# Coldmap: Extending SSD Lifetime Exploiting Multi-Page Mapping Information

Jaewon Seo
*Department of Computer Science and Engineering*
*Korea University*
Seoul, South Korea
yarks@korea.ac.kr

Gunjae Koo
*Department of Computer Science and Engineering*
*Korea University*
Seoul, South Korea
gunjaekoo@korea.ac.kr

2024 13th Non-Volatile Memory Systems and Applications Symposium (NVMSA) | 979-8-3503-8855-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/NVMSA63038.2024.10693670

***Abstract*—Solid-state drives (SSDs) include flash translation layer (FTL) functions to manage the inherent characteristics of NAND flash memory. One critical aspect of the flash management functions is garbage collection which reclaims NAND flash blocks that include many invalid pages provoked by page data updates. The garbage collection function can degrade the performance and lifespan of an SSD since the garbage collection creates redundant page reads/writes and block erases in an SSD. In order to tackle the performance and cost issues of garbage collection, we propose Coldmap, an efficient garbage collection mechanism based on the multi-page mapping information for duplicated flash pages. We reveal that the duplicated pages mapped by multiple logical pages exhibit a longer lifespan compared to the non-duplicated pages. Coldmap groups the duplicated pages into new flash blocks while garbage collection is performed. Since the duplicated pages grouped in the *cold zone* blocks exhibit a longer lifetime, the corresponding blocks are erased less frequently, thus Coldmap can minimize the block erase counts. Our evaluation results using FEMU exhibit that Coldmap can effectively improve the performance and lifetime of an SSD by 15.4% and 84.3%, respectively, compared to the existing remapping-based SSD firmware.**

***Index Terms*—SSD, Garbage Collection, FTL**

## I. Introduction

Solid-state drives (SSDs) have become mainstream storage devices that can provide high-performance data input/output (I/O) for accommodating a large amount of data. SSDs employ NAND flash memory as the main data storage media, thus an SSD includes firmware functions that can manage the native characteristics of NAND flash memory. For instance, the SSD firmware includes a page translation mechanism that associates logical pages with physical flash pages since page writes in NAND flash memory can be allowed only for erased (or initialized) pages. The SSD firmware also provides a garbage collection function that erases flash memory blocks to provide available empty pages for future page writes. Since NAND flash memory only allows block-level erasure, the garbage collection provokes redundant page reads/writes in an SSD. Namely, the blocks to be erased by the garbage collection function include valid pages apparently, thus such valid pages have to be copied (i.e. read and written) to other empty pages. Hence, the performance of an SSD can be significantly degraded if garbage collection is performed frequently. Moreover, the lifetime of an SSD can be significantly influenced by the garbage collection mechanism since NAND flash memory allows a limited erasure count per block.

Contemporary storage data often contains a significant proportion of duplicated data segments. Therefore, effective management of duplicate data is crucial for optimizing performance and cost in computer systems. Several studies present storage firmware-based approaches to handle duplicated data pages within an SSD. These approaches enable many-to-one mapping in flash translation, associating multiple logical data sectors with a single physical flash page for duplicate data. If data duplication is detected during writes to an SSD, the proposed flash translation firmware associates the logical pages of the duplicated data with existing physical pages. Since writes to NAND flash memory are not performed for the duplicated sectors, the flash firmware with the proposed remapping mechanisms can enhance the performance of an SSD by minimizing effective flash memory transactions.

In this paper, we propose an efficient garbage collection mechanism called *Coldmap*, based on the remapping mechanism for duplicated data in an SSD. We analyze the lifetime of flash pages to reveal that the flash pages associated with multiple logical sectors stay longer without updates. Based on the analysis results, Coldmap groups the physical flash pages associated with multiple logical pages in a *cold zone* block while garbage collection is performed. Since the pages in the cold zone exhibit a longer lifetime before invalidation, the garbage collection function selects the blocks in the cold zone less frequently for block erasure. Therefore, Coldmap can effectively reduce garbage collection and block erase counts. We implement the Coldmap mechanism using FEMU which employs the remapping approach for duplicated flash pages. Our evaluation results with various SSD workloads reveal that Coldmap can effectively reduce garbage collection counts thus Coldmap can improve the performance by 15.4% and the lifetime of an SSD by 84.3%, compared to the existing remap-based SSD.

