ZnH_2: Augmenting ZNS-based Storage System with Host-Managed Heterogeneous Zones

Yingjia Wang, Lok Yin Chow, Xirui Nie, Yuhong Liang, and Ming-Chang Yang
The Chinese University of Hong Kong
Hong Kong, China

ABSTRACT

Zoned Namespace (ZNS) is an emerging interface that shows great promise for high-density and low-cost cloud environment deployments. Modern high-density SSDs, on the other hand, typically use a hybrid SLC/QLC architecture to mitigate the deficiencies of QLC. Unfortunately, simply integrating ZNS with the mainstream host-transparent hybrid architecture would lead to substantial performance and endurance overhead, defeating the purpose of this architecture in the first place.

In this paper, we examine the possibility of coupling heterogeneous flash management (i.e., SLC and QLC) into the existing zone management at the host. We present ZnH_2, an augmented ZNS-based storage system with host-managed heterogeneous zones. ZnH_2 comprises both SSD firmware and host software designs to unleash the full potential of the hybrid architecture. We build ZnH_2 based on RocksDB, the mainstream application of ZNS, and its filesystem backend ZenFS. Evaluation on YCSB benchmark shows that, compared to the host-transparent counterpart, ZnH_2 achieves up to 28.9% higher load performance and at most 61.1% reduction on the QLC write volume. ZnH_2 also increases the throughput in YCSB macro-benchmarks by 9.5% on average.

ACM Reference Format:
Yingjia Wang, Lok Yin Chow, Xirui Nie, Yuhong Liang, and Ming-Chang Yang. 2024. ZnH_2: Augmenting ZNS-based Storage System with Host-Managed Heterogeneous Zones. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3676536.3676653

1 INTRODUCTION

Flash-based solid-state drives (SSDs) have long held a high share of the storage market due to their notable advantages in performance, energy consumption, and impact resistance. They typically employ the traditional *block interface*, which abstracts the flash storage space into logical blocks that can be read or written randomly. Although this abstraction simplifies host management and software design, SSD requires a complex flash translation layer (FTL) inside to conceal the discrepancy between the block interface and underlying flash characteristics. Nowadays, flash-based SSDs are not ideal and keep evolving, and they are undergoing innovation in both interface and flash memory form.

Regarding the interface, it has been revealed that the block interface has an ever-rising performance and cost tax [12, 13, 22]. *Zoned Namespace (ZNS)* [12] is an emerging interface aiming at the above tax and has gained attention from both academia and industry. ZNS abstracts the flash space into *zones*, which can be read randomly but can only be written *sequentially*. Each zone maintains a write pointer to record the next-written location, and the write request not aligned with the write pointer will be rejected. A zone can only be reclaimed via reset from the host, after which the write pointer is reverted to the start of the zone. The ZNS interface couples host behaviors with the flash characteristics, significantly eliminating the block interface tax (see §2.1). In addition, the SSD still retains sophisticated flash media management, such as wear-leveling and ECC correction, making it easier for the host to standardize and adapt. The ZNS-based storage system (including ZNS SSDs and ZNS-compatible host software) has demonstrated the potential to be deployed in cloud environments as a low-cost solution [11, 45].

In addition to the interface advancements, flash memory technology continues evolving towards denser forms, where multiple bits can be encoded in a flash memory cell [29, 34, 36]. For example, SLC, MLC, TLC, and QLC represent that 1, 2, 3, and 4 bit(s) are encoded in a cell, respectively. Storing more bits per cell can drastically increase capacity and reduce cost but at the trade of severe performance and endurance degradation. It has been characterized that QLC bears 20~60× page program latency than SLC, and the P/E cycles of QLC are only about 1% of those of SLC [36]. This indicates that, even with high capacity and low cost, adopting QLC SSDs faces significant challenges in terms of performance and endurance. To mitigate these challenges, modern high-density SSDs typically employ a *hybrid* architecture [1, 3, 4], which maintains an SLC cache by programming a portion of QLC flash blocks in SLC mode in a *host-transparent* manner. SLC cache not only provides higher performance but also significantly improves the flash endurance. Specifically, hot data with shorter lifetimes often remains in the SLC cache until destage, and only cold data with longer lifetimes will be migrated to QLC blocks when the SLC cache is not sufficient.

Although the hybrid SSD architecture has been extensively explored in the past, none of them discussed how to design the architecture of the hybrid ZNS SSD. It is crucial for ZNS-based storage systems to employ the hybrid architecture for both low-cost merit and better performance/endurance. However, based on our in-depth investigation, the host-transparent approach oriented from the mainstream hybrid SSD exhibits high inefficiencies on ZNS. The major obstacle is that, due to the limited SSD-internal DRAM, ZNS SSD is only capable of maintaining a coarse-grained zone-leveling

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676653

mapping table. Such coarse granularity not only restricts the potential of migrating data in a fine-grained manner but also confines the host capability for cross-layer optimization (see §2.3). Therefore, the host-transparent hybrid ZNS SSD inevitably leads to a high performance and endurance overhead.

