WA-Zone: Wear-Aware Zone Management Optimization for LSM-Tree on ZNS SSDs

LINBO LONG, Chongqing University of Posts and Telecommunications, China
SHUIYONG HE, Chongqing University of Posts and Telecommunications, China
JINGCHENG SHEN, Chongqing University of Posts and Telecommunications, China
RENPING LIU, Chongqing University of Posts and Telecommunications, China
ZHENHUA TAN, Chongqing University of Posts and Telecommunications, China
CONGMING GAO, Xiamen University, China
DUO LIU, Chongqing University, China
KAN ZHONG, Huawei Technologies, Co., Ltd., China
YI JIANG, Chongqing University of Posts and Telecommunications, China

ZNS SSDs divide the storage space into sequential-write zones, reducing costs of DRAM utilization, garbage collection, and over-provisioning. The sequential-write feature of zones is well-suited for LSM-based databases, where random writes are organized into sequential writes to improve performance. However, the current compaction mechanism of LSM-tree results in widely varying access frequencies (i.e., hotness) of data and thus incurs an extreme imbalance in the distribution of erasure counts across zones. The imbalance significantly limits the lifetime of SSDs. Moreover, the current zone-reset method involves a large number of unnecessary erase operations on unused blocks, further shortening the SSD lifetime.

Considering the access pattern of LSM-tree, this paper proposes a wear-aware zone-management technique, termed *WA-Zone*, to effectively balance inter- and intra-zone wear in ZNS SSDs. In WA-Zone, a wear-aware zone allocator is first proposed to dynamically allocate data with different hotness to zones with corresponding lifetimes, enabling an even distribution of the erasure counts across zones. Then, a partial-erase-based zone-reset method is presented to avoid unnecessary erase operations. Furthermore, because the novel zone-reset method might lead to an unbalanced distribution of erasure counts across blocks in a zone, a wear-aware block allocator is proposed. Experimental results based on the *FEMU* emulator demonstrate the proposed WA-Zone enhances the ZNS-SSD lifetime by 5.23×, compared with the baseline scheme.

Additional Key Words and Phrases: ZNS SSDs, LSM-tree, wear-leveling, data allocation

1 INTRODUCTION

The NVMe Zoned Namespace (ZNS) is emerging as a promising storage interface for flash-based SSDs [1, 40]. ZNS SSDs group multiple blocks that reside on different flash chips into a sequential-write zone, which reduces the amount of in-device DRAM allocated for the mapping table [3, 18, 28, 36]. Meanwhile, ZNS SSDs transfer the responsibility of data management to the host, thereby avoiding in-device garbage collection (GC) and over-provisioning (OP) [35, 39]. Ideally, the host can classify the data of a specific application based on lifetime according to the access pattern of the application and allocate data with a similar lifetime into the same zone. In this way, all data on a zone is more likely to be invalidated simultaneously, thus further reducing write amplification caused by data migration.

This work was supported by grants from the National Natural Science Foundation of China 62102219, 61902045 and 62172067, Chongqing High-Tech Research Key Program cstc2021jcyj-msxmX0981, cstc2021jcyj-msxmX0530, and cstb2022nscq-msx0601, and Chongqing Natural Science Foundation Innovation and Development Joint Fund CSTB2022NSCQ-LZX0074.
Authors' addresses: L. Long, S. He, J. Shen, R. Liu, Z. Tan, Y. Jiang, School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, 2 Chongwen Rd, Nan'an District, Chongqing, 400065, China; C. Gao, School of Information Science and Engineering, Xiamen University, Scientific Research Building #1, Haiyun Park, Xiamen, Fujian, 361005, China; D. Liu, School of Big Data & Software Engineering, Chongqing University, 174 Shazheng Rd, Shapingba District, Chongqing, 400044; K. Zhong, Huawei Technologies Co., Ltd., Longgang District, Shenzhen, Guangdong, 518129, China. Corresponding author: J. Shen, e-mail: shenjc@cqupt.edu.cn.

The Log-Structured Merge-Tree (LSM-tree) is a high-performance data structure for write-intensive workloads and is widely adopted by NoSQL databases, such as Google's LevelDB [33], Facebook's RocksDB [12], and Alibaba's X-Engine [19]. To achieve high write throughput, LSM-tree converts random writes on storage into sequential writes at the expense of reading performance and space usage. Meanwhile, a leveled compaction policy is widely utilized to organize the LSM-tree data into multiple sorted levels of a limited size to reduce space amplification and read amplification [7, 11, 26, 42, 46]. The sequential writes secured by the LSM-tree can be efficiently handled by the sequential-write zones of ZNS SSDs. However, the large number of extra writes involved by the compaction mechanism of LSM-tree might significantly reduce the lifetime of ZNS SSDs.

Considering the access pattern of LSM-tree, existing work focuses on resolving the issues related to space amplification and write amplification incurred by using LSM-tree on ZNS SSDs. A segment-based GC scheme is designed to assign cold segments to a zone, isolating cold segments from others [9]. An efficient key-value data placement is proposed to allocate the data at the same level to a zone because these data have similar lifetimes [34]. Since data (i.e., SSTable in LSM-tree) with overlapping key ranges are likely to be merged, Lee *et al.* [24] propose to place data with overlapping key ranges in the same zone. Moreover, to well cooperate with the current LSM-tree compaction mechanism, Jung *et al.* [22] further distinguishes long-lived data from short-lived data at the same level to ensure that the data in each zone have similar lifetimes.

The aforementioned work focuses on allocating the data with a similar lifetime into a zone to reduce write amplification. Nevertheless, the previous work ignores the inter-zone wear-leveling issue in ZNS SSDs, i.e., data on LSM-tree widely vary in lifetime. Therefore, a sophisticated method is necessary to allocate proper zones to data. Otherwise, short-lived data might be frequently assigned to a small number of zones, resulting in an extremely unbalanced distribution of erasure counts across zones. The extremely unbalanced wear across zones can shorten the lifetime of ZNS SSDs. To make matters worse, a zone can be reset when all the data residing on it becomes invalid, even if a large portion of the zone remains unused. Such a zone-reset scheme can cause many unnecessary erase operations on unused blocks, further reducing the SSD lifetime. To address these issues, we therefore design a novel zone-management technique that exploits the LSM-tree access patterns to improve the lifetime of ZNS SSDs.

In this paper, we propose a wear-aware zone-management technique, termed *WA-Zone*, which comprises inter-zone and intra-zone management. In inter-zone management, a wear-aware zone allocator is designed to dynamically allocate the hottest data in the lower level of the LSM-tree to the zone with the least wear to perform wear leveling across zones. In intra-zone management, a partial-erase-based zone-reset method is first proposed to eliminate unnecessary erasure operations on unused blocks. Then, a wear-aware block allocator is implemented to evenly distribute erasure counts across blocks in a zone. To effectively balance inter- and intra-zone wear in ZNS SSDs, we implement the two management schemes and integrate them into file systems and devices, respectively.

The contribution of this work is summarized as follows:

- Considering the access pattern of LSM-tree, a wear-aware zone allocator is proposed to effectively balance erasure counts across zones to improve the lifetime of ZNS SSDs.
- A partial-erase-based zone-reset method is presented to eliminate the unnecessary erase operations on unused blocks during zone reset. Furthermore, a wear-aware block allocator is designed to achieve wear leveling across blocks in a zone.

- We implement the proposed WA-Zone based on a modified emulator and conduct in-depth experiments to evaluate the effectiveness of the proposed techniques applied to typical workloads.

The remainder of this paper is structured as follows. Section 2 begins with a brief background on LSM-tree and ZNS SSDs. Then, Section 3 gives a motivation analysis. After that, we present the WA-Zone in detail in Section WA-Zone. Finally, Section 5 shows the evaluation results of the proposed techniques, Section 6 provides a literature review of related work, and Section 7 concludes this paper.

2 BACKGROUND

In this section, we first describe the LSM-tree architecture. The background of ZNS SSDs is then given. In addition, we analyze the lifetime issue of ZNS SSDs when they handle the LSM-tree data.

2.1 LSM-tree

LSM tree is a high-performance, write-efficiency data structure. Compared with other data structures (e.g., B+ tree), the LSM-tree data structure exhibits a higher performance on write-intensive workloads, which benefits from converting random write operations into sequential write operations [31]. Currently, the data compaction strategies of the LSM tree can be mainly divided into size-tiered compaction and leveled compaction. Leveled compaction can achieve higher read performance and less space requirement than size-tiered compaction, and is widely adopted by many modern read- and write-intensive databases.

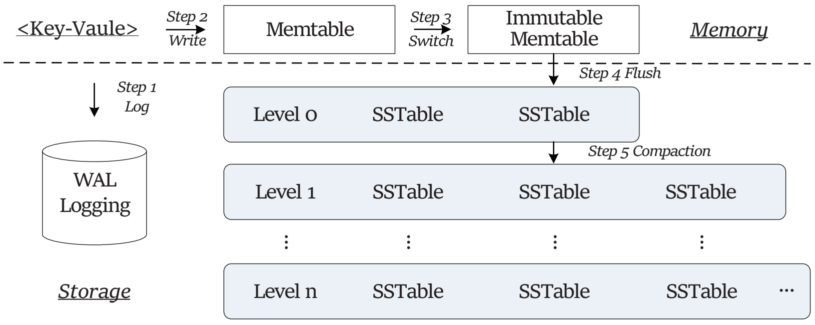


Fig. 1. Architecture of the LSM tree and its write operation based on leveled compaction.

Figure 1 shows the architecture and the write operation of the LSM tree based on leveled compaction. When a key-value pair arrives, it will be first written into two places including WAL (*step 1*) and Memtable (*step 2*). WAL is a write-ahead log that is used for data recovery. Memtable is stored in a memory buffer and provides interfaces for writing, reading, and deleting key-value data. A special data structure, such as a jump table or a red-black tree, is usually utilized to sort the data in Memtable by key. When the size of Memtable arrives at a pre-configured threshold, it will be switched into a read-only Memtable (ImmuTable Memtable) (*step 3*).

After the memory buffer is full, multiple Immutable Memtables are merged into one SSTable (Sorted String Table) and are flushed to Disk/SSD storage (*step 4*). Especially, SSTables are further organized into multiple limited-size levels. The new SSTable is first written into Level 0. If the new size of Level 0 exceeds the pre-configured size of Level 0, an SSTable in Level 0 will be selected and merged with some SStables (overlapping with the key ranges of the SSTable in Level 0) in Level 1 to form some new SStables of Level 1 (*step 5*). If the new size of Level 1 still exceeds its pre-configured size, leveled compaction is repeated until the last level. Leveled compaction ensures that the key-value data at the same level is unique and ordered. Therefore, leveled compaction reduces the read amplification and the space amplification of the traditional key-value stores.