The remainder of this paper is organized as follows. We introduce garbage collection and the remapping mechanisms for duplicated data in Section II. We exhibit the lifetime analysis of flash pages in Section III. We present the proposed Coldmap mechanism in Section IV. We exhibit the evaluation results in Section V. We conclude the paper in Section VI.

## II. Background

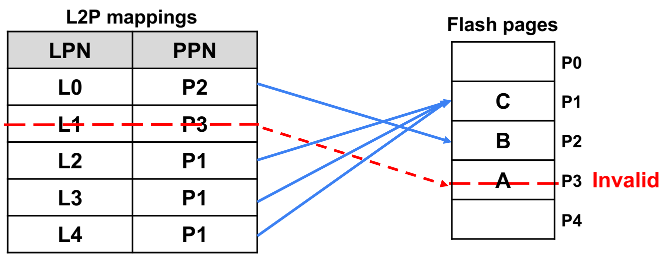
### A. Garbage Collection in SSD

NAND flash memory employed as the main storage media of SSDs does not support in-place writes, thus flash memory cells have to be erased before data writes or updates. In order to achieve high-performance flash page writes, the flash translation layer (FTL) firmware in an SSD writes modified page data to new empty pages (i.e. erased pages) instead of performing cumbersome erase-and-write processes. After the modified pages are written to empty pages, the firmware just marks the existing old pages as *invalid* pages. The invalid pages need to be erased later to maintain the available storage space in an SSD. Moreover, NAND flash memory allows block-level erases only to avoid data corruption by high erasure voltage levels. Hence, the FTL firmware performs *garbage collection* to prepare NAND flash blocks available for instant writes. For this process, FTL monitors the number of invalid pages per block. Then, FTL performs block-level erasure for the flash blocks where the number of invalid pages exceeds the threshold. Note that the garbage collection causes many additional writes in an SSD since the flash blocks to be erased can include valid pages also. Furthermore, the block erasure of flash memory is extremely slow, thus frequent garbage collection can degrade the performance of an SSD.

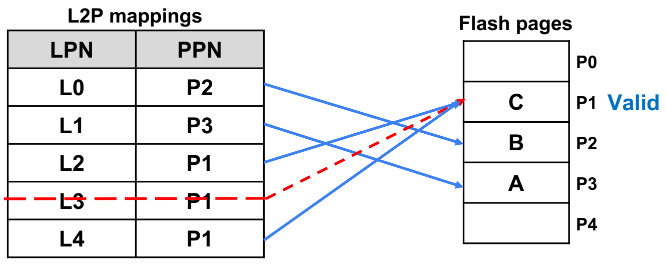
The garbage collection process significantly influences the lifetime of an SSD. NAND flash blocks support limited erasure counts since the erase process can wear out flash memory cells with high-voltage drives. For instance, modern quad-level cell (QLC) flash that is deployed in high-density SSDs supports up to approximately 1000 program/erase (P/E) cycles per block [1], [2]. The FTL firmware prohibits any accesses to worn-out flash blocks if the reliability levels of the blocks become lower than the permitted threshold. Hence, the available space of an SSD can shrink as more blocks are marked as *unavailable*. Since garbage collection increases the erase counts of the target flash blocks, the lifetime of an SSD can decrease due to frequent garbage collection processes.