In this paper, we explore the idea of shifting heterogeneous flash management (i.e., SLC and QLC) from the device to the host, seamlessly integrating with the existing zone management. To this end, we present ZnH_2, an augmented ZNS-based storage system with Host-managed Heterogeneous zones. ZnH_2 incorporates two major components: (1) HZNS SSD and (2) HZNS-aware host software. We first propose HZNS (H: heterogeneous), an advanced ZNS interface, which extends both command and device attributes for the host to manage heterogeneous zones *explicitly*. To unleash the full potential of HZNS, we incorporate a fine-grained and semantic-guided data migration process and revamp the zone allocation policy in the HZNS-aware host software. To minimize the data transfer overhead, we also offload the migration process to the SSD by leveraging NVMe Simple Copy [35]. We build ZnH_2 based on RocksDB [7] and its filesystem backend ZenFS [9], where RocksDB is the mainstream application of ZNS. Specifically, we implement the HZNS-aware logic in ZenFS, and thus RocksDB (and also the upper layers of RocksDB) can benefit without any code modification. Evaluation on YCSB benchmark tool [14] shows that, compared to the host-transparent counterpart, ZnH_2 can improve the load performance by up to 28.9% and reduce the QLC write volume by at most 61.1%. ZnH_2 also demonstrates the throughput improvement with an average of 9.5% in real YCSB macro-benchmarks. We open-source ZnH_2 for public use on GitHub[1].

Our contributions can be summarized as follows:

- We perform a deep investigation of why the current host-transparent hybrid SSD architecture is not efficient on ZNS (see §2.3).
- We propose ZnH_2, a novel augmented ZNS-based storage system with host-managed heterogeneous zones, featuring both new ZNS interface design and new software management policies (see §3).
- We build ZnH_2 with RocksDB, ZenFS, and an emulated HZNS SSD, and evaluation on YCSB benchmarks is quite encouraging (see §4).

The remainder of this paper is organized as follows. The background and motivation are presented in §2. The design of ZnH_2, including HZNS and HZNS-aware software adaptations, are demonstrated in §3. The evaluation and conclusion are shown in §4 and §5, respectively.

2 BACKGROUND AND MOTIVATION

2.1 Zoned Namespace (ZNS)

To get around the block interface tax, the Zoned Namespace (ZNS) interface was proposed and has drawn broad interest from academia and industry. How ZNS avoids the block interface tax can be summarized in three aspects. First, due to the sequential write constraint of ZNS, the logical and physical addresses in a zone are naturally coupled. Thus, ZNS SSD does not require the fine-grained page-level mapping table, which greatly reduces the equipment of SSD-internal DRAM. Second, ZNS eliminates device-side garbage collection (GC) and shifts GC control upper to the host. Therefore, ZNS SSD needs a much smaller flash over provision (OP) to counteract GC-caused performance degradation. Third, with zone abstraction, the host can holistically optimize data placement, e.g., place data with different lifetimes into different zones, leading to better overall performance and system write amplification.

Nevertheless, the performance and cost advantages of ZNS are built on the necessity for host cooperation. Many host software in different layers has supported ZNS, including device mapper (e.g., dm-zoned [5]), filesystem (e.g., F2FS [23], BtrFS [33]) and application (e.g., RocksDB [12], Ceph [16]). Among them, device mapper offers the highest degree of compatibility as all applications and filesystems can transparently run on ZNS SSDs. On the other hand, achieving compatibility directly in applications demonstrates the most efficiency because there is no semantic isolation on the I/O stack to restrict host-level optimization.

While the ZNS ecosystem is continuously enriched, the ZNS interface itself is also being revisited [17, 18, 30, 31, 37, 38]. To avoid the high data transfer overhead between host and device, ZNS+ [17] enables log-structured filesystems to offload data copy during segment compaction into the device. ZNS+ also supports overwriting a zone to reduce host GCs. eZNS [31] focuses on multi-tenant environments and exposes the relationship between zones and flash parallel units (e.g., die) to the host. In this way, host software can holistically allocate and manage zones to improve overall flash bandwidth usage and reduce inter-tenant I/O interference. oZNS [38] unlocks the strict sequential write constraint of ZNS and proposes an indirection layer inside SSDs for the goal of improving the SSD performance under small-size writes.

2.2 High-Density Flash Memory and Hybrid SSD Architecture

Based on the number of bits a flash cell can store, flash memory is characterized into different types. For example, SLC (single-level cell), MLC (multi-level cell), TLC (triple-level cell), and QLC (quad-level cell) represent that 1, 2, 3, and 4 bit(s) are encoded in a cell, respectively. When each flash cell stores more than one bit, different flash pages exhibit varied performance and endurance patterns. For example, QLC has four types of pages: L (lower), CL (central lower), CU (central upper), and U (upper). The read and write latency of the upper page can reach 2.0× and 6.1× compared to those of the lower page [19].