Due to the level-by-level compaction, SSTables on lower levels (e.g., Level 0, Level 1, and Level 2) will have more opportunities to be compacted [41]. As a result, the lower-level SSTables might be frequently updated, and their lifetimes are usually less than the higher-level SSTables. Most writes/updates in LSM-tree focus on the lower-level SSTables, leading to an extremely unbalanced write distribution. Furthermore, the unbalanced write distribution will significantly reduce the limited lifetime of SSDs.

2.2 The Architecture of ZNS SSDs

As the capacity and density of SSDs increase, the traditional block-interface SSDs suffer from many challenges, such as the large DRAM cost for page mapping tables, the write amplification, and over-provisioning caused by in-device GC [27, 29, 38]. ZNS, which evolved from the ZAC/ZBC standard (used for Shingled Magnetic Recording Hard Disk Drives) [8, 45, 47] and open-channel SSD [2, 44], provides an effective solution to these challenges. Meanwhile, the ZNS command set is standardized in the NVMe 2.0 specification [43] and has been widely studied as a next-generation interface. Currently, many SSD manufacturers, such as Samsung, SK Hynix, and Western Digital, have already released their new enterprise ZNS SSDs.

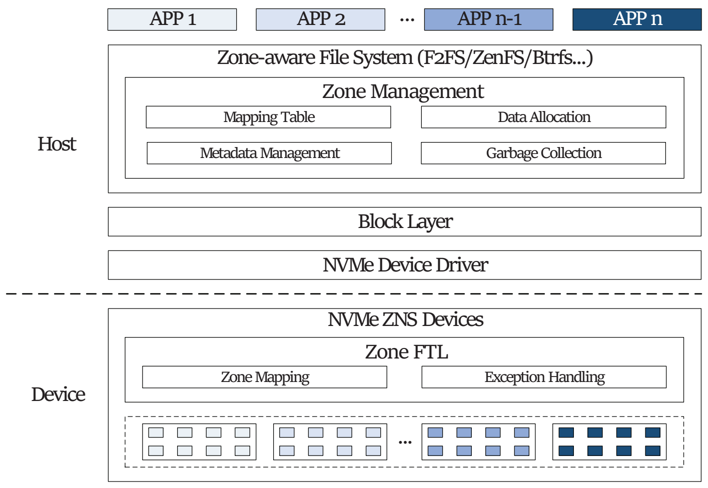


Fig. 2. Architecture of the ZNS SSD.

Figure 2 shows a typical architecture of ZNS SSDs. Multiple blocks on different flash chips are first grouped into a zone [20]. In this way, the logical address space of ZNS SSDs is divided into fixed-size (e.g., 512MB) zones that can be read randomly but must be written sequentially. Compared to block-interface SSDs, ZNS SSDs exhibit many significant benefits. First, reducing the DRAM cost for the mapping table in devices. Only the coarse-grained mapping table between zone and block is pre-configured in the device. Second, avoiding in-device GC and over-provisioning. The host is directly responsible for data allocation on zones and GC, simplifying the functionality of SSDs' Flash Translation Layer (FTL). Zone-aware file systems, such as F2FS [16], ZenFS [17], and

Btrfs [48], have added the associated module to support zone management. Third, reducing the write amplification. Since the in-device GC is avoided, the device-side write amplification is directly eliminated. Moreover, the host can utilize the access patterns of applications to execute an effective data allocation on zones, in order to further reduce the write amplification by data migration. Therefore, according to the access patterns of applications, many data placement techniques are proposed to minimize the write amplification on ZNS SSDs.

2.3 Mapping Mechanisms for ZNS SSDs

Since the mapping scheme of ZNS SSDs remains undefined in the NVMe ZNS specifications, various mapping schemes can be utilized by the drive vendors to implement zone mapping to erase blocks for a specific purpose. Typically, the mapping mechanisms can be categorized into *static mapping* and *dynamic mapping*.

In the *static mapping* scheme, a zone is statically mapped to multiple fixed erase blocks, which reduces or even eliminates the space overhead of the zone-to-block mapping table and the time overhead of address translation [23]. Moreover, the internal structure of each zone (e.g., the interference between zones) becomes more transparent to the host, making it easier to co-optimize with the access characteristics of the application. The access performance of static mapping becomes more predictive.

On the other hand, the *dynamic mapping* scheme exhibits higher flexibility, allowing drive vendors to dynamically map specific blocks to a zone when the zone is opened [16]. Meanwhile, it can achieve higher scalability for various applications than the static mapping scheme. As a result, various techniques based on dynamic mapping have been proposed to improve access parallelism or wear leveling [18, 32]. Nevertheless, compared to the static mapping scheme, the dynamic mapping scheme incurs more space and time overheads. Furthermore, given the dynamic nature of the relationship between zones and resources within the ZNS SSD, it becomes much more complicated and elusive to devise collaborative optimizations that exploit both application access patterns and ZNS SSD features.

The two mapping schemes have their advantages and shortcomings. Drive vendors can choose a specific mapping scheme based on a specific purpose. Since the internal structure of the static mapping ZNS SSD is more transparent to the host, this paper concentrates on the wear-leveling optimization of the static mapping ZNS SSD to fully exploit the data access pattern of the LSM-based databases.

2.4 Wear-leveling Mechanisms for ZNS SSDs

Wear leveling is typically used to prolong the lifetime of the flash-based SSD and can be implemented on either the host side or the device side, depending on the FTL implementation. In traditional SSDs, device-side wear leveling is widely studied since traditional SSDs maintain the L2P table in DRAM on the SSD and the host is unable to directly access the information of the internal resources. Nevertheless, the device cannot effectively obtain knowledge related to the access pattern of applications. Accordingly, the device-side wear leveling usually utilizes the principle of temporal locality to identify the hot data and allocate the block with minimal erasure counts to the hot data [30]. Emerging SSDs, such as open-channel SSDs and ZNS SSDs, are proposed to realize most of the FTL functionalities in the host. Typically, since the information about internal resources is exposed to the host and the host can more effectively sense application access patterns, collaborative optimizations regarding both application access patterns and device characteristics can be performed to improve the performance of SSDs (e.g., wear leveling) [2, 44].

To achieve wear leveling on ZNS SSDs, existing studies mainly focus on the in-device wear-leveling techniques, which can be divided into two categories. One method proposes to map free erase blocks to a new zone in a polling fashion [16]. On the other hand, these erase blocks with minimal erase counts are preferentially mapped to a new zone when it is opened [32]. The above related work can be summarized as dynamic wear leveling, which can efficiently achieve wear leveling on free blocks. However, these blocks on the opened zones holding cold data cannot be immediately recycled and still result in an unbalanced distribution of erasure counts across zones. Moreover, data on LSM tree-based databases exhibit widely varying lifetimes, although the write amplification can be minimized by allocating the data with similar lifetimes into a zone. Without an effective wear leveling technique between zones, erasure counts between zones may exhibit an extremely unbalanced distribution, which will shorten the lifetime of SSDs. Therefore, this paper proposes a new wear-aware zone allocator to improve the lifetime of ZNS SSDs.

2.5 The Current Zone Reset Mechanism for ZNS SSDs

ZNS SSDs transfer the responsibility of zone reclamation to the host, which is usually implemented in file systems, like zone cleaning in ZenFS [3] and segment cleaning in F2FS [16]. Zone reclamation is triggered when the free space is lower than a specified threshold. Typically, a victim zone with less valid data is first selected. Then, all valid data on the victim zone are migrated to other zones. Subsequently, the victim zone is reset to free this zone space. However, a large data migration overhead might be incurred, which will cause a series of issues, such as performance degradation and I/O blocking [15].

Thereby, a state-of-the-art runtime zone-reset scheme is implemented in current zone-based file systems (e.g., ZenFS) [5]. Its core idea is to reset a zone when all data on the zone become invalid. The runtime zone-reset scheme can effectively reduce the frequency of zone reclamation calls. However, even if a zone has a lot of unused space, as long as all data on the zone becomes invalid, the zone will be reset. Moreover, a zone-reset command erases all blocks on the zone including unused blocks. These unnecessary erase operations on unused blocks further reduce the SSD lifetime. To this end, this paper further presents a partial-erase-based zone-reset method to reduce the unnecessary erase operations on unused blocks.

3 ACCESS PATTERN AND MOTIVATION

In this section, we first analyze the distribution of erasure counts across zones in ZNS SSDs based on typical workloads of *RocksDB* [13] (refer to Section 5 for the detailed experiment configuration). Then, we discuss the unnecessary erase operations on unused blocks. Finally, the motivation of this paper is given based on these analyses.

3.1 Unbalanced Distribution of Erasure Counts across Zones

Available zones are randomly allocated without considering their lifetimes, which causes an extremely unbalanced distribution of erasure counts across zones. Precisely, to avoid write amplification, the current zone-allocation method allocates SSTables at the same level (i.e., lifetime) of an LSM-tree to the same zone. However, such a zone allocation method randomly allocates a new zone for new data regardless of data hotness. Consequently, hot data may be concentrated in a small number of zones, which incurs an unbalanced distribution of erasure counts across zones.

Figure 3 shows the distribution of erasure counts across zones after 10 million and 20 million key pairs were written. Apparently, most of the erasure counts are owned by a small number of zones. On average, 80% of erasure counts are owned by 32.47% of zones. In particular, a large proportion of zones are rarely or never erased. More precisely, about 50% of zones were never erased after 10 million key-value pairs were written and 30% of zones remained not erased after 20 million key-value pairs were written. A conclusion can therefore be drawn that the current zone-allocation

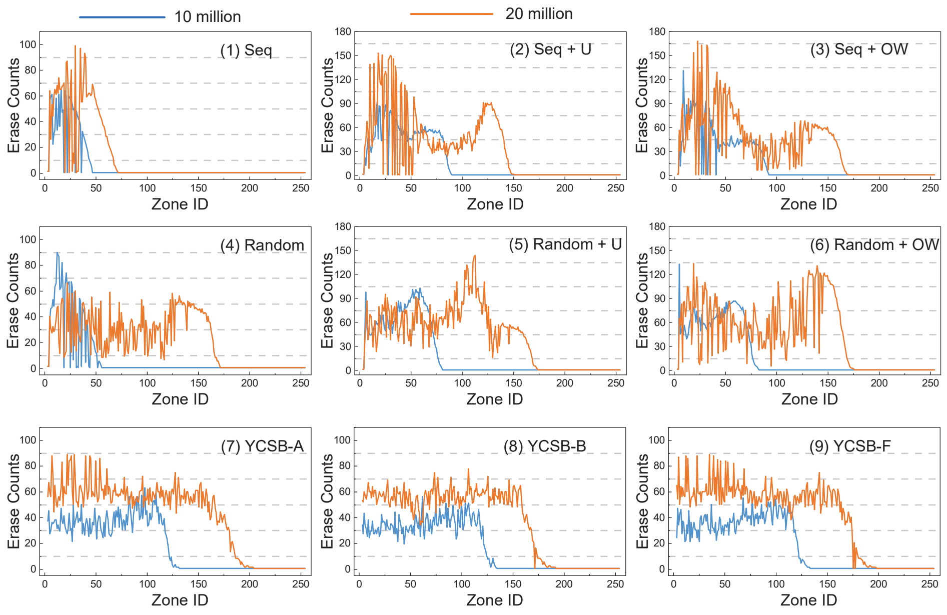


Fig. 3. Distribution of erasure counts among zones in ZNS SSDs.

method results in an extremely unbalanced distribution of erasure counts across zones. Such an imbalance in the distribution of erasure counts significantly reduces the lifetime of SSDs.