### B. Remapping Duplicated Pages in SSD

In modern storage data, we can frequently observe duplicated data sectors especially for multi-tenant systems [3]. Namely, these duplicated data sectors in a file system occupy multiple pages in an SSD even though the data in the flash pages are identical. Furthermore, the SSD firmware needs to handle multiple I/O requests for the duplicated data. Hence, the storage space and hardware resources in an SSD can be significantly wasted by such duplicated data sectors. In order to tackle such inefficiency for duplicated data sectors, several researchers proposed approaches that can handle duplicated flash pages in an SSD [4]–[14]. Such approaches, called remap-based SSDs, exploit the flash page translation mechanism in the FTL layer of an SSD to handle duplicated sectors. Note that in a traditional FTL one logical page number (LPN) is mapped to one physical page number (PPN) only. On the other hand, remap-based SSDs allow a single physical page to be associated with multiple logical pages that contain identical data. Namely, the FTL firmware of the remap-based SSDs allows many-to-one mapping in the flash transition table if the multiple logical sectors contain the same data. In order to identify the duplicated data in multiple logical sectors, the remap-based SSDs exploit the hash values of the data in logical sectors. Namely, the remap-based SSDs link multiple logical pages that exhibit the same hash values to a single physical page. By doing so, the remap-based SSDs can minimize the flash write transactions since flash writes are not performed for the duplicated pages.



(a) Updates for a single-mapping page



(b) Updates for a multi-mapping page

Fig. 1: Flash translation updates by remapping mechanisms

## III. Motivation

As described in the previous section, the remap-based SSDs can improve the performance of an SSD by minimizing flash writes for duplicated data. Moreover, the remap-based mechanisms can reduce the number of physical pages occupied by duplicated data. In this section, we investigate the lifetime of single-mapping pages (i.e. non-duplicated data pages) and multi-mapping pages (i.e. duplicated data pages) in the remap-based SSDs.

Figure 1 illustrates how the remap-based SSD firmware manages flash translation for singe-mapping and multi-mapping pages. Note that the remap-based SSD associates a single logical page with a single physical page for non-duplicated data as shown in Figure 1a. If the data in the single-mapping page (L1 in the figure) is updated, the physical page (P2) is *invalidated* and a new empty physical page is associated. Note that invalidated pages need to be retrieved by the garbage collection function. On the other hand, the remap-based SSD associates multiple logical pages (L2, L3, and L4) with a single physical page (P1) for duplicated pages as shown in Figure 1b. In this case, the physical page P1 remains in a

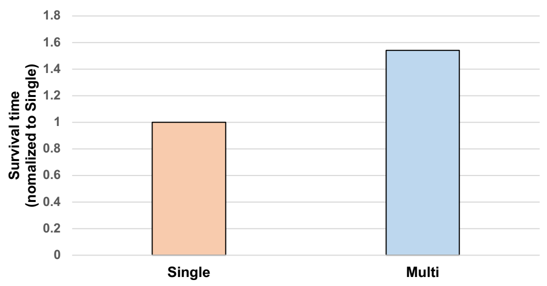


Fig. 2: Lifespan of single-mapping and multi-mapping pages

*valid* state even though one of the logical pages is updated. P1 will be finally invalidated after all three associated logical pages are updated. Hence, it is probable that multi-mapping pages remain valid for a longer duration compared to single-mapping pages.

In order to analyze the lifetime of single-mapping and multi-mapping pages in the remap-based SSDs, we measure the survival time (i.e. the time interval from a page write to a page invalidation) of a page. For this experiment, we use FEMU, an NVMe SSD emulator based on QEMU/KVM [15]. We configure 16 GB of a remap-based SSD model using FEMU. We use FIO as a workload for this analysis [16]. FIO is configured for random writes with a data duplication rate of 30%. Note that the data duplication rate we configured for this study is similar to the fraction of the duplicated data collected from smartphone file systems [3].

In Figure 2 we compare the lifespan of single-mapping and multi-mapping pages in the remap-based SSD. The y-axis of the graph represents the average survival time of physical pages. Note that the survival time refers to the time interval between a page write and its subsequent invalidation. The survival time is normalized to the survival time of single-mapping pages. Our analysis reveals that multi-mapping pages exhibit 53% longer survival time compared to single-mapping pages. Consequently, our experiment discloses we can categorize short-living and long-living pages in remap-based SSDs based on the different page mapping classes (i.e. single-mapping and multi-mapping).