With distinct merits of different cell types, *hybrid* SSD architecture supporting heterogeneous flash (i.e., SLC and QLC) is commonly adopted in commercial QLC SSDs [26]. Prior works on hybrid SSDs are mainly based on the block interface, and these SSDs internally manage an SLC cache and data migration (from SLC to QLC) in a *host-transparent* manner. The SLC cache consists of a fraction of QLC blocks that store only one bit per cell to achieve SLC-like performance and endurance [2]. Typically, the SLC cache ratio is decreased as the SSD space usage grows. For example, Table 1 shows the SLC cache ratio setting of a commercial SSD Intel 660p [1]. When the SLC cache ratio exceeds the threshold, FTL will

[1] https://github.com/yingjia-wang/ZnH2

Table 1: The SLC cache ratio of Intel 660p SSD under different space usage [1].

Space Usage (%)	0~20	20~30	30~40	40~50	50~60	60~70	70~100
SLC Cache (%)	56	50	40	30	25	20	10

continuously migrate valid data in SLC blocks to QLC blocks until the ratio falls below.

Previous works have extensively studied the different aspects of hybrid SSD architecture. Numerous works propose modeling-based or reinforcement-learning-based techniques to adjust the SLC cache ratio dynamically [20, 36, 39, 41, 43]. In addition, Shi et al. suggest writing data directly to QLC when write traffic is high and mitigating performance degradation during GCs by reverting a certain portion of QLC blocks to SLC [34]. Li et al. propose to identify frequently accessed flash pages and migrate them to the SLC cache [27]. Zhang et al. advocate that the distribution of the SLC cache should fully utilize the internal flash parallelism [44]. There are also works targeting hybrid SSDs based on different interfaces [24, 28]. For example, Lee et al. coordinate SLC and MLC regions in the filesystem layer based upon the open-channel SSDs [24]. For multi-streamed SSDs, Lim et al. separate data and parity into different streams in SSD RAIDs and write them in MLC and SLC, respectively [28].

2.3 Motivation

Different from existing works, this paper makes the first attempt to explore the hybrid SSD architecture based on the emerging ZNS interface. Otherwise, even with a magnified SSD capacity, the pure QLC ZNS SSD would suffer from deficient performance and endurance, making it less practical to deploy. Typically, hybrid SSDs are based on the traditional block interface and thus adopt a host-transparent approach for managing SLC cache and data migration. It indicates that the hybrid SSD based on the ZNS interface could also follow this approach, which can attain full compatibility with the existing ZNS ecosystem and thus speed up the deployment.

However, we examine that this host-transparent approach is highly inefficient for ZNS. For simplicity, we refer to the hybrid ZNS SSD employing this host-transparent approach as *DM Hybrid* (DM: device-managed) in the following of this paper. The key obstacle of DM Hybrid is that ZNS demands limited SSD-internal DRAM for low cost (e.g., 10~20× smaller than the block-interface SSD [10]) and thereby only maintains a coarse-grained zone-level mapping table. Such coarse granularity, unfortunately, restricts the potential of migrating data in a fine-grained manner. Furthermore, the decoupling with existing zone management (which is located at the host) incurs semantic isolation, such as data validity and lifetime, making coarse-grained migration even worse due to the lack of cross-layer optimization.

We use Figure 1(a) to illustrate the above problems by introducing how DM Hybrid manages heterogeneous zones (i.e., SLC and QLC zones) and performs data migration. During data migration, DM Hybrid first (1) selects an SLC zone to reclaim and then (2) migrates all data inside to a QLC zone. In the first step, to select an SLC zone, hybrid ZNS SSD can only refer to some rough statistics of zones (e.g., last accessed time). In the second step, data can only be

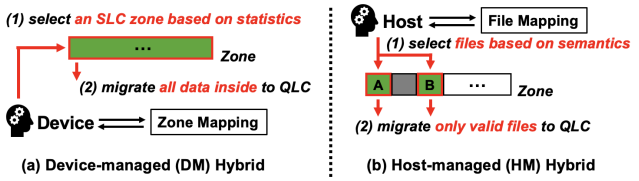

Figure 1: Comparison of device-managed (DM) and host-managed (HM) hybrid ZNS SSDs. The green and gray colors indicate whether the data is valid or not from the perspective of device or host.

migrated based on zone granularity; both valid and invalid data are forced to be migrated due to unknown data validity information (i.e., what data is valid or invalid in a zone). Hence, DM Hybrid with the above two steps inevitably results in high performance and write amplification overhead.

