Optimizing Data Migration for Garbage Collection in ZNS SSDs

Zhenhua Tan[1], Linbo Long[1*], Renping Liu[1], Congming Gao[2], Yi Jiang[1], and Yan Liu[1]
[1]College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[2]School of Information Science and Engineering, Xiamen University, Xiamen 361005, China

***Abstract*—ZNS SSDs shift the responsibility of garbage collection (GC) to the host. However, data migration in GC needs to move data to the host's buffer first and write back to the new location, resulting in an unnecessary end-to-end transfer overhead. Moreover, due to the pre-configured mapping between zones and blocks, GC needs to perform a large number of unnecessary block-to-block data migrations between zones. To address these issues, this paper proposes a simple and efficient data migration method, called IS-AR, with in-storage data migration and address remapping. Based on a full-stack SSD emulator, our evaluation shows that IS-AR reduces GC latency by $6.78\times$ and improves SSD lifetime by $1.17\times$ on average.**

I. Introduction

NVMe Zoned Namespace (ZNS) interface is emerging as a promising interface standard for flash-based SSDs [1, 2]. To eliminate device garbage collection (GC), ZNS SSDs allow applications to write data sequentially to different zones with an efficient data placement [3]. However, the space occupied by invalid data in a zone can only be released after all data in the zone is invalid, resulting in low space utilization. Usually, the host GC is still required to ensure there is enough free space. Most existing related studies focus on reducing the number of data migrations, but their data migration methods are inefficient [4, 5].

On the one hand, data migration on ZNS SSDs triggers an unnecessary end-to-end transfer overhead. During GC, valid data in the victim zone needs to be first moved to the host's buffer and then written back to the new location in the target zone. On the other hand, block-to-block data migrations between zones are highly expensive, causing significant rewrite overhead. Ideally, blocks with valid data in the victim zone can be remapped to the target zone, avoiding the higher block-to-block rewrite overhead. However, the mapping between zones and blocks in ZNS SSDs is pre-configured based on the parallelism between chips [6].

Therefore, this paper proposes a simple and efficient data migration for GC in ZNS SSDs, termed IS-AR. First, a new ZNS command, named Zone_MD, is designed to implement in-storage data migration, avoiding the end-to-end transfer overhead. Then, a remapping strategy based on parallel physical blocks is proposed to reduce the high block-to-block rewrite overhead. Based on a full-stack SSD emulator with typical workloads, our evaluations exhibited that IS_AR achieved $6.78\times$ the reduction of GC latency and $1.17\times$ the improvement of SSD lifetime on average.

*Corresponding author: Linbo Long. E-mail: longlb@cqupt.edu.cn.

II. IS-AR: In-storage data migration and Address remapping

A. In-storage data migration (IS)

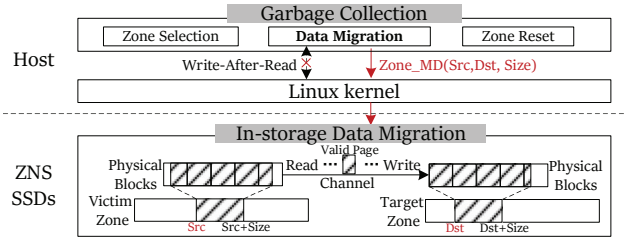

Fig. 1. The method of in-storage data migration.

As shown in Figure 1, a new ZNS command (i.e., *Zone_MD* (*Src, Dst, Size*)) is added in ZNS SSDs to help to enable in-storage data migration, where *Src* is the starting logical address of valid data in the victim zone, *Dst* is the write point (WP) of the target zone, and *Size* represents the length of the continuous valid data to be migrated. When the GC needs to migrate data, *Zone_MD* is first used to notify the device to migrate data internally. When the device receives the command, it locates the physical pages according to the starting logical address (*Src*) of valid data. After that, the data of these pages are read, transferred and rewritten to the physical pages corresponding to the destination logical address (*Dst*) through the internal channel. In this process of data migration, data is transferred only within the device, avoiding end-to-end transfer.