3.2 Unnecessary Erase Operation on Unused Blocks

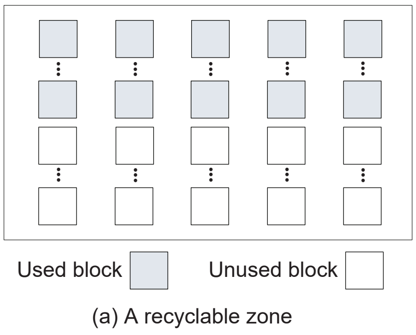
Typically, a zone is reset when all data residing on it becomes invalid without regard to many unused blocks, which involves a large number of unnecessary erases on unused blocks. As shown in Figure 4a, a recyclable zone may contain many unused blocks, and both used and unused blocks are erased.

Figure 4b illustrates the space utilization of 4,000 zones when they were reset in various workloads. The average space utilization of all the zones is 70%. In particular, 32.35% of zones have a space utilization lower than 30%. When these zones are reset, more than 70% of blocks of them are unused. Moreover, 37.60% of zones have a space utilization lower than 50%. Considering the large proportion of zones that hold many unused blocks, a more sophisticated zone-reset method should be proposed to reduce the unnecessary erasure operations and thereby enhance the lifetime of SSDs.

Summarily, this work is motivated by the aforementioned observations that the current zone-allocation method causes an extremely unbalanced distribution of erasure counts across zones and that the current zone-reset method involves a large number of unnecessary erasures on the unused blocks. A new zone-reset method can be designed to erase only the used block in a zone. However, such a method may result in an unbalanced distribution of erasure counts across blocks in a zone, reducing the lifetime improvement. To these ends, a novel wear-aware zone-management technique, WA-Zone, is proposed to achieve inter- and intra-zone wear leveling.

4 WA-ZONE: WEAR-AWARE ZONE MANAGEMENT OPTIMIZATION FOR LSM-TREE ON ZNS SSDS

The WA-Zone technique adopts novel management strategies to achieve an overall wear leveling for ZNS SSDs where LSM tree-based applications perform (Fig. 5). According to the access pattern



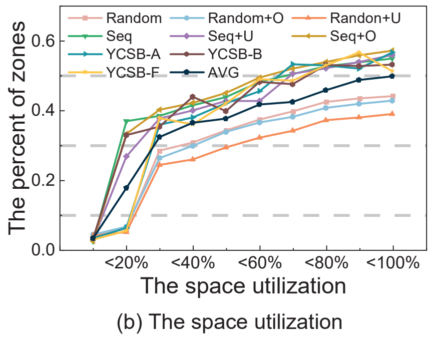


Fig. 4. Utilization of blocks within a zone: (a) a recyclable zone that may be reset even if it has many unused blocks, and (b) the space utilization after 4,000 zone resets.

of LSM-based applications, WA-Zone first dynamically allocates data with different hotness to zones with corresponding lifetimes to achieve wear-leveling across zones. Furthermore, WA-Zone adopts a novel zone-reset method to eliminate unnecessary erase operations on unused blocks. Moreover, to fully benefit from the new zone-reset method, a new block-allocation strategy is designed to evenly distribute erasure counts across blocks in a zone.

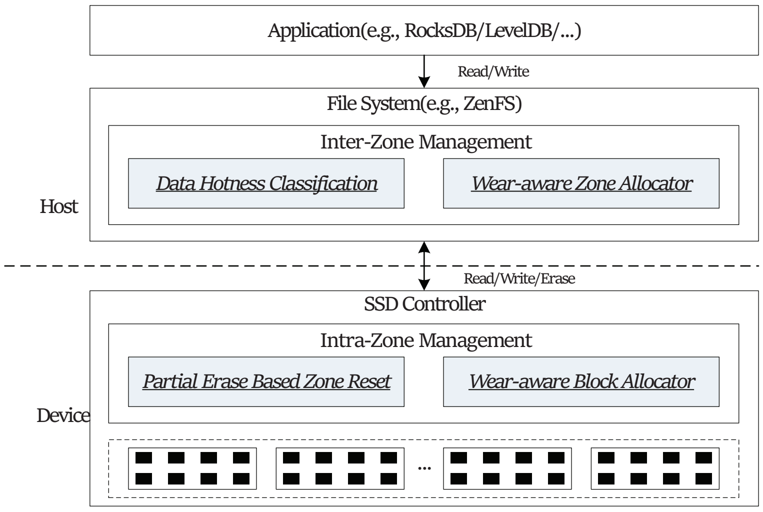

Application(e.g., RocksDB/LevelDB/...)
Read/Write
File System(e.g., ZenFS)
Inter-Zone Management
Data Hotness Classification
Wear-aware Zone Allocator
Host
Read/Write/Erase
SSD Controller
Intra-Zone Management
Partial Erase Based Zone Reset
Wear-aware Block Allocator
Device
...

Fig. 5. Overview of WA-Zone.

As to the implementation of WA-Zone, a data hotness classification module is first integrated into the inter-zone management part of the file system (e.g., ZenFS) to classify the hotness of data for LSM-based applications (e.g., RocksDB and LevelDB). Secondly, Considering data hotness, a wear-aware zone-allocate module is implemented to balance the wear across zones. The module allocates the hottest data to the zone with the least wear and writes the coldest data to the zone with the most wear. In this way, we can achieve wear leveling across zones. Thirdly, a partial-erase-based zone-reset module is integrated into the intra-zone management part of the SSD controller to avoid unnecessary erase operations on unused blocks when a zone is recycled. Finally, a wear-aware block-allocate module is implemented to evenly distribute erasure counts across all the blocks in a zone.

The remainder of this section will present in detail the aforementioned WA-Zone modules that are grouped into two parts: inter- and intra-zone management.

4.1 Inter-Zone Management

This section presents the data-hotness classification and wear-aware zone allocation modules. The modules are designed to achieve wear leveling across zones.

4.1.1 Data Hotness Classification. Due to the level-by-level compaction mechanism, data on the same level have the same opportunities to be compacted. Moreover, data at lower levels have more opportunities to be compacted. Typically, data at the same level have similar lifetimes. Meanwhile, data on lower levels have a shorter lifetime compared to data on higher levels. Therefore, we can effectively classify the lifetime of data by its level in the LSM tree. Many related works focus on allocating the data within one level into a zone to reduce write amplification. In contrast, this paper focuses on dynamically allocating data with different hotness to the corresponding zones to balance the wear across zones. Accordingly, we conduct a series of experiments to analyze the differences in the lifetime of data at different levels (refer to Section 5 for the detailed experimental configuration), as shown in Figure 6a. Experimental results show two intriguing findings as follows.

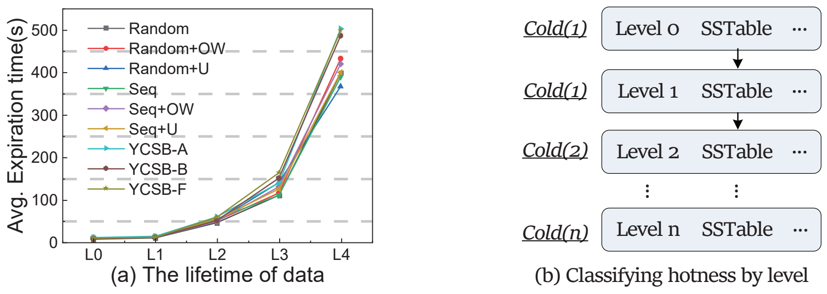


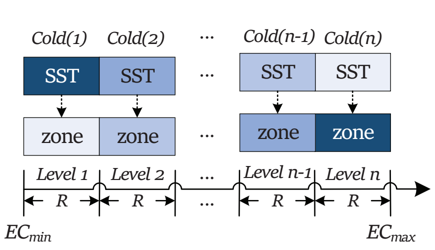
Fig. 6. Data hotness classification.

(1) The data at Level 0 and Level 1 have a similar lifetime. When the memory buffer is full, ImmuTable Memtables are merged into an SSTable which is stored at Level 0. Level 0 holds the most recently written data. Data is written to the next level only when the size of Level 0 reaches a pre-configured threshold. Therefore, Level 0 stores the most short-lived data in an LSM-tree. Because the data at Level 0 comes from Immutable Memtables and is not sorted by key, most data at Level 1 are frequently compacted with the data at Level 0. As a result, the data at Level 0 and Level 1 have a similar lifetime.

(2) For data at levels higher than Level 1, the lifetime of high-level data is several times as long as that of low-level data. The data at any level higher than Level 1 is sorted by key. When the data size at a level reaches a pre-configured threshold, an SSTable at this level will be selected to be merged with a next-level SSTable with overlapping key ranges. In this way, the space amplification by data compaction is greatly reduced. Figure 6a shows cases that data at different levels widely vary in lifetime. For instance, the average lifetime of the data at Level 2 is 6 times as long as that of the data at Level 1 and the average lifetime of the data at Level 3 is 10 times as long as that of the data at Level 1.

$$Cold(i) = \begin{cases} 1, i = 0 \\ i, 0 < i \leq n \end{cases} \quad (1)$$

Based on the above observations, we redefine *Cold(i)* to represent the data hotness at the i-th level, as shown in Equation (1). A smaller value of i means that the data is more likely to have been updated recently. Since the data at Level 0 and Level 1 have a similar lifetime, we classify these data into a group of *Cold(1)* (Fig. 6b). For data at levels higher than Level 1, we define the hotness value based on the number of the level where the data belongs, i.e, the hotness level of data at Level i is *Cold(i)* ($0 < i \leq n$, n is the maximum level of the LSM-tree). Based on the data hotness information, we can dynamically assign data to appropriate zones and thereby achieve the wear leveling across zones.