Such overhead becomes much more significant when a huge gap exists between zone capacity and data unit of software, resulting in severe valid/invalid data mixing and lifetime differences. Specifically, the capacity in a zone can reach a few GBs [12, 40], but the software of ZNS SSDs typically employs a small data unit to lessen the reclamation overhead (e.g., in compaction). For example, the default sizes of an SSTable in RocksDB and a segment in F2FS are only 64MB [8] and 2MB [23], respectively. It is theoretically possible to align the data unit with the zone capacity. However, this approach is not desirable in practice because all related data must be read, processed, and finally written during data reclamation, resulting in high memory usage and runtime stalls [21, 32].

Although the host software could deliver additional statistics to assist the ZNS SSD, it is non-trivial and sub-optimal to transfer such multifaceted and dynamically-changed information constantly. In this paper, we explore shifting heterogeneous zone management from the device to the host (i.e., *HM Hybrid* in Figure 1(b), HM: host-managed), which can well resolve the aforementioned issues. Specifically, host software can simply consult the already-maintained file mapping for data semantics (e.g., validity, lifetime), which can be used to assist the data migration process. As shown in Figure 1(b), HM Hybrid can select files for migration based on their semantics. Besides, the host can only migrate valid data and prioritize migrating data with longer lifetimes with negligible additional overhead.

3 ZnH$_2$: AN AUGMENTED ZNS-BASED STORAGE SYSTEM

3.1 System Overview

We propose ZnH$_2$, an augmented ZNS-based storage system with Host-managed Heterogeneous zones. The system overview is shown in Figure 2. ZnH$_2$ comprises two major components: (1) HZNS SSD and (2) HZNS-aware host software (e.g., RocksDB and ZenFS). Specifically, HZNS (H: heterogeneous) is an advanced ZNS interface that extends both command and device attributes for the host to explicitly manage heterogeneous zones. In addition, HZNS-aware host software integrates new policies on data migration and zone allocation, to unleash the full potential of HZNS.

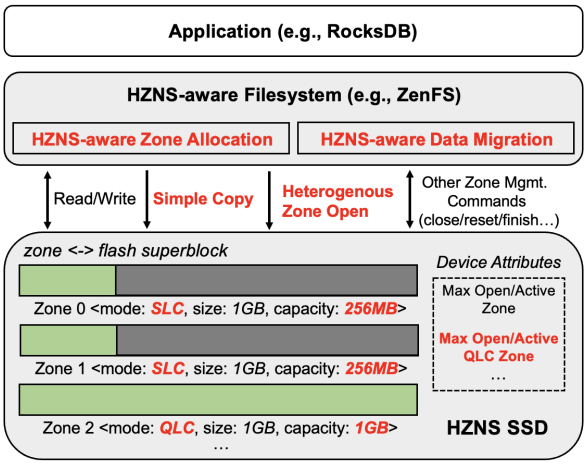

Figure 2: Overview of ZnH_2.

The key rationale of our proposal is to seamlessly couple the heterogeneous zone management with the existing zone management in the host software. By doing so, HZNS-aware host software can leverage the rich host-level semantics and thus accomplish a fine-grained and semantic-guided data migration, leading to increased overall performance and extended SSD lifespan. One noteworthy issue of host-managed data migration, compared to device-managed, is that data requires additional transfers between host and device, which adversely impacts performance and energy efficiency. We resolve this problem by offloading the migration process to the SSD using NVMe *Simple Copy* [35]. Regarding zone allocation, HZNS-aware host software leverages the explicit heterogeneous zone abstraction, which not only supports the host to open heterogeneous zones on demand but also provides a simple portal to reduce double writes (i.e., from host to SLC and subsequently from SLC to QLC).

In the following, we first introduce the detailed design of HZNS in §3.2. We then demonstrate how RocksDB and ZenFS perform explicit heterogeneous zone management as a showcase in §3.3.

3.2 Design of HZNS

We first propose HZNS, an advanced ZNS interface that enables host software to manage heterogeneous zones explicitly. To this end, HZNS is designed to have both command and device attribute extensions, which are fully compatible with the existing ZNS specification.

3.2.1 Heterogeneous zones and flash mapping. Different from ZNS SSDs that have a uniform zone abstraction, HZNS SSD exposes heterogeneous zone modes (e.g., SLC and QLC zones) for host software to use. Therefore, accessing data in different modes of zones could exhibit different performance and endurance characteristics.

In ZNS SSDs, zones are typically one-to-one mapped to a flash superblock, which consists of flash blocks across multiple dies for higher access performance. HZNS SSD also employs this strategy, that is, each SLC or QLC zone is also mapped to an independent flash superblock. However, even though SLC and QLC zones have the same zone size, due to different numbers of bits stored per cell, the capacity (i.e., available space for host software to use) of the SLC zone is only 1/4 of that of the QLC zone. Assume a flash superblock is 1GB, the capacity of the SLC zone and the QLC zone is 256MB and 1GB, respectively.