B. Address remapping (AR)

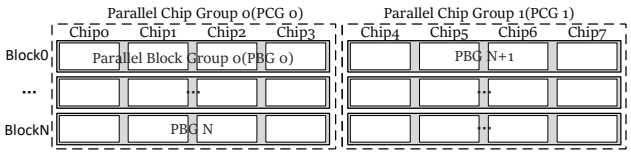

Fig. 2. A dynamic zone mapping method.

To ensure the parallelism of writing in a zone, a dynamic zone mapping method is first presented based on parallel block groups (PBGs). As shown in Figure 2, we first group the chips in SSD to form multiple parallel chip groups (PCG). Read and write operations can be processed in parallel with PCG. Then, the blocks with the same offset in the PCG are further divided into multiple PBGs. Similarly, read and write operations on a PBG can be executed in parallel. Thus, we use PBG as the remap minimum unit while the parallelism in a zone can be preserved. When a zone is allocated, we dynamically map PBGs to the zone.

978-3-9819263-7-8/DATE23/© 2023 EDAA

Based on the dynamic zone mapping, a remapping strategy is proposed to reduce the high block-to-block rewrite overhead. In the victim zone, there are two cases of PBGs that need to be migrated, i.e., the entire PBG holds valid data (V1) and only a portion of the PBG holds valid data (V2). In the target zone, the PBG that is currently being written also has two cases, i.e., all the space of the PBG is written (T1) and a portion of the PBG is not written (T2). Based on these four cases, four remapping methods are proposed, as shown in Figure 3.

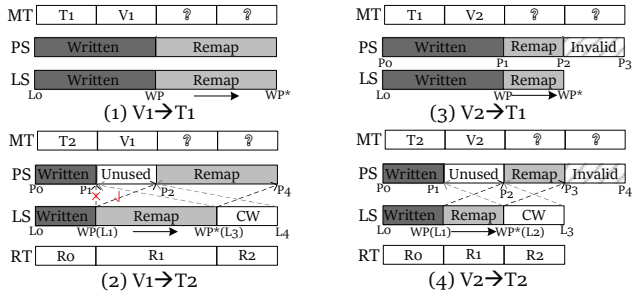

Fig. 3. The diagram of address remapping.

As shown in Figure 3 (1) (2), *V1 is remapped to the target zone containing T1 or T2*. The logical space (LS) and physical space (PS) of V1 are directly remapped to the target zone. Meanwhile, *WP* is shifted to *WP** (i.e. WP+V1_Size). In addition, the mapping between LS and PS is updated to the mapping table (MT). Further, to utilize the unused space (P1-P2) in T2, the section of LS with the size of P2-P1 after *WP** is mapped to the unused space (P1-P2). In this way, the unused space can be continued to write (CW) after remapping. To maintain the correct indexing relationship, a new remapping table (RT) is added to record the offset of the logical address from the physical address at range (i.e., a continuous segment of LS with the same offset) granularity.

As shown in Figure 3 (3) (4), *V2 is remapped to the target zone containing T1 or T2*. The entire PS of V2 is remapped directly to the PS of the target zone. However, only the LS of the valid data in V2 is remapped to the target zone since the target zone needs only valid data. The space of invalid data in V2 is reserved until the target zone is reset (i.e. part of the invalid space is temporarily occupied by target zone). Therefore, we use the current space utilization (*su*) of ZNS SSDs as a benchmark. If the percentage of valid data in V2 is greater than *su*, we perform the remapping. Otherwise, we directly rewrite all valid data in V2. In this way, we try to get a trade-off between space utilization and performance.

Based on the above, remapping can minimize the block-to-block rewrite overhead as much as possible. Further, we weigh the benefits and costs of remapping to reduce the overhead caused by remapping.