(a) Grouping zones into n levels

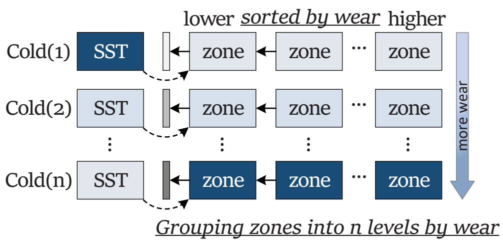

lower sorted by wear higher
Cold(1)
Cold(2)
Cold(n)
SST
zone
more wear
Grouping zones into n levels by wear
(b) Wear-aware zone allocator

Fig. 7. Wear-aware zone allocator.

4.1.2 Wear-aware Zone Allocator. The wear-aware zone allocator is proposed to further balance the wear across zones. As shown in Figure 7a, the core idea is to sort the zones by wear (i.e., erasure counts) and classify zones into as many groups as the number of the data hotness levels. Then, a free zone allocator is implemented to assign data to the corresponding zone such that the data hotness is equal to the wear level of the zone. Consequently, hot data can be stored in zones with small wear levels, and vice versa. Ideally, we can obtain the wear leveling across zones.

However, since the cold data might always remain untouched, zones with large wear levels that initially hold cold data will become zones with small wear levels, resulting in an unbalanced distribution of erasure counts across zones. To further balance the wear across zones, a cold-data migration algorithm is proposed to migrate cold data from zones with small wear levels to zones with large wear levels. We thus present a zone grouping method, a free zone allocator, and a cold-data migration algorithm as follows.

(1) Grouping zones into n levels by wear. As shown in Figure 7b, we first get the maximal erasure counts (EC_{max}) and the minimal erasure counts (EC_{min}) among zones of a ZNS SSD. Then, the range between EC_{max} and EC_{min} is divided into n segments of a length of R. Note that n is the number of the data hotness levels and R can be calculated by Equation 2.

$$R = \frac{EC_{max} - EC_{min}}{n} \quad (2)$$

According to the erasure counts, zones are assigned to the n range groups. Ideally, data should be allocated to a zone with a level number equal to the data hotness, i.e., the data with the hotness i should be written to the zones at Level i. Each zone group manages two lists: *a free-zone list* and *a used-zone list*.

Algorithm 1 Free Zone Allocator.

```
Input: (1) $n$: number of zone levels, (2) $Z$: free zone lists sorted by wear where $Z_{i,j}$ represents the
    $j$-th zone in the $i$-level free-zone list, (3) $J_i$: number of zones in the $i$-level free-zone list , (4) $h$:
    hotness of the data to be stored
Output: a free zone to allocate
    /*Allocate the minimal-wear zone in the $h$-levels free zone list to the data*/
 1: if $J_h > 0$ then
 2:     return $Z_{h,1}$
 3: end if
    /*Look for the maximal-wear zone ($Z_{mi,mj}$) in the free-zone lists from Level (h-1) to Level 1*/
 4: for $i = h - 1$ to 1 do
 5:     if $J_i > 0$ then
 6:         $mi \leftarrow i, mj \leftarrow J_i$
 7:         Calculate the difference ($DM$) between the wear of $Z_{mi,mj}$ and that of the $h$-level range
            group
 8:     end if
 9: end for
    /*Look for the minimal-wear zone ($Z_{si,sj}$) in the free zone lists from Level (h+1) to Level n*/
10: for $i = h + 1$ to n do
11:     if $J_i > 0$ then
12:         $si \leftarrow i, sj \leftarrow 1$
13:         Calculate the difference ($DS$) between the wear of $Z_{si,sj}$ and that of the $h$-level range group
14:     end if
15: end for
    /*Choose between $Z_{si,sj}$ and $Z_{mi,mj}$*/
16: if $DM <= DS$ then
17:     return $Z_{mi,mj}$
18: else
19:     return $Z_{si,sj}$
20: end if
```

(2) Free zone allocator (FZA). When new data requires a new zone, FZA dynamically allocates the data to a zone with a level number close to the data hotness and therefore balances the wear across zones. Algorithm 1 gives the four main steps of FZA to write data with hotness h to a new zone.

First, we search for an available zone in the *h-level free-zone list* (*Lines 1–3*). If the *h-level free-zone list* has free zones, the minimal-wear zone in the list is allocated to the data. Besides, the hotness of the data is updated to the lifetime value field of the zone, which is usually utilized in many databases to represent the hotness of data on zones. If *the h-level free zone list* has no free zones, FZA looks for a similar wear zone with the wear range of h-level zones.

Secondly, FZA looks for the maximal wear zone ($Z_{mi,mj}$) from the *(h-1)* level to *1* level *free zone lists* (*Lines 4–9*) and then calculates the difference (DM) between the wear of this zone and the h-level wear range (i.e., $EC_{min} + (h-1) * R$—$EC_{min} + h * R$).

Thirdly, FZA looks for the minimal wear zone ($Z_{si,sj}$) from the *(h+1)* level to *n* level *free zone lists* (*Lines 10–15*) and then calculates the difference (DS) between the wear of this zone and the h-level wear range.

Algorithm 2 Cold Data Migration.

Input: (1) n: number of zone levels, (2) i-level used-zone list and J: number of zones in the i-level used-zone list,(3) L_j: lifetime of the data on the j-th zone, (4) V_j: ratio of valid data of the j-th zone

Output: a zone with cold data to migrate

```
/*Search for a zone with the coldest data and the minimal valid-data ratio.*/
1: for $h = n$ to $i + 1$ do
2:     $v \leftarrow 1$.
3:     for $j = 1$ to $J$ do
4:         if $L_j = h$ and $V_j < v$ then
5:             $r \leftarrow j, v \leftarrow V_j$.
6:         end if
7:     end for
8:     if $r$ != null then
9:         return the zone $z_r$
10:    end if
11: end for
```

Finally, depending on the size of the two variables (DM and DS), we decide to assign the data to zone $Z_{si,sj}$ or zone $Z_{mi,mj}$ (*Lines 16–20*). If DM is smaller than DS, it indicates that zone $Z_{mi,mj}$ is closer to the h-level wear range compared to zone $Z_{si,sj}$. And allocate the data to zone $Z_{mi,mj}$. If not, we allocate the data to zone $Z_{si,sj}$.

(3) Cold data migration (CDM). Ideally, FAZ can adaptively balance the wear among zones in ZNS SSDs. However, FAZ needs to be further optimized for real workloads. On the one hand, a small portion of data at an LSM-tree level can be of different hotness compared to other data at the same level. A zone holding the hotter data can be reclaimed quickly. The wear of the zone can be effectively balanced by FAZ when the zone is reallocated. Nevertheless, a zone holding the colder data cannot be reclaimed promptly. FAZ cannot reclaim the zone voluntarily, resulting in an imbalance of the wear on the zone. On the other hand, free zones at a level might be used up when the amount of data at the corresponding LSM-tree level grows dramatically. To make matters worse, some zones at the level may hold cold data, which cannot be reclaimed promptly. Accordingly, the wear imbalance among zones will be further exacerbated. In this paper, we further propose a cold data-migration scheme (CDM) to facilitate wear leveling between zones.

When cold data is written in a zone with less wear, CDM will be executed because the less-worn zone will probably not be reclaimed timely. The core idea is to migrate one of the zones that hold the coldest data at the corresponding level. We explain CDM with an example of allocating an i-level zone to the data with the hotness h. If i is greater than h, it indicates that the i-level free-zone list is empty. Moreover, a less-worn zone may be far from i-level zones. In this case, some less-worn zones that hold cold data remain always not erased. Therefore, we first check whether i-level used zones have cold data (i.e., its hotness is smaller than i). If so, we search for zones that hold the coldest data that have the maximal hotness. We then select one of these zones with the least valid data to migrate.

The details of CDM are shown in Algorithm 2. We first seek these zones that hold the coldest data from hotness n to hotness *(i+1)*. Specially, when the first data is stored in a new zone, the hotness of the data will be recorded as the lifetime of the data on the zone. Thus, the hotness (h) is consistent with the lifetime of data on zones. In this way, the coldest data with the maximal hotness among i-level used zones will be first found out. If the hotness of the coldest data is obtained, two variables (r and v) are utilized to record the zone that holds the coldest data and the minimal valid data ratio (*lines 4–6*). Finally, we migrate the cold data of the zone to a new zone (*lines 8–10*). In general, the cold data migration is only executed when a zone is allocated to the data with a lower level (i.e., i). The core idea is to release an i-level used zone that holds the coldest data and minimal valid data ratio, in order to balance the wear on these less wear zones.

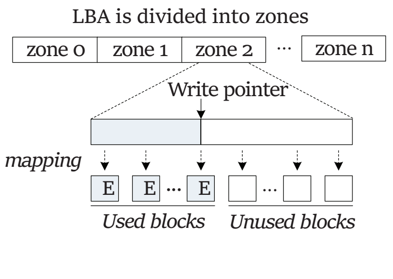
4.2 Intra-Zone Management

The intra-zone management part is implemented to achieve wear leveling across blocks within a zone. The part includes a partial-erase-based zone-reset module to eliminate the unnecessary erase operations on unused blocks and a ware-aware block-allocator module to evenly distribute erasure counts across blocks. The two modules will be explained in detail.

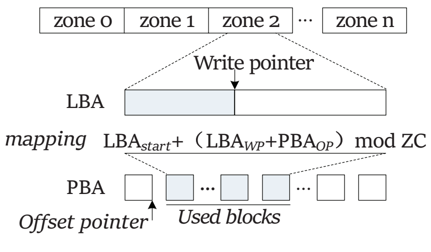
4.2.1 Partial-Erase-Based Zone Reset. The current zone-reset method erases unused blocks when resetting a zone. As discussed in Section 3.2, the average space utilization of all zones is about 70%, which means that nearly 30% of erase operations are performed on unused blocks. Theoretically, we can increase the SSD lifetime by 30% if we manage to eliminate the unnecessary erases. The partial-erase-based zone-reset method is thus proposed to achieve this goal.

Figure 8a shows the basic idea of the partial-erase-based zone reset. ZNS SSDs divide the logical block address (LBA) into zones. Each zone has a sequential segment of LBA and is mapped to multiple physical blocks. Meanwhile, a write pointer records the currently written location in the LBA. Considering these features of a ZNS zone, the partial-erase-based zone reset first calculates the range of the LBA associated with blocks that have been written with reference to the write pointer. Secondly, based on the range of the written LBA, the used physical blocks can be traced in the zone mapping table. Thirdly, a partial erase function is implemented to erase these used blocks.