3.2.2 Zone command extension. HZNS supports *heterogeneous zone open* command that extends the existing zone open command to support zone mode indication. The new command uses a reserved bit of the zone management send command (e.g., bit 09 in command dword 13 [6]). Except for this, other commands (e.g., read, write, zone close, zone reset, and zone finish) remain unchanged from those in ZNS SSDs and are not necessary to extend.

It is worth noting that ZNS SSDs support opening zones in either explicit or implicit ways. Particularly, *implicit open* can be automatically activated when trying to write an empty zone. For HZNS, *implicit open* is only applied to SLC zones, and *explicit open* must be invoked when opening a zone in QLC.

3.2.3 Device attribute extension. Due to limited SSD-internal resources (e.g., DRAM), ZNS restricts the maximum number of zones in open and active states by two device attributes: *max open zone* and *max active zone*. Here, an active zone can be either open or closed. Only open zones can accommodate write I/Os, while both open and closed zones can serve read I/Os. Currently, ZNS SSD products do not clearly differentiate them. For example, the Western Digital ZN540, the commonly used TLC ZNS SSD in research, has both max open/active zone values of 14 [12].

HZNS specifies two other device attributes: *max open QLC zone* and *max active QLC zone* to restrict the maximum number of QLC zones in open and active states. The reason for distinguishing QLC is that an open/active QLC zone requires much higher memory resources than the SLC counterpart due to more bits in a cell and more complex flash programming [15, 42]. In evaluation, we reasonably configure *max open/active zone* and *max open/active QLC zone* to 14 and 9, respectively.

3.2.4 Discussions. HZNS extends the zone abstraction to expose heterogeneous zone management to the host, and does not particularly affect the techniques still managed inside the ZNS SSD (e.g., wear leveling, error correction).

With respect to wear leveling, although the host can now determine the zone mode (i.e., SLC or QLC), wear leveling is still transparently handled by the SSD. For example, HZNS SSD can still periodically swap a portion of flash blocks in SLC and QLC to balance the overall flash lifetime [36]. As for error correction, flash blocks in host-managed SLC and QLC zones still adopt respective policies since high-density flash is more error-prone and thus requires stronger correction codes [19].

3.3 Explicit Heterogeneous Zone Management: A Showcase on RocksDB and ZenFS

On top of the HZNS SSD, we choose RocksDB [7], the mainstream application of ZNS, and its filesystem plugin ZenFS [9] as a showcase. RocksDB is a persistent key-value store from Facebook and serves as the storage engine of many large-scale storage services, websites, and upper-layer software. ZenFS provides end-to-end

compatibility for ZNS and stores RocksDB files in zones. We implement the HZNS-aware policies in ZenFS so that RocksDB (and also the upper layers of RocksDB) can benefit from HZNS without any code modification. For simplicity, in the following of this paper, HZNS-aware ZenFS is abbreviated as *H-ZenFS*.

To manage heterogeneous zones explicitly and efficiently, H-ZenFS introduces a new data migration process and revamps the existing zone allocation process.

3.3.1 Data migration. H-ZenFS initiates a background thread to periodically check the SLC cache ratio (5 seconds by default) and perform data migration when the SLC cache fills up. In the following, we first introduce three principles of how H-ZenFS performs data migration. Then, we take them together and illustrate the whole process of data migration.

The objective of the three principles is to take advantage of the rich host-level semantics (i.e.,#1 and #2) as well as address a key shortcoming of host-managed data migration (i.e.,#3). They are also applicable to other ZNS-compatible host software.

#1: fine-granularity-based
#2: data-lifetime-guided
#3: simple-copy-optimized

Principle #1 points out the importance of fine-grained data migration because a small migration granularity could reduce data lifetime differences and thus optimize the effect of hot/cold data separation. H-ZenFS migrates data based on *extent*, the smallest data management unit in ZenFS. Each file consists of one or more extents, each of which has contiguous logical address space and does not span across different zones. When data migration is triggered, the information of extents can be simply extracted from the latest filesystem snapshot[2] with trivial additional overhead.

Principle #2 advocates prioritizing migrating data with a larger lifetime when the SLC cache is not sufficient. Since the migration is based on extent, H-ZenFS tags each extent in the snapshot with an additional field: the lifetime of its corresponding file. When data migration is triggered, H-ZenFS extracts and sorts the extents based on their lifetime, and then prefers to migrate those with larger lifetimes. If extents have the same lifetime, H-ZenFS prefers the ones with the higher zone reclamation ratio. Here, the zone reclamation ratio is defined as the invalid data capacity divided by the total zone capacity. H-ZenFS prefers such extents because the zones with higher reclamation ratios are considered to be fully-invalidated and reused in the near future.