## IV. Coldmap

### A. Proposed Garbage Collection

We can develop more efficient garbage collection mechanisms if we accurately estimate the lifetime of physical pages in NAND flash memory. Note that the efficiency of garbage collection deteriorates if short-living and long-living pages are mixed within a NAND flash block. Namely, long-living pages probably remain valid but many short-living pages are invalidated in a flash block after data updates. If garbage collection is executed for this block, valid pages, which are likely long-living pages, need to be copied to other empty pages. In this case, the garbage collection provokes additional flash reads and writes. Moreover, the block space is wasted by many invalidated *short-living* pages, thus NAND flash memory space can be utilized inefficiently.

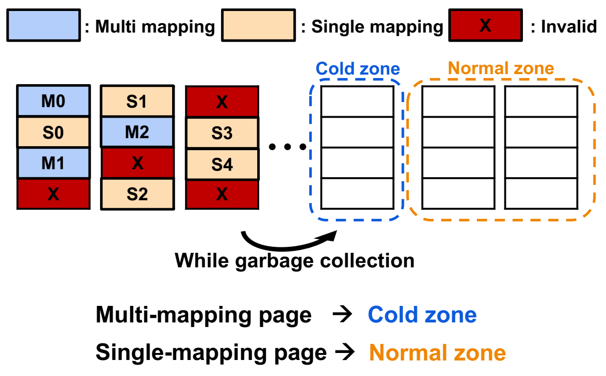


Fig. 3: Garbage collection by Coldmap

In this work, we propose Coldmap, an efficient garbage collection mechanism based on the different page-mapping classes in remap-based SSDs. Figure 3 depicts the base idea of the proposed Coldmap garbage collection scheme. As mentioned, the performance overhead by garbage collection becomes higher if long-living and short-living pages are mixed within a flash block. As described in Section III, we can accurately categorize long-living and short-living pages based on the different logical-to-physical mapping classes in the remap-based SSDs. Namely, we can expect multi-mapping pages can be categorized as *long-living* pages since our analysis reveals multi-level mapping pages can survive longer compared to single-mapping pages. Coldmap exploits such characteristics to classify long-living and short-living pages during garbage collection.

As shown in Figure 3, Coldmap copies multi-mapping pages to a block marked as a *cold zone* before the target block undergoes block erasure by the garbage collection function. On the other hand, the single-mapping pages are copied to flash blocks in a *normal zone*. Note that we expect the blocks in a *cold zone* can be erased less frequently since the long-living pages (i.e. multi-mapping pages) remain valid for a longer time. Since the *cold zone* blocks can be erased less frequently, Coldmap assigns the flash blocks that exhibit higher erase counts to a *cold zone* to increase the lifetime of NAND flash blocks. The flash blocks in a normal zone can be erased more frequently compared to the *cold zone* blocks. However, the pages in a normal zone block exhibit similar lifespans (i.e. short-living) thus we can find fewer valid blocks when these normal zone blocks are erased by garbage collection. Consequently, Coldmap can increase the lifetime of flash blocks by assigning multi-mapping pages to *cold zone* blocks. Coldmap can also reduce additional flash reads and writes by grouping flash pages expected to have similar lifespans.

### B. Proposed Coldmap Architecture

Figure 4 illustrates the overall architecture and data flows of Coldmap. Coldmap is implemented on the remap-based SSD that supports multiple logical-to-physical mapping for duplicated pages. In the figure, the orange lines represent

