmos+ OpenSSD: Rapid prototype for flash storage systems. In ***TOS***, 2020.

[12] Yang F, et al. Out-of-channel data placement for balancing wear-out and I/O workloads in RAID-enabled SSDs. In ***DATE***, 2023.

[13] Wu W L, et al. CDS: Coupled Data Storage to Enhance Read Performance of 3D TLC NAND Flash Memory. In ***TC***, 2023.

[14] Wu J, et al. Polling sanitization to balance I/O latency and data security of high-density SSDs. In ***TOS***, 2024.

[15] Park J, et al. Reducing solid-state drive read latency by optimizing read-retry. In ***ASPLOS***, 2021.

[16] Huang D, et al. A Read Latency Variation Aware Independent Read Scheme for QLC SSDs. In ***DATE***, 2024.

[17] Yao X, et al. Extremely-Compressed SSDs with I/O Behavior Prediction. In ***TOS***, 2024.

[18] Luo L, et al. Critical Data Backup with Hybrid Flash-Based Consumer Devices. In ***TACO***, 2023.

[19] Li Q, et al. Midas Touch: Invalid-Data Assisted Reliability and Performance Boost for 3d High-Density Flash. In ***HPCA***, 2024.

[20] Wei Q, et al. Reinforcement learning-assisted management for convertible SSDs. In ***DAC***, 2023.

[21] Tavakkol A, et al. MQSim: A framework for enabling realistic studies of modern Multi-QueueSSD devices. In ***FAST***, 2018.

[22] Narayanan D, et al. Write off-loading: Practical power management for enterprise storage. In ***TOS***, 2008.

[23] Lee C, et al. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In ***SYSTOR***, 2017.

[24] Li J, et al. Pattern-based write scheduling and read balance-oriented wear-leveling for solid state drivers. In ***MSST***, 2019.