Although the novel zone-reset method can eliminate unnecessary erase operations, the method may incur an extremely unbalanced distribution of erasure counts across blocks. More precisely, when a zone is reset, its write pointer will be moved to the starting position of the zone LBA. When the zone is re-allocated, new data will be written from the starting position of the LBA. Consequently, most erasure counts will be concentrated on blocks with small LBAs. Therefore, we further propose a wear-aware block allocator to address the issue of the extreme imbalance in erasure count distribution. Otherwise, the storage system cannot fully benefit from the partial-erase-based zone-reset method.



(a) Partial-erase-based zone reset



(b) Wear-aware block allocator

Fig. 8. Zone reset.

4.2.2 Wear-aware Block Allocator. Figure 8b shows the main mechanism of the wear-aware block allocator. The basic idea is to write a new zone from a certain physical block address (PBA) which is the written position at the time of the most recent reset. For write requests on a newly acquired zone, we must associate the starting position of zone LBA with the PBA of the most recently written block in the zone. To realize the functionality, we treat a zone as a circular queue. Namely, we adopt a modulo operation to associate the physical and logical addresses as well as avoid any invalid address exceeding the zone capacity.

The wear-aware block allocator mainly includes four main steps. First, when a zone is reset, an offset pointer records the offset of PBA, i.e., the distance (PBA_{op}) between the starting PBA and the recently written PBA. Secondly, when new data needs to be written on the zone, the PBA for the new data can be calculated by

$$PBA_w = LBA_{start} + (LBA_{wp} + PBA_{op}) \bmod ZC. \tag{3}$$

Note that LBA_{start} and ZC represent the starting LBA and the zone capacity, respectively. Additionally, LBA_{wp} is the distance between the write pointer and the starting LBA. Thirdly, when the data residing in a zone needs to be read, the PBA of the data can be calculated by

$$PBA_r = LBA_{start} + (LBA_r + PBA_{op}) \bmod ZC. \tag{4}$$

Note that LBA_r represents the LBA of the data. Finally, when a reset command is invoked, we update the offset pointer, which can be calculated by

$$PBA_{op'} = (LBA_{wp} - LBA_{start} + PBA_{op}) \bmod ZC. \tag{5}$$

In summary, the wear-aware block allocator utilizes an offset pointer to maintain the currently written PBA at the time of the last reset. For each data request, we recalculate the new PBA by a simple modulo operation. The storage and performance overhead is negligible and has no impact on the overall performance of ZNS SDDs.

5 EXPERIMENTS

In this section, extensive experiments based on typical workloads are conducted to evaluate the effectiveness of the proposed WA-Zone. The experimental setup and benchmark are first presented in Section 5.1. Section 5.2 then analyzes the performance improvement achieved by WA-Zone.

5.1 Experimental Setup

To evaluate the effectiveness of WA-Zone, an NVMe SSD emulator (i.e., *FEMU*) [25], which is a QEMU-based flash emulator, is utilized to implement the target ZNS SDD. Since FEMU is directly exposed to the guest operation system (OS), it can be easily used to emulate different types of SSDs, including the white-box mode [4], the black-box mode (wherein SSDs are equipped with a page-level mapping-based FTL), and the ZNS mode. Therefore, a target ZNS SDD emulated by FEMU can directly expose the NVMe Zone interface to the host, enabling the host to manage data allocation for the target ZNS SSD.

Table 1 shows the related configuration of the target ZNS SSD. First, the version of FEMU is 5.2 and the host Linux kernel version is 5.12. Secondly, the target ZNS SSD is configured with eight channels and each channel holds four chips. Each chip is divided into two dies and eight planes. Meanwhile, each plane includes 256 blocks. Each block includes 128 4-KB pages. Consequently, the capacity of the target ZNS SSD is 32 GB. Thirdly, we further emulate 256 128-MB zones. Fourthly, the host software includes RocksDB 7.7.3 [14] and ZenFS 2.0.1 [10].

The inter-zone management part is integrated into ZenFS, which is divided into three main steps. First, we implement an h-level free-zone list and an h-level used-zone list to classify all zones into h range groups. The free-zone list at each level is sorted according to the erasure counts, improving the efficiency of the lookup operation and update operation during zone allocating and

Table 1. Experimental setup.

Component	Description
FEMU	Version: 5.2, Linux Kernel: 5.12
SSD	Channels: 8, Chips/Channel: 4, Dies/Chip: 2, Planes/Die: 4, Blocks/Plane: 256, Pages/Block: 128, Page Capacity: 4 KB
Flash latency	Read latency: 40us, Program latency: 200us, Erase latency: 2000us
ZNS	Zone size: 128 MB, Number of Zones: 256
Software Version	RocksDB Version: 7.7.3, ZenFS Version: 2.0.1
db_bench	Key Size: 128 B, Value Size: 8192 B, SST File Size: 64 MB, I/O Mode: Direct I/O, Thread: 32

reclaiming. Secondly, we modify the original zone allocation module of ZenFS to implement the new data hotness classification mechanism and wear-aware zone allocator. When new data requires a new zone, the module classifies the corresponding hotness level according to the LSM-tree level associated with the data, and subsequently searches for a corresponding free zone in the free-zone list according to this hotness level. Finally, we implement a new cold data migration module (i.e., CDM) to recycle zones that have been occupied by cold data. When a zone is allocated to data associated with a lower level or zone reclamation is triggered, the CDM module is performed.

The intra-zone management part is implemented in the Zone FTL within the emulated ZNS SSD device. We first replace the original zone reset with a partial-erase-based zone reset to eliminate unnecessary erasures. Then, a new metadata (PBA_{op}) is added to each zone to record the distance between the starting PBA and the recently written PBA after the partial-erase-based zone reset is performed. After that, to realize the wear-aware block allocator, an address relocation module is added to enable the physical blocks in a zone to be used in polling. When a zone write/read operation is received, the address relocation module is triggered to calculate the PBA of data.

Finally, a built-in benchmark tool (i.e., *db_bench*) of RocksDB is used to generate different workloads that are used to evaluate the effectiveness of WA-Zone [6]. Six workloads with 20 million key-value pairs and different access characteristics are utilized, including *fillrandom* (termed *Random*), $fillrandom + overwrite$ (termed *Random+OW*), $fillrandom + updaterandom$ (termed *Random+U*), *fillseq* (termed *Seq*), $fillseq + overwrite$ (termed *Seq+OW*), and $fillseq + updaterandom$ (termed *Seq+U*). The *fillseq* and *fillrandom* workloads write key-value pairs in sequential key order and random key order, respectively. The *updaterandom* workload performs read-modify-write operations for random keys. The *overwrite* workload generates random key-value pairs to overwrite the existing key-value pairs. Moreover, $fillrandom + overwrite$, $fillrandom + updaterandom$, $fillseq + updaterandom$, and $fillseq + overwrite$ represent combinations of workloads with different write modes. In addition, three YCSB workloads with skewed access distributions (i.e., YCSB-A, YCSB-B, and YCSB-F) are selected to evaluate the effectiveness of the proposed techniques.

5.2 Experimental Results and Analyses

In this section, we first analyze the experimental results of the Inter-Zone management (*Inter-Zone*) and the Intra-Zone management (*Intra-Zone*) schemes of WA-Zone, respectively. A comprehensive performance analysis is subsequently given.

(1) Inter-Zone. We first compare *Inter-Zone* with a baseline scheme (termed *Origin-Zone*) which utilizes the original zone-allocation scheme of ZenFS. To evaluate the effectiveness of our proposed scheme, we further implement two comparison schemes, i.e., *Seq-Zone* [16] and *eZNS* [32]. Specially, all zones in ZNS SSDs are allocated sequentially in *Seq-Zone*. *eZNS* preferentially maps these erase blocks with minimal erasure counts to a new zone when it is opened.

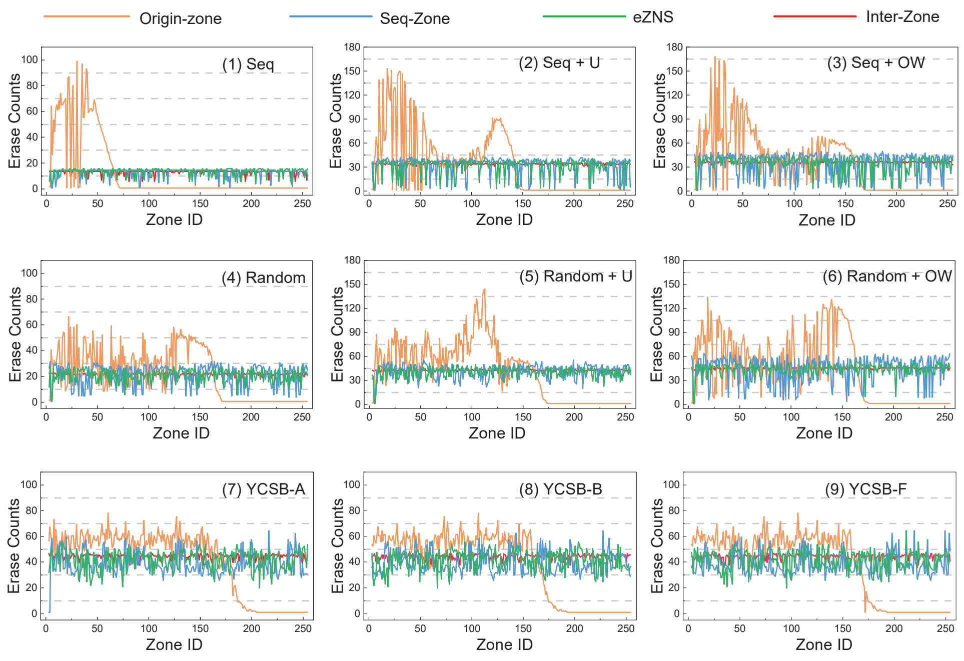


Fig. 9. Distribution of erasure counts across zones in ZNS SSD.

Figure 9 illustrates the distribution of erasure counts across zones in ZNS SSDs using *Origin-Zone*, *Seq-Zone*, *eZNS*, and *Inter-Zone*, respectively. Intuitively, *Seq-Zone* avoids frequent erasures on a small proportion of zones and thus achieves a more balanced distribution of erasure counts across zones than *Origin-Zone*. Compared to *Origin-Zone*, *Seq-Zone* reduces the discrepancy between the maximum and minimum erasure counts among all zones by 53.49%. *eZNS* dynamically maps the youngest free blocks to new zones so that those erased blocks with the least number of erasures are written first. Compared to *Seq-Zone*, *eZNS* is more effective in reducing the discrepancy between the maximum and minimum erasure counts by 19.08%. *Inter-Zone* is equipped with a wear-aware zone allocator which dynamically allocates data with variable hotness to zones with corresponding lifetimes. Compared to *eZNS*, *Inter-Zone* achieves accurate and sophisticated zone allocation that further balances the wear among zones. Furthermore, compared to *eZNS*, *Inter-Zone* shrinks the gap between the maximum and minimum erasure counts by 88.74%. To evaluate the effectiveness of *Inter-zone*, we further compute the standard deviation of the erasure count distribution among the three schemes. Compared to *Origin-Zone*, *Seq-Zone* and *eZNS* reduce the standard deviation by 72.31% and 78.66%, respectively. Moreover, *Inter-zone* obtains a low standard deviation of 1.12 because of a further reduction of 90.16%.