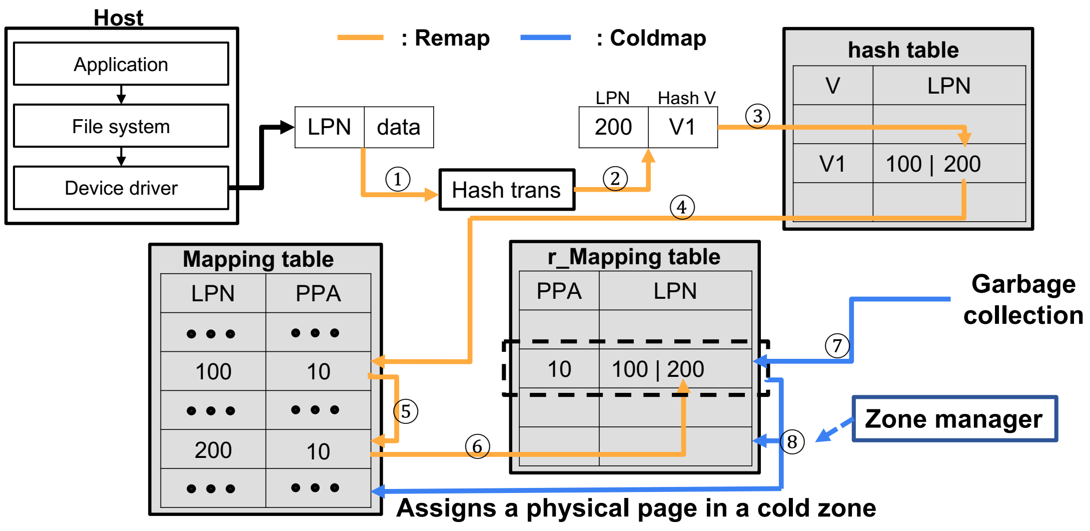


Fig. 4: Coldmap architecture

the page mapping mechanism of the remap-based SSD for supporting many-to-one flash page mapping. The blue lines describe Coldmap's garbage collection scheme that exploits the mapping information of the remap-based SSD.

When the host system requests sector writes, the NVMe write command encapsulates the logical page number (200 in this example) and page data (①). Coldmap generates the hash value (V1) of the page data to generate the hash ID based on the page data (②). Coldmap exploits *MurmurHash3* to implement a faster hash generation function with fewer hash collisions [17]. In order to support many-to-one page mapping for duplicated data, the remap-based SSD incorporates the hash table that includes mapping information of a hash ID to logical page numbers (③). In this example, the hash ID V1 is mapped to two logical page numbers 100 and 200. Base on the information of the hash table, Coldmap generates the logical-to-physical mapping table (④). Since the logical page numbers 100 and 200 are mapped to the same hash ID V1, these logical pages are associated with the same physical page number 10. In order to manage and reconstruct the mapping information, the remap-based SSD stores the reverse-mapping (i.e. physical-to-logical mapping) information in the spare area of a page (⑥).

While garbage collection is performed, Coldmap refers to the reverse-mapping information to identify single-mapping and multi-mapping (⑦). As described in the previous section, Coldmap assigns the multi-mapping pages to *cold zone* blocks (⑧). The zone manager determines which flash blocks are allocated to a *cold zone*. Coldmap marks worn-out blocks that exhibit high erase counts as *cold zone* blocks. After pages are migrated to new empty pages, Coldmap updates the mapping table (⑧).

## V. Evaluation

We evaluate Coldmap using FEMU which implements the remap-based SSD firmware [15]. We configure 16 GB of NVMe SSD and 12 GB of garbage collection threshold using FEMU. In order to activate the garbage collection function, we set up the steady-state condition by performing 12 GB of sequential writes using FIO without data duplication. Then, we run various workloads that include duplicated page writes to evaluate the garbage collection performance by Coldmap. We compare Coldmap with the baseline SSD without the remapping scheme (*baseline*) and the remap-based SSD (*Remap*). All performance metrics are normalized to the baseline configuration.

We evaluate popular storage workloads to evaluate Coldmap. FIO is configured to write 2GB of random data with a data duplication rate of 30% [16]. TPC-C handles 5 warehouse databases using MySQL [18]. Sysbench accesses 60 tables with MySQL with Pareto distributions [19]. YCSB is configured with 500K record counts using RocksDB with 20% of reads and 80% of writes [20].