Principle #3 emphasizes the utilization of NVMe *Simple Copy* [35], a new NVMe command that has been supported in Linux kernel to reduce the overhead from host-managed data migration. The process of Simple Copy is shown in Figure 3. Simple Copy migrates data from multiple source LBA ranges to the destination LBA. Here, the LBA range includes the source LBA and the data length in logical pages. The primary drawback of host-managed data migration is that, despite host software knowing both the source and destination addresses of data for migration, data must undergo transfers (1) from device to host and (2) from host to device. This would induce a large number of read/write operations, consuming more host resources and energy and adversely affecting the system performance. By leveraging Simple Copy, H-ZenFS can offload data migration to the SSD and avoid the above problems.

[2]The snapshot in ZenFS stores the filesystem superblock information, the mapping of files to extents, and the mapping of extents to zones for recovery purposes.

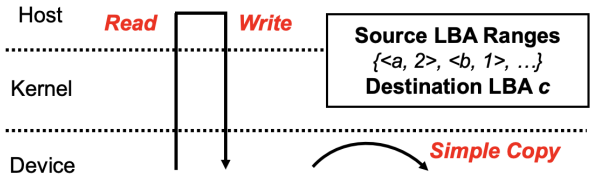

Figure 3: Comparison between existing host-side and simple-copy-based data migration. The host should indicate source LBA ranges and destination LBA in the Simple Copy command.

The whole process of data migration is described as follows. Upon migration, H-ZenFS reads the latest filesystem snapshot, selects valid extents based on their lifetimes and zone reclamation ratios, and migrates them to the QLC zones. After the selected extents are successfully rewritten, H-ZenFS persists the new locations and also updates the extent-to-zone mapping in the filesystem. Therefore, the data read is only forwarded to the new location after the mapping is updated. Since the persistence of the mapping is performed in the background, the performance impact on RocksDB is minimal. Besides, the metadata volume is also trivial compared to the total data volume.

3.3.2 Zone allocation. The baseline ZenFS attempts to co-locate files with similar lifetimes in the same zone. For a file to be written, ZenFS tries to allocate a zone with a larger but closest lifetime with the file lifetime, to avoid further lengthening the lifetime of the zone. Here, the lifetime of a zone is defined as the maximum lifetime of all files stored. If not found, ZenFS tries to open a new zone, which is limited by the value of *max open/active zone*. If *max open/active zone* reaches the upper limit, ZenFS finishes some zones to release resources, after which these zones become full and can not receive upcoming writes.

H-ZenFS shares most of the strategies in zone allocation compared to the baseline ZenFS. The major difference is that, when a new zone needs to be opened, H-ZenFS explicitly opens a QLC zone when (1) the SLC cache is not sufficient and (2) the checking on *max open/active QLC zone* is passed. Otherwise, H-ZenFS will still open an SLC zone. When all zones are configured in SLC, H-ZenFS can only use 25% of the SSD capacity. By opening more QLC zones, H-ZenFS can gracefully gain more capacity to use as the stored data volume increases (see §4.2).

With the explicit SLC and QLC zone abstraction, a key feature of H-ZenFS is that large-lifetime data can be directly written to the QLC zones. Thus, H-ZenFS has the merit of avoiding flash bandwidth contention caused by double writes, especially when the SSD capacity usage is high. The importance of this feature has already been noticed in a previous work [34]. However, this is not a trivial task in block-interface hybrid SSDs, which may have to monitor the data traffic to determine whether the data is written in SLC or QLC [34].

Table 2: Configurations of FEMU HZNS/ZNS SSD. The SLC and QLC latency setups refer to [43] and [19], respectively.

SSD Parameter	Value
# Flash Channels/Dies	8/64
Flash Page/Block/Superblock Size	16KB/16MB/1GB
SSD Capacity	128GB
Zone Size	1GB
SLC/QLC Zone Capacity	256MB/1GB
Max Open/Active Zone	14/14
Max Open/Active QLC Zone	9/9
SLC Read/Write/Erase Latency	30μs/160μs/3ms
QLC Read Latency (L/CL/CU/U)	48μs/64μs/80μs/96μs
QLC Write Latency (L/CL/CU/U)	850μs/2.3ms/3.7ms/5.2ms
QLC Erase Latency	3.5ms

4 EVALUATION

4.1 Experimental Setups

We implement both HZNS and ZNS in a popular NVMe SSD emulator FEMU [25], which runs as a virtual machine with 32GB memory, 16 cores, and a Linux kernel of version 5.10.9. The virtual machine runs on a physical server equipped with four 24-core/48-thread Intel Xeon Platinum 8260 2.40 GHz CPU sockets. The SSD configurations are summarized in Table 2, and SSDs with heterogeneous zones (i.e., device-managed hybrid ZNS SSD and HZNS SSD, see below) adopt the same SLC cache ratio setting from Intel 660p (see Table 1) for a fair comparison.