(2) Intra-Zone. In *Intra-Zone*, we propose a partial erase-based zone reset and a wear-aware block allocator. To evaluate the effectiveness of the two schemes of *Intra-Zone*, we further implement two schemes for comparison. The first scheme (termed *PE-Zone*) only includes the partial erase-based zone reset. The second scheme (termed *Intra-Zone*) implements both methods in *Intra-Zone*.

Figure 10 shows the total number of block-erasure counts in *PE-Zone* and *Origin-Zone*. Considering the current zone reset method causes a large number of unnecessary erase operations on unused blocks when a zone is reset, we present a partial erase-based zone reset to avoid these unnecessary erase operations on unused blocks. Compared with *Origin-Zone*, *PE-Zone* can reduce

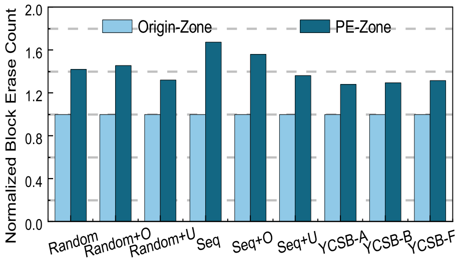


Fig. 10. Total block-erasure counts involved in *Origin-Zone* and *PE-Zone* processing nine different workloads.

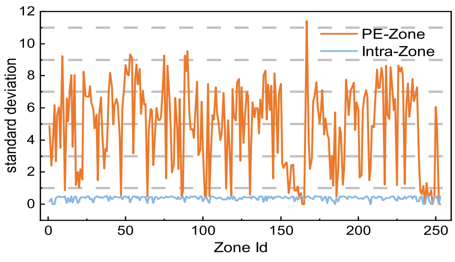


Fig. 11. Standard deviation of the block-erasure counts for each zone involved in *PE-Zone* and *Intra-Zone* processing nine different workloads.

the total number of block erasures by 41.99%, 45.67%, 31.86%, 67.47%, 55.94%, 36.21%, 27.56%, 29.54%, and 31.24%, respectively. On average, 40.83% of the total number of block erasure counts for nine workloads can be avoided. This contributes to a 40.83% reduction in write traffic on ZNS SSDs and lifts the upperbound of the ZNS-SSD lifetime by 40.83%, making space for further improvement.

Figure 11 shows the standard deviation of block-erasure counts for each zone for *PE-Zone* and *Intra-Zone*. *PE-Zone* considerably reduces write traffic on ZNS SSDs. Nonetheless, a zone is written from the first block within it each time the zone is reallocated, involving an unbalanced erasure count distribution between blocks and a shortened ZNS-SSD lifetime. Thus, we present the wear-aware block allocator in *Intra-Zone* to balance the erasure counts on blocks within a zone. We calculate the standard deviation of block erasure counts for each zone to evaluate our proposed schemes. Precisely, the maximal standard deviation obtained by *PE-Zone* is 11.42. On average, the standard deviation obtained by *PE-Zone* is 4.91. Compared to *PE-Zone*, *Intra-Zone* balances the wear between blocks within a zone. Furthermore, the maximum and average standard deviation obtained by *Intra-Zone* is merely 0.51 and 0.38, respectively. Therefore, *Intra-Zone* distributes the erasure operations on a zone evenly to all blocks within the zone.

***(3) Inter-Zone* & *Intra-Zone*.** In this part, we choose seven schemes for comparison, including *Origin-Zone*, *Inter-Zone*, *Inter-Zone + PE-Zone*, *Inter-Zone + Inter-Zone*,*eZNS*, *Seq-Zone* and *Ideal*. *Ideal* represents the write traffic of workloads that are evenly distributed to all blocks in the ZNS SSD. Meanwhile, we choose two metrics, the maximal erasure counts on blocks and the first failure time of blocks, to analyze the effectiveness of our proposed techniques.

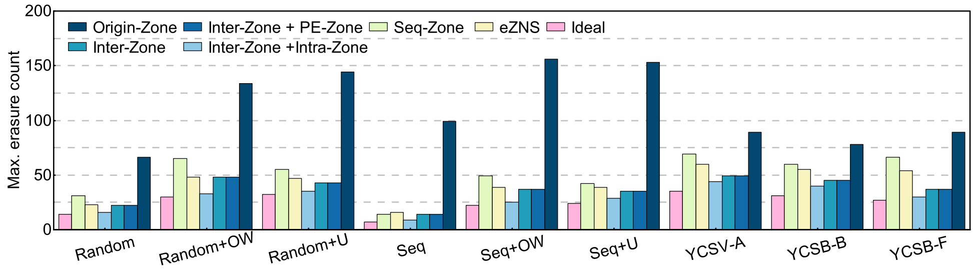


Fig. 12. Maximal erasure counts among blocks involved in five zone-allocation schemes processing nine different workloads.

Figure 12 shows the maximal erasure counts among blocks for nine different workloads. Compared with *Origin-Zone*, *Inter-Zone* reduces 69.37% the maximal erasure counts among blocks.

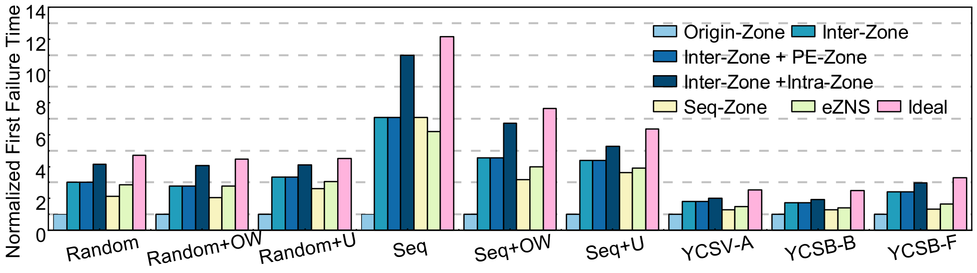


Fig. 13. First block-failure time involved in five zone-allocation schemes processing nine different workloads.

Although *PE-Zone* can avoid 40.48% erasure counts, *Inter-Zone + PE-Zone* does not get a significant reduction of the maximal erasure counts among blocks due to the unbalanced distribution of block erasure counts in a zone. Therefore, a wear-aware block allocator in *Intra-Zone* is designed to achieve the wear leveling on blocks within a zone. Compared with *Inter-Zone*, *Inter-Zone + Intra-Zone* can effectively decrease 23.53% of the maximal erasure counts among blocks, which benefits from both the reduction of write traffic in *PE-Zone* and the wear-leveling on blocks with a zone. Besides, the performance of the *Inter-Zone + Intra-Zone* is close to that of *Ideal*, and the maximal erasure counts among blocks in *Ideal* is about 87.82% that of *Inter-Zone + Intra-Zone*.

Figure 13 shows the first failure time of blocks in ZNS SSD for nine different workloads. In this experiment, we assume that each block has the same maximum number of erasures (*ME*). If the number of erasures on a block exceeds *ME*, the block is worn out. Meanwhile, we assume that the maximal erasure counts among blocks after each workload is executed occurs in the same block of ZNS SSDs. Given these, we can evaluate the first block-failure time (i.e., the maximum number of executions before a block is worn out) for nine different workloads. Compared to *Origin-Zone*, *Inter-Zone* achieves 3×, 2.79×, 3.35×, 7.07×, 4.54×, 4.37×, 2.67×, 1.73×, and 2.4× the first block-failure time, respectively. Combined with the wear-aware allocator, *Inter-Zone + PE-Zone* further enhances the first block-failure time by 40.44% on average for the nine workloads. Summarily, compared with *Origin-Zone*, *Inter-Zone* improves the ZNS-SSD lifetime by 3.54× on average, and *Intra-Zone* achieves a further enhancement of 1.69× in terms of the ZNS-SSD lifetime. Meanwhile, compared to *Seq-Zone* and *eZNS*, *Inter-Zon+Inter-Zone* enhances 40.38% and 31.46% the ZNS-SSD lifetime, respectively.

5.3 Overhead Discussion

The data hotness classification, the wear-aware zone allocator, and the wear-aware block allocator in WA-Zone involve certain overheads in terms of space and time. In this section, we analyze the space overhead and the time overhead of WA-Zone in detail.

5.3.1 The Space Overhead of WA-Zone. The space overhead is primarily caused by an *h*-level free-zone list and an *h*-level used-zone list in the data hotness classification module. Each zone requires a structure containing the number of erasures that needs to be maintained in a linked list. Assume that the structure uses 4 bytes to record the number of erasures and the zone number, respectively. Consequently, a total number of 2^{32} erasure counts and 2^{32} zones can be supported, which is sufficient for an enterprise-grade ZNS SSD. Hence, the linked-list-based zone management only incurs a space overhead of 8 bytes for each zone. The accumulated space overhead is less than 8 KB for a thousand zones, which is acceptable given the large capacity of the host-side main memory where the two linked lists reside.

Additionally, metadata (PBA_{op}) is required to record the distance between the starting PBA and the recently written PBA in each zone. The 4-byte size of the metadata is also sufficient, which can support the recording of the offset of a 4-GB zone. Assume PBA_{op} and its zone number are recorded in DRAM, the total space overhead is also less than 8 KB, which is acceptable. Besides, PBA_{op} can be stored in the reserved space for each zone to avoid DRAM space consumption.

5.3.2 The Time Overhead of WA-Zone. On the host side, an insertion algorithm is employed by the data hotness classification module into the zone list. The algorithm has a time complexity of $O(n)$ because zone lists have already been sorted by erasure count. Meanwhile, the time complexities of FZA and CDM in the wear-aware zone allocator are $O(n)$ and $O(n^2)$, respectively. Summarily, all the aforementioned algorithms are bounded by polynomial time. Moreover, n is considered to be of a moderate size because the number of zones is limited. Typically, these algorithms merely consume sub-nanoseconds each time, running on a general-purpose processor on the host side. On the other hand, the program latency on the SSD side is normally hundreds if not thousands of microseconds. Accordingly, this time overhead is negligible compared with the program latency.