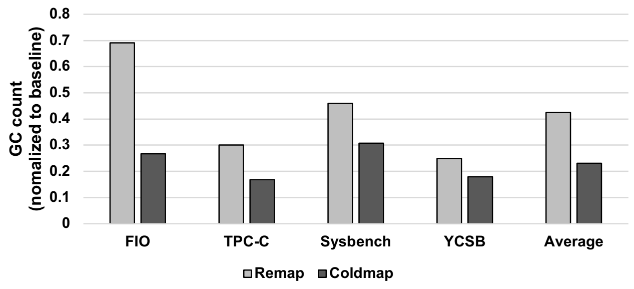


Fig. 5: Garbage collection count

### A. Garbage Collection Performance

Figure 5 exhibits the garbage collection counts normalized by the baseline configuration. Note that frequent garbage collection decreases the performance of an SSD since the garbage collection causes additional block erasure and page migrations.

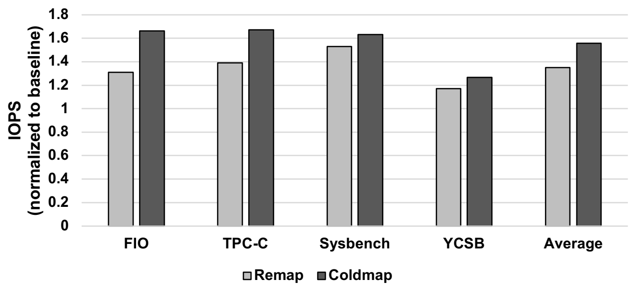


Fig. 6: Perfomance (IOPS)

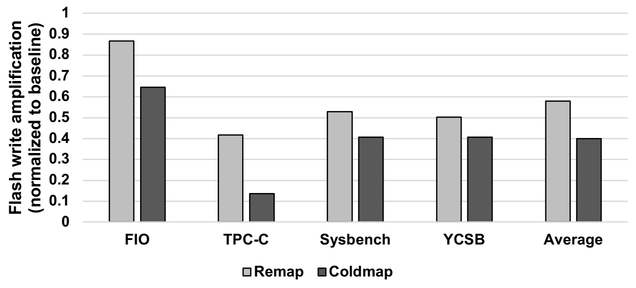


Fig. 7: Write amplification factor

Remap decreases the garbage collection counts compared to the baseline since Remap can reduce the flash memory space occupied by stored data. Our evaluation reveals that Coldmap can reduce the garbage collection counts dramatically since Coldmap can classify pages based on expected lifespans. Our evaluation results exhibit Coldmap's garbage collection count is 23.0% of the baseline SSD on average. Coldmap also reduces the garbage collection counts by 45.8% on average compared to Remap.

## B. Performance

Figure 6 exhibits the performance by Remap and Coldmap. In order to evaluate the performance of Coldmap, we measure input/output performance per second (IOPS), and the measured IOPS values are normalized to the baseline. Remap can improve the performance of an SSD by 35.0% since Remap can reduce the number of writes for duplicated data. Coldmap can improve the performance of an SSD further. Compared to the baseline and Remap, Coldmap uplifts the performance of an SSD by 55.8% and 15.4% on average, respectively. As described in Section V-A, Coldmap can reduce garbage collection counts dramatically compared to Remap. Since garbage collection requires extremely slow block erasure, the performance overhead by garbage collection is significantly increased as garbage collection counts get larger. Coldmap can improve the performance of an SSD effectively by optimizing the garbage collection mechanism based on the expected lifespans of flash pages.

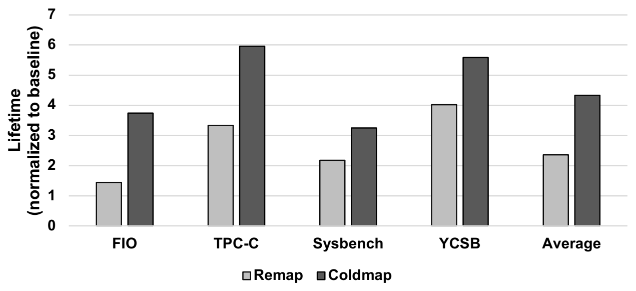


Fig. 8: SSD Lifetime

## C. Write Amplification Factor

A write amplification factor (WAF) presents the flash writes overhead by garbage collection. WAF can be calculated as shown in the equation below. A high WAF represents that an SSD performs more flash writes than the requests from a host system. Thus an SSD that exhibits a high WAF suffers from degraded performance and lifetime. WAF can be affected by the proportion of page updates and the number of garbage collection operations.