We evaluate the following three schemes:

- **QLC** means that the hybrid architecture is not applied, and all zones are programmed in QLC. That is, the vanilla RocksDB and ZenFS are running on the pure QLC ZNS SSD.
- **DM Hybrid** means that the device is responsible for managing heterogeneous zones in a host-transparent fashion. That is, the vanilla RocksDB and ZenFS are running on the device-managed hybrid ZNS SSD. The detailed methodology of DM Hybrid is introduced in §2.3.
- **HM Hybrid** is our proposed scheme where the host manages heterogeneous zones explicitly. This scheme corresponds to ZnH_2 (i.e., the RocksDB and HZNS-aware ZenFS are running on the HZNS SSD).

We employ all the workloads in YCSB benchmark [14] for evaluation with 1KB key-value pairs. The versions of RocksDB and ZenFS are v7.9.2 and v2.1.2, respectively. The parameters in RocksDB are configured to the default values, except for the SSTable size in the sensitivity study (see §4.5). We use direct I/O to bypass the effect of the kernel page cache.

4.2 Runtime Capacity and Usage of HZNS SSD

From the perspective of the host, HM Hybrid displays a unique capacity-variant pattern due to the explicit heterogeneous zone management. If the stored data volume continuously increases, HM Hybrid can gracefully gain more available SSD capacity by opening more QLC zones. To clearly visualize this feature, in Figure 4, we depict the runtime statistics of (1) SSD available capacity, (2) total used capacity (both SLC and QLC), and (3) SLC used capacity. The number of key-value pairs in YCSB Load is 30M.

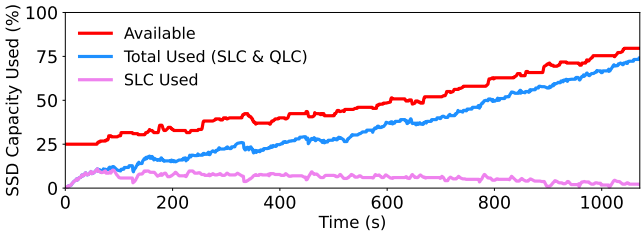

Figure 4: Runtime capacity and usage of HZNS SSD in YCSB Load.

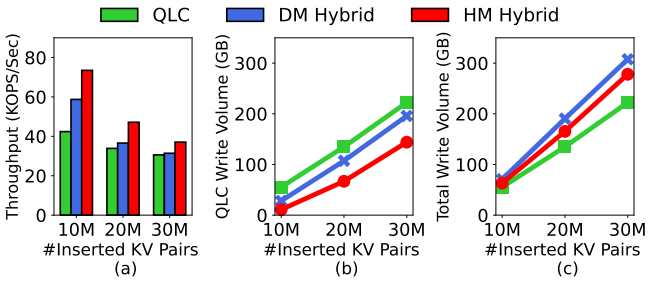

Figure 5: Throughput and write volume in YCSB Load with different numbers of inserted key-value pairs.

We can see that, in the beginning, only 25% of the SSD capacity is usable because HM Hybrid configures all zones in SLC by default. After around 60 seconds, the SLC cache fills up for the first time, and HM Hybrid starts to allocate QLC zones and migrate cold data to them. As the amount of stored data grows, the proportion of the SLC cache decreases and more data has to be stored in QLC. After loading 30M key-value pairs, the SSD capacity and usage are as large as 79.6% and 74.0%, respectively; and at this time, most of the data (i.e., 97.3%) is stored in QLC.

4.3 Performance in YCSB Load Benchmark

We first investigate the advantages of HM Hybrid in terms of load performance and SSD lifetime. Specifically, the impact on the SSD lifetime is indicated by the QLC write volume and the total write volume. Figure 5 demonstrates the throughput and write volume in YCSB Load with 10M, 20M, and 30M inserted key-value pairs.

From Figure 5(a), we first observe that HM Hybrid outperforms DM Hybrid and QLC in terms of throughput by 18.0%~28.9% and 21.2%~73.3%, respectively. Compared to DM Hybrid, the performance gain mainly comes from the effective data migration, which leads to a notable decrease in the QLC write volume. From Figure 5(b), HM Hybrid reduces the QLC write volume by 26.5%~61.1% and 35.2%~80.3%, respectively, compared to DM Hybrid and QLC. It can be seen from Figure 5(c) that the total write volume of HM Hybrid exceeds that of QLC due to the existence of SLC cache. However, since the endurance of SLC is significantly better than QLC (e.g., 100× [36]), this is not a big concern.

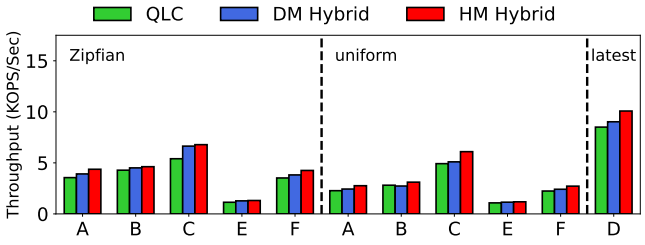

Figure 6: Throughput in YCSB A-F workloads with different access patterns. A: 50% reads and 50% updates. B: 95% reads and 5% updates. C: 100% reads. D: 95% latest reads and 5% writes. E: 95% scans and 5% writes. F: 50% reads and 50% read-modify-writes.