In addition, due to the wear-aware block allocator, each access to the data needs to calculate the PBA of the data based on the corresponding formula. Its time complexity is $O(1)$. Likewise, the process of address relocation can complete this calculation process in sub-nanoseconds times. The only concerns in terms of performance are related to write amplification, bandwidth overhead, and time delay which may be caused by the cold data migration in CDM. Therefore, we further collect write amplification, throughput, and write latency in WA-Zone, as shown below.

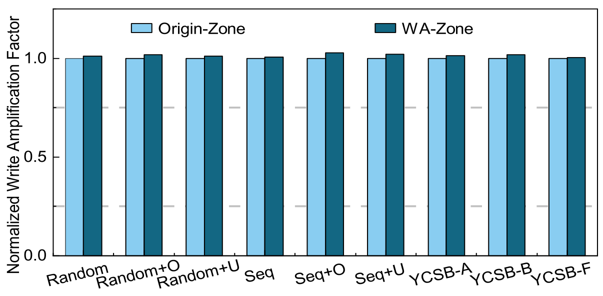


Fig. 14. Write amplification incurred by *Inter-Zone* processing nine different workloads.

Figure 14 shows the write amplification of the *Inter-Zone*. CDM is performed only when a new zone is assigned to data with a hotness level lower than the wear level of the zone. Moreover, to reduce write amplification, a corresponding zone that holds the coldest data and has the minimal valid-data ratio is selected to be recycled. As a result, the *Inter-Zone* introduces a negligible factor of write amplification (1.45% on average) compared to *Origin-Zone*. Meanwhile, compared to *Origin-Zone*, WA-Zone is not affected considerably in terms of throughput and write latency. As shown in Figure 15 and Figure 16, compared to *Origin-Zone*, the throughput is reduced by only 1.42%, and write latency is increased by only 2.5%, which is acceptable.

6 RELATED WORK

Compared to the traditional block-interface SSDs, ZNS SSDs exhibit many favorable features, such as lower DRAM cost and less over-provisioning space. Many related works on ZNS SSDs have been proposed to analyze and optimize the performance of ZNS SSDs, which can be roughly divided into five types, including performance analysis, data allocation, GC optimization, parallelism optimization, and dynamic mapping.

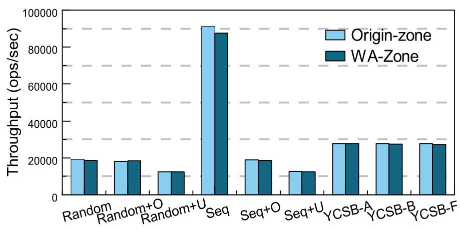


Fig. 15. Throughput involved in Origin-Zone and WA-Zone.

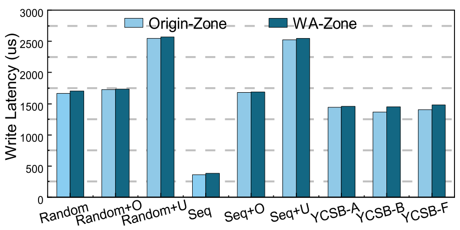


Fig. 16. Write Latency involved in Origin-Zone and WA-Zone.

Performance Analysis. Many related works have been first presented to evaluate and discuss the benefits of ZNS SSDs. Firstly, Stavrinos *et al.* [40] analyze how ZNS SSDs simplify or solve the problems of traditional block-interface SSDs. Second, based on a modified F2FS and RocksDB, Bjørling *et al.* [2, 3] show that a ZNS SSD has a higher throughput and lower tail latency compared to a block-interface SSD. Third, Jin *et al.* [21] discuss the challenges and opportunities of optimizing the traditional index structures for ZNS SSDs. The above work exhibit that ZNS SSDs offer many distinct advantages over traditional block-interface SSDs, including higher performance, lower DRAM costs, and lower GC costs. In this paper, we further analyze the unbalanced distribution of erasure counts across zones and the unnecessary erase operation on unused blocks in ZNS SSDs based on typical workloads of RocksDB. These motivations provide a new idea for the application of the LSM tree-based database in ZNS SSDs.

Data allocation. Since ZNS SSDs transfer the responsibility of data management to the host, many related works are proposed to reduce write amplification by effectively allocating within zones of ZNS SSDs. The sequential write characteristics on the zones are well-suited to the LSM tree, so most of the current work is focused on optimizing the LSM tree on ZNS SSDs. Firstly, [34] arranged zones with consideration of the lifetime and hotness of key-value data. Then, Lee *et al.* [24] proposed to allocate data with overlapping key ranges into the same zone as these data are likely to be merged. Additionally, lifetime-leveling compaction is further designed to avoid the existence of the partially-invalid zones [22]. These works can efficiently reduce the total erase counts on ZNS SSDs but ignore the inter-zone wear-leveling issue in ZNS SSDs. The extremely unbalanced distribution of erasure counts across zones in ZNS SSDs is first exhibited in this paper. Therefore, we designed a new wear-leveling technique, which can be combined with these works to further improve the lifetime of ZNS SSDs.

GC optimization. A segment-based garbage collection is first designed to segregate hot and cold data, thereby reducing the long latency of data migration during GC [9]. Then, [37] propose to accelerate GC operations on ZNS SSDs by parallelizing GC processing of the victim segment. These works demonstrate the necessity of GC operation in ZNS SSDs. Ideally, data with a similar lifetime are allocated into a zone by an effective data allocation technique. Nevertheless, all data in a zone are hard to be simultaneously invalid under a real workload. GC operations are still required to improve the space utilization of ZNS SSDs, especially if the ZNS SSDs space is limited. Therefore, we propose CDM to periodically migrate cold data to free the zone with less wear. Moreover, CDM can be utilized in conjunction with the above GC work to optimize GC performance and wear leveling of ZNS SSDs.

Parallelism optimization. To improve the parallelism of the small-zoned ZNS SSD, many related works are first proposed. Bae *et al.* [1] propose to detect inter-zone interference and design a parallelism-aware schedule technique. Meanwhile, two parallel I/O mechanisms are proposed to distribute I/O data across multiple zones to enhance the external parallelism [20, 32]. These techniques are orthogonal to the proposed techniques in this paper. Similarly, our techniques can collaborate with these works to further improve the performance of ZNS SSDs.

Dynamic mapping. Since the NVMe ZNS specification does not define a mapping scheme for ZNS SSDs, various mapping schemes can be utilized to optimize the performance of ZNS SSDs. Han *et al.* [16] first propose ZNS+, a new LFS-aware interface, to reduce the segment compaction overhead of LFS in dynamic mapping ZNS SSDs. Then, an elastic-zoned namespace interface [32] and SplitZNS [18] utilize different mapping mechanisms to improve the performance of ZNS SSDs according to the access pattern of workloads. However, wear leveling is ignored or only considered on the free physical blocks. Moreover, the data allocation in the host also ignores the inter-zone wear leveling of ZNS SSDs, which might further exacerbate the imbalance in wear between zones. Meanwhile, a series of experiments based on the above work show the distribution of erasure counts across zones is still unbalanced. Therefore, this paper explores the inter-zone wear leveling in the host.

7 CONCLUSION AND FURTHER WORK

This paper first discusses the unbalanced distribution of erasure counts across zones in ZNS SSDs and the unnecessary erase operations on unused blocks incurred by the current zone-reset mechanism. To significantly improve the lifetime of ZNS SSDs, according to the access pattern of LSM-tree and the features of ZNS SSDs, a wear-aware zone management technique (termed WA-Zone) including inter-zone management part (*Inter-zone*) and intra-zone management part (*Intra-zone*) is presented. In *Inter-zone*, a wear-aware zone allocator is proposed to balance the wear across zones by exploiting the access pattern of the LSM-tree. In *Intra-zone*, a partial-erase-based zone-reset method is first presented to avoid the unnecessary erase operations on unused blocks. A wear-aware block allocator is further proposed to balance the wear across blocks in a zone. Finally, we conduct a series of experiments to evaluate the effectiveness of WA-Zone. Experimental results exhibit that *Intra-zone* can enhance the lifetime of ZNS SSDs by 3.54×, compared with the baseline. *Inter-zone* achieves a further lifetime improvement of 1.69×.

In addition, WA-Zone can be easily applied to the static mapping ZNS SSDs. Due to the fixed block-to-zone mapping, the inter-zone management part can be directly implemented in the host-side file system or applications without the need of hardware support. Meanwhile, the intra-zone management part only needs to record the last writing location within a zone which is utilized to append data from a new write when the zone is reused. This simple polling write mechanism can be easily supported by ZNS SSD controllers, which merely involves negligible space and time overheads.

As for further work, we have several research directions to further enhance the performance of WA-Zone. First, the grouping zones method in FZA can be further optimized based on more access characteristics of the workloads, e.g., the amount of data at an LSM-tree level or the skewed access distributions, to enhance FZA's adaptive wear-leveling capabilities. Secondly, parallelism constraints between zones can be further considered in the zone allocator to improve the performance of small zone-based ZNS SSDs. Additionally, more effective device-side wear-leveling techniques and collaborative optimizations for device-side wear leveling and host-side wear leveling can be explored to achieve wear leveling in ZNS SSDs which employ a dynamic-mapping mechanism.

REFERENCES

[1] Hanyeoreum Bae, Jiseon Kim, Miryeong Kwon, and Myoungsoo Jung. 2022. What You Can't Forget: Exploiting Parallelism for Zoned Namespaces. In *Proceedings of 14th ACM Workshop on Hot Topics in Storage and File Systems (Hotstorage'22)*. 79–85.

[2] Matias Bjørling. 2019. From Open-channel SSDs to Zoned Namespaces. In *Proceedings of 2019 Linux Storage and Filesystems Conference (Vault'19)*, Vol. 1.

[3] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R Ganger, and George Amvrosiadis. 2021. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *Proceedings of 2021 USENIX Annual Technical Conference (ATC'21)*. 689–703.

[4] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. 2017. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proceedings of 15th USENIX Conference on File and Storage Technologies (FAST'17)*. 359–374.

[5] Sungjin Byeon, Joseph Ro, Safdar Jamil, Jeong-Uk Kang, and Youngjae Kim. 2023. A Free-Space Adaptive Runtime Zone-Reset Algorithm for Enhanced ZNS Efficiency. In *Proceedings of the 15th ACM/USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'23)*. 109–115.

[6] Zhichao Cao, Siying Dong, Sagar Vemuri, and David HC Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *Proceedings of 18th USENIX Conference on File and Storage Technologies (FAST'20)*. 209–223.

[7] Yunpeng Chai, Yanfeng Chai, Xin Wang, Haocheng Wei, and Yangyang Wang. 2020. Adaptive Lower-level Driven Compaction to Optimize LSM-tree Key-value Stores. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 34, 6 (2020), 2595–2609.