$$\mathrm{WAF} = \frac{\text{Amount of data written to the flash}}{\text{Amount of data written by the host}}$$

Figure 7 exhibits the WAF by Remap and Coldmap. Compared to the baseline Remap can reduce WAF since Remap nullifies flash writes when duplicate pages are written to the SSD. Since Coldmap is also implemented on the remap-based SSD, it can decrease flash writes when the host requests writes for duplicated page data. Furthermore, Coldmap can decrease the write amplification factor by the garbage collection. Compared to the baseline and Remap, Coldmap decrease the write amplification factor by 60.1% and 31.1%, respectively.

## D. SSD Lifetime

Since flash memory cells have limited erase counts, the number of block erases triggered by garbage collection determines the lifetime of an SSD. As described in Section IV, Coldmap employs several approaches that can increase the lifetime of flash memory. Our evaluation reveals Coldmap can increase the lifetime of an SSD by 4.34 times on average, compared to the baseline. Since Coldmap decrease the number of garbage collection, Coldmap can also increase the lifetime of an SSD by 84.3% compared to Remap.

## VI. Conclusion

In this paper, we propose Coldmap, an efficient garbage collection mechanism based on the page remapping approach for duplicated flash pages. Our observation data reveal the flash pages that store duplicated data exhibit longer lifespan before data updates. Coldmap identifies physical pages associated with multiple logical pages to group the corresponding pages into new flash blocks during garbage collection. Our evaluation results using various SSD workloads exhibit Coldmap can improve the performance and lifetime of an SSD by minimizing garbage collection counts. Coldmap exhibits 15.4% performance uplifts and 84.3% longer SSD lifetime compared to the existing remapping-based SSD solution.

## Acknowledgment

This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT) of the Government of South Korea (NRF-2021R1C1C1012172) and in part by MSIT of South Korea under the convergence security core talent training business support program (IITP-2024-RS-2022-II221198) supervised by the Institute of Information & Communications Technology Planning & Evaluation (IITP).

## References

[1] Ping Huang, Guangping Wan, Ke Zhou, Miaoqing Huang, Chunhua Li, and Hua Wang. Improve effective capacity and lifetime of solid state drives. In *2013 IEEE Eighth International Conference on Networking, Architecture and Storage*, pages 50–59. IEEE, 2013.

[2] Sungjin Lee, Taejin Kim, Kyungho Kim, and Jihong Kim. Lifetime management of flash-based ssds using recovery-aware dynamic throttling. In *FAST*, volume 3, pages 1–6, 2012.

[3] Qirui Yang, Runyu Jin, and Ming Zhao. {SmartDedup}: Optimizing deduplication for resource-constrained devices. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 633–646, 2019.

[4] Zev Weiss, Sriram Subramanian, Swaminathan Sundararaman, Nisha Talagala, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. {ANViL}: Advanced virtualization for modern {Non-Volatile} memory devices. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 111–118, 2015.

[5] Hyun Jin Choi, Seung-Ho Lim, and Kyu Ho Park. Jftl: A flash translation layer based on a journal remapping for flash memory. *ACM Transactions on Storage (TOS)*, 4(4):1–22, 2009.

[6] Feng Chen, Tian Luo, and Xiaodong Zhang. {CAFTL}: A {Content-Aware} flash translation layer enhancing the lifespan of flash memory based solid state drives. In *9th USENIX Conference on File and Storage Technologies (FAST 11)*, 2011.

[7] Yanqin Jin, Hung-Wei Tseng, Yannis Papakonstantinou, and Steven Swanson. Improving ssd lifetime with byte-addressable metadata. In *Proceedings of the International Symposium on Memory Systems*, pages 374–384, 2017.