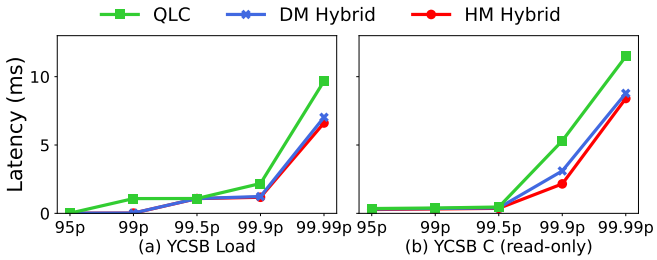

Figure 7: Tail latency in different percentiles in YCSB Load and YCSB C (read-only).

The above results indicate that, under different write loads, by enabling host software to manage heterogeneous zones explicitly, HM Hybrid not only achieves higher write throughput but also greatly lessens the endurance overhead. The merit of better endurance is highly desirable in high-density flash deployment.

4.4 Performance in YCSB Macro-benchmarks

We also compare HM Hybrid with the other two schemes in real YCSB macro-benchmarks with both skewed and uniform patterns. Figure 6 demonstrates the throughput in YCSB macro-benchmarks, and the characteristics of these benchmarks are introduced in the caption of Figure 6. For each workload, we perform 5M operations after loading 20M key-value pairs.

HM Hybrid exhibits an average of 9.5% and 18% higher throughput than DM Hybrid and QLC, respectively. The SLC cache is the cause of the performance gain over QLC. In contrast, the performance improvement over DM Hybrid comes from the fine-grained and semantic-guided data migration. HM Hybrid performs a more accurate hot/cold data separation, so that more hot data can reside in the SLC cache.

4.5 Extended Study

Tail latency in YCSB Load and YCSB C (read-only). For all three schemes, we explore the tail latency in different percentiles. Figure 7 demonstrates the tail latency in YCSB Load and YCSB C (read-only). The workload settings are the same as §4.4.

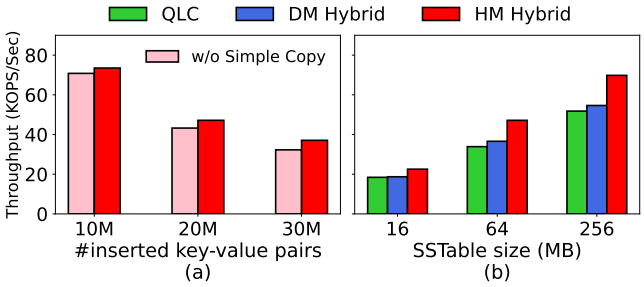

Figure 8: Throughput in YCSB Load (a) with and without Simple Copy and (b) with different SSTable sizes.

In both workloads, HM Hybrid can significantly reduce tail latency compared to QLC but a small amount compared to DM Hybrid. Compared to QLC in 99.9p, HM Hybrid shows 46.1% and 59.4% shorter latency in YCSB Load and YCSB C, respectively, while those are 31.7% and 26.8% in 99.99p. Compared to DM Hybrid in 99.9p, HM Hybrid has 4.6% and 30.6% shorter latency in YCSB Load and YCSB C, respectively. The values in 99.99p are 6.0% and 4.3%.

Performance w/ and w/o Simple Copy. We explore the performance impact of Simple Copy in HM Hybrid. Figure 8(a) demonstrates the throughput in YCSB Load with and without Simple Copy. When Simple Copy is disabled, the throughput of HM Hybrid is decreased by 3.8%, 9.0%, and 14.9% when inserting 10M, 20M, and 30M key-value pairs, respectively. Without Simple Copy, HM Hybrid needs to constantly read and write files, leading to the proliferation of I/O operations and thus reducing the migration efficiency.

Performance under different SSTable sizes. We conduct a sensitivity study on the performance improvement of HM Hybrid under different SSTable sizes. Figure 8(b) demonstrates the throughput in YCSB Load with 20M key-value pairs inserted.

HM Hybrid can achieve higher throughput compared to DM Hybrid and QLC with an average of 25.7% and 32.1%, respectively. The results show that HM Hybrid exhibits better load performance across a wide range of SSTable size configurations.

5 CONCLUSION

This paper makes the first attempt to examine the architecture of the hybrid SLC/QLC ZNS SSD. Based on our in-depth investigation, we present ZnH$_2$, an augmented ZNS-based storage system with host-managed heterogeneous zone management, incorporating both SSD firmware and host software designs. Our showcase