[8] Shuo-Han Chen, Chun-Feng Wu, Ming-Chang Yang, and Yuan-Hao Chang. 2022. A File-Oriented Fast Secure Deletion Strategy for Shingled Magnetic Recording Drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 41, 8 (2022), 2463–2476.

[9] Gunhee Choi, Kwanghee Lee, Myunghoon Oh, Jongmoo Choi, Jhuyeong Jhin, and Yongseok Oh. 2020. A New LSM-style Garbage Collection Scheme for ZNS SSDs. In *Proceedings of 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'20)*. 1–6.

[10] Western Digital. [n. d.]. Zenfs. https://github.com/westerndigitalcorporation/zenfs/. 2022.

[11] Chen Ding, Ting Yao, Hong Jiang, Qiu Cui, Liu Tang, Yiwen Zhang, Jiguang Wan, and Zhihu Tan. 2022. TriangleKV: Reducing Write Stalls and Write Amplification in LSM-Tree Based KV Stores With Triangle Container in NVM. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 33, 12 (2022), 4339–4352.

[12] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Evolution of Development Priorities in Key-value Stores Serving Large-scale Applications: the RocksDB Experience. In *Proceedings of 19th USENIX Conference on File and Storage Technologies (FAST'21)*. 33–49.

[13] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: evolution of development priorities in a key-value store serving large-scale applications. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 1–32.

[14] Facebook. [n. d.]. Rocksdb. https://github.com/facebook/rocksdb/. 2022.

[15] Congming Gao, Liang Shi, Cheng Ji, Yejia Di, Kaijie Wu, Chun Jason Xue, and Edwin Hsing-Mean Sha. 2018. Exploiting Parallelism for Access Conflict Minimization in Flash-Based Solid State Drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 37, 1 (2018), 168–181.

[16] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. 2021. ZNS+: Advanced Zoned Namespace Interface for Supporting In-storage Zone Compaction. In *Proceedings of 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*. 147–162.

[17] Hans Holmberg. 2020. ZenFS, Zones and RocksDB-Who Likes to Take out the Garbage Anyway. https://snia.org/sites/default/files/SDC/2020/074-Holmberg-ZenFS-Zones-and-RocksDB.pdf.

[18] Dong Huang, Dan Feng, Qiankun Liu, Bo Ding, Wei Zhao, Xueliang Wei, and Wei Tong. 2023. SplitZNS: Towards an Efficient LSM-Tree on Zoned Namespace SSDs. *ACM Transactions on Architecture and Code Optimization (TACO)* 20, 3 (2023), 26 pages.

[19] Gui Huang, Xuntao Cheng, Jianying Wang, Yujie Wang, Dengcheng He, Tieying Zhang, Feifei Li, Sheng Wang, Wei Cao, and Qiang Li. 2019. X-Engine: An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing. In *Proceedings of 2019 International Conference on Management of Data (SIGMOD/PODS'19)*. 651–665.

[20] Minwoo Im, Kyungsu Kang, and Heonyoung Yeom. 2022. Accelerating RocksDB for small-zone ZNS SSDs by parallel I/O mechanism. In *Proceedings of 23rd International Middleware Conference Industrial Track (Middleware'22)*. 15–21.

[21] Peiquan Jin, Xiangyu Zhuang, Yongping Luo, and Mingchen Lu. 2021. Exploring Index Structures for Zoned Namespaces SSDs. In *Proceedings of 2021 IEEE International Conference on Big Data (IEEE BigData'21)*. 5919–5922.

[22] Jeeyoon Jung and Dongkun Shin. 2022. Lifetime-leveling LSM-tree Compaction for ZNS SSD. In *Proceedings of 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage'22)*. 100–105.

[23] Siu Jung, Seungjin Lee, Jungwook Han, and Youngjae Kim. 2023. Preemptive Zone Reset Design within Zoned Namespace SSD Firmware. *Electronics* 12 (2023).

[24] Hee-Rock Lee, Chang-Gyu Lee, Seungjin Lee, and Youngjae Kim. 2022. Compaction-Aware Zone Allocation for LSM based Key-Value Store on ZNS SSDs. In *Proceedings of 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage'22)*. 93–99.

[25] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. 2018. The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. In *Proceedings of 16th USENIX Conference on File and Storage Technologies (FAST'18)*. 83–90.

[26] Jianchuan Li, Peiquan Jin, Yuanjin Lin, Ming Zhao, Yi Wang, and Kuankuan Guo. 2021. Elastic and Stable Compaction for LSM-tree: A FaaS-based Approach on Terarkdb. In *Proceedings of 30th ACM International Conference on Information and Knowledge Management (CIKM'21)*. 3906–3915.

[27] Renping Liu, Zhenhua Tan, Linbo Long, Yu Wu, Yujuan Tan, and Duo Liu. 2022. Improving fairness for SSD devices through DRAM over-provisioning cache management. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 33, 10 (2022), 2444–2454.

[28] Renping Liu, Zhenhua Tan, Yan Shen, Linbo Long, and Duo Liu. 2022. Fair-ZNS: Enhancing Fairness in ZNS SSDs through Self-balancing I/O Scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2022).

[29] Weiguang Liu, Jinhua Cui, Tiantian Li, Junwei Liu, and Laurence T Yang. 2022. A Space-Efficient Fair Cache Scheme Based on Machine Learning for NVMe SSDs. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 34, 1 (2022), 383–399.

[30] Linbo Long, Jinpeng Huang, Congming Gao, Duo Liu, Renping Liu, and Yi Jiang. 2023. ADAR: Application-Specific Data Allocation and Reprogramming Optimization for 3D TLC Flash Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 42, 6 (2023), 1824–1837.

[31] Chen Luo and Michael J Carey. 2020. LSM-based Storage Techniques: A Survey. *The VLDB Journal* 29, 1 (2020), 393–418.

[32] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. 2023. eZNS: An Elastic Zoned Namespace for Commodity ZNS SSDs. In *Proceedings of 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)*. 461–477.

[33] Manas Minglani, Jim Diehl, Xiang Cao, Binghze Li, Dongchul Park, David J Lilja, and David HC Du. 2017. Kinetic Action: Performance Analysis of Integrated Key-Value Storage Devices vs. Leveldb Servers. In *Proceedings of 23rd International Conference on Parallel and Distributed Systems (ICPADS'17)*. 501–510.

[34] Gijun Oh, Junseok Yang, and Sungyong Ahn. 2021. Efficient Key-Value Data Placement for ZNS SSD. In *Applied Sciences*. 11842.

[35] Devashish R Purandare, Peter Wilcox, Heiner Litz, and Shel Finkelstein. 2022. Append is Near: Log-based Data Management on ZNS SSDs. In *Proceedings of 12th Annual Conference on Innovative Data Systems Research (CIDR'22)*. 1–10.

[36] Wenjie Qi, Zhipeng Tan, Jicheng Shao, Lihua Yang, and Yang Xiao. 2022. InDeF: An Advanced Defragmenter Supporting Migration Offloading on ZNS SSD. In *Proceedings of 40th International Conference on Computer Design (ICCD'22)*. 307–314.

[37] Dongjoo Seo, Ping-Xiang Chen, Huaicheng Li, Matias Bjørling, and Nikil D. Dutt. 2023. Is Garbage Collection Overhead Gone? Case study of F2FS on ZNS SSDs. In *Proceedings of the 15th ACM/USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'23)*. 102–108.

[38] Liang Shi, Jianhua Li, Chun Jason Xue, Chengmo Yang, and Xuehai Zhou. 2011. ExLRU: A Unified Write Buffer Cache Management for Flash Memory. In *Proceedings of the 11th International Conference on Embedded Software (EMSOFT'21)*. 339–348.

[39] Hojin Shin, Myounghoon Oh, Gunhee Choi, and Jongmoo Choi. 2020. Exploring Performance Characteristics of ZNS SSDs: Observation and Implication. In *Proceedings of 9th Non-Volatile Memory Systems and Applications Symposium (NVMSA'20)*. 1–5.

[40] Theano Stavrinos, Daniel S Berger, Ethan Katz-Bassett, and Wyatt Lloyd. 2021. Don't Be A Blockhead: Zoned Namespaces Make Work on Conventional SSDs Obsolete. In *Proceedings of 13th Workshop on Hot Topics in Operating Systems (HotStorage'21)*. 144–151.

[41] Xuan Sun, Jinghuan Yu, Zimeng Zhou, and Chun Jason Xue. 2020. FPGA-based Compaction Engine for Accelerating LSM-tree Key-Value Stores. In *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE'20)*. 1261–1272.

[42] Chenlei Tang, Jiguang Wan, and Changsheng Xie. 2022. Fencekv: Enabling Efficient Range Query for Key-value Separation. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 33, 12 (2022), 3375–3386.

[43] Nick Tehrany and Animesh Trivedi. 2022. Understanding NVMe Zoned Namespace (ZNS) Flash SSD Storage Devices. *CoRR* abs/2206.01547 (2022).

[44] Peng Wang, Guangyu Sun, Song Jiang, Jian Ouyang, Shiding Lin, Chen Zhang, and Jason Cong. 2014. An Efficient Design and Implementation of LSM-tree based Key-value Store on Open-channel SSD. In *Proceedings of 9th European Conference on Computer Systems (EuroSys'14)*. 1–14.

[45] Xuchao Xie, Liquan Xiao, and David HC Du. 2019. Zonetier: A zone-based Storage Tiering and Caching Co-Design to Integrate SSDs with SMR Drives. *ACM Transactions on Storage (TOS)* 15, 3 (2019), 1–25.

[46] Peng Xu, Nannan Zhao, Jiguang Wan, Wei Liu, Shuning Chen, Yuanhui Zhou, Hadeel Albahar, Hanyang Liu, Liu Tang, and Zhihu Tan. 2022. Building a Fast and Efficient LSM-Tree Store by Integrating Local Storage with Cloud Storage. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 3 (2022), 26 pages.

[47] Ting Yao, Zhihu Tan, Jiguang Wan, Ping Huang, Yiwen Zhang, Changsheng Xie, and Xubin He. 2019. SEALDB: An Efficient LSM-tree based KV Store on SMR Drives with Sets and Dynamic Bands. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 30, 11 (2019), 2595–2607.

[48] Runyu Zhang, Duo Liu, Chaoshu Yang, Xianzhang Chen, Lei Qiao, and Yujuan Tan. 2022. Optimizing CoW-based File Systems on Open-channel SSDs with Persistent Memory. In *Proceedings of 25th Design, Automation and Test in Europe (DATE'22)*. 496–501.