[8] Aayush Gupta, Raghav Pisolkar, Bhuvan Urgaonkar, and Anand Sivasubramaniam. Leveraging value locality in optimizing {NAND} flash-based {SSDs}. In *9th USENIX Conference on File and Storage Technologies (FAST 11)*, 2011.

[9] Kyuhwa Han, Hyukjoong Kim, and Dongkun Shin. Wal-ssd: Address remapping-based write-ahead-logging solid-state disks. *IEEE Transactions on Computers*, 69(2):260–273, 2019.

[10] Fan Ni, Xingbo Wu, Weijun Li, Lei Wang, and Song Jiang. Leveraging ssd's flexible address mapping to accelerate data copy operations. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1051–1059. IEEE, 2019.

[11] Gihwan Oh, Chiyoung Seo, Ravi Mayuram, Yang-Suk Kee, and Sang-Won Lee. Share interface in flash storage for relational and nosql databases. In *Proceedings of the 2016 International Conference on Management of Data*, pages 343–354, 2016.

[12] Sangwook Shane Hahn, Sungjin Lee, Cheng Ji, Li-Pin Chang, Inhyuk Yee, Liang Shi, Chun Jason Xue, and Jihong Kim. Improving file system performance of mobile storage systems using a decoupled defragmenter. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 759–771, 2017.

[13] You Zhou, Qiulin Wu, Fei Wu, Hong Jiang, Jian Zhou, and Changsheng Xie. {Remap-SSD}: Safely and efficiently exploiting {SSD} address remapping to eliminate duplicate writes. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 187–202, 2021.

[14] Qiulin Wu, You Zhou, Fei Wu, Hong Jiang, Jian Zhou, and Changsheng Xie. Understanding and exploiting the full potential of ssd address remapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):5112–5125, 2022.

[15] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. The case of femu: Cheap, accurate, scalable and extensible flash emulator. In *Proceedings of 16th USENIX Conference on File and Storage Technologies (FAST)*, Oakland, CA, February 2018.

[16] Jens Axboe. Flexible i/o tester, 2022. GNU GPL v2.0.

[17] Wenlong Yi, Qiude Li, Hua Yin, Hao Tang, and Yingding Zhao. Efficient user inspection algorithm based on dual bloom filters oriented for blockchain data management systems. In *2021 XXIV International Conference on Soft Computing and Measurements (SCM)*, pages 179–182. IEEE, 2021.

[18] Vadim Tkachenko, Eddy Reyes, Mark Callaghan, and scaeloutSean. *Percona-Lab/tpcc-mysql*. 6 2018.

[19] Alexey Kopytov, Mark Callaghan, Vadim Tkachenko, Robins, Daniël van Eeden, Heinrich Schuchardt, jcfp, Daniel Black, Marc-T, Olaf Dietsche, René Cannaò, Martin Pluskal, Jay Pipes, James Page, Eric Lambert, Dillon Amburgey, Andrey Malets, Alexey Bychko, caojiafeng, Vasily Tarasov, Tomáš Mózes, FUKAI Takaaki, Peter Carrero, Paul Menzel, Patrick Bußmann, Monty Taylor, manzur, Johnathan Phan, Jeremy Cole, and Ignacio Nin. *akopytov/sysbench*. 3 2024.

[20] Sean Busbey, allanbank, Andy Kruth, Kevin Risden, Connor McCoy, Jason Tedor, Brian Cooper, Chris Larsen, Jaemyoun, stfeng2, Nitsan Wakart, gse89, Uncle Betty, Robert Lehmann, Eugene Blikh, weston-platter, Jeff Yemin, Johan Oskarsson, Kirill Vlasov, Michael Nitschinger, yuyanting, danielpoltx, Sudipto Das, Misha Brukman, Chrisjan Matser, Steffen Friedrich, Todd Lipcon, Ivan Prisyazhnyy, Bruno Michel, and Ben Stopford. *brianfrankcooper/YCSB*. 4 2024.