III. EVALUATION

To evaluate IS-AR, we implemented it in ZNS SSD based on *FEMU* [7]. The detailed configurations are shown in Table I. In addition, we compare IS-AR with the traditional data migration method (termed Trad-DM), which does not implement in-storage data migration and remapping. Further, we utilized *db_bench* (RocksDB's internal benchmark) to generate six classic workloads. A comparison of the performance of IS-AR and Trad-DM under six workloads is shown in the table II, where the first column is the elapsed time of data migration (i.e. latency) during GC, and the second column is the total number of physical block erasures (i.e. lifetime loss). Compared with Trad-DM, IS-AR can not only significantly reduce the latency of data migration during GC by an average of $6.78\times$, but also greatly improve the lifetime of ZNS SSDs by $1.17\times$ on average under six workloads.

TABLE I
EXPERIMENT SETUP.

Configuration	Description
FEMU	FEMU version: 5.2; Linux kernel: 5.12
SSD	Channels: 8, Chips/Channel: 4; Dies/Chip: 1, Planes/Die: 4; Blocks/Plane:256, Pages/Block: 256; Page Capacity: 4KB
ZNS	Zone size: 128MB; Number of Zones: 256; Zone Parallelism: 32
Flash latency	Read latency: 40us; Program latency: 200us; Erase latency: 2000us
Software Version	RocksDB Version: 7.3; Zenfs Version: 2.0.1
db_bench	Key Size: 128bytes; Value Size: 8192bytes; Max SST File Size: 64MiB; I/O Mode: Direct I/O

TABLE II
PERFORMANCE COMPARISON BETWEEN IS-AR AND TRAD-DM

Workloads	Latency(s)		Lifetime loss	
	Trad-DM	IS-AR	Trad-DM	IS-AR
Random	38.49	6.95	192768	170067
Random+Overwrite	33.28	6.41	181453	158880
Random+Updaterandom	45.22	5.71	232640	209488
Seq	37.81	5.1	59236	41813
Seq+Overwrite	64.45	9.37	114368	92885
Seq+Updaterandom	95.13	12.82	183296	153248

IV. CONCLUSIONS

In this paper, we present IS-AR, which offloads the responsibility of data migration between zones to the device through the new ZNS interface, Zone_MD, and deploys *address remapping* on the device to further improve the efficiency of data migration. Further, we significantly demonstrate that IS-AR improves the efficiency of data migration during GC and the lifetime of ZNS SSDs.

ACKNOWLEDGMENT

This work was supported by grants from the National Natural Science Foundation of China 61902045, 62102219 and 62172067, Chongqing High-Tech Research Key Program cstc2021jcyj-msxmX0981, cstc2021jcyj-msxmX0530, and cstb2022nscq-msx0601.

REFERENCES

[1] "NVMe Zoned Namespaces," https://zonedstorage.io/docs/introduction/zns.
[2] R. Liu, Z. Tan, Y. Shen, L. Long, and D. Liu, "Fair-ZNS: Enhancing Fairness in ZNS SSDs through Self-balancing I/O Scheduling," *TCAD*, 2022.
[3] M. Bjørling, A. Aghayev, H. Holmberg, and et al., "ZNS: Avoiding the Block Interface Tax for Flash-based SSDs," in *ATC*, 2021, pp. 689–703.
[4] G. Choi, K. Lee, M. Oh, and et al., "A New LSM-style Garbage Collection Scheme for ZNS SSDs," in *HotStorage*, 2020.
[5] G. Oh, J. Yang, and S. Ahn, "Efficient Key-Value Data Placement for ZNS SSD," *Applied Sciences*, vol. 11, no. 24, p. 11842, 2021.
[6] K. Han, H. Gwak, D. Shin, and et al., "ZNS+: Advanced Zoned Namespace Interface for Supporting In-storage Zone Compaction," in *OSDI*, 2021, pp. 147–162.
[7] H. Li, M. Hao, M. H. Tong, and et al., "The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator," in *FAST*, 2018, pp. 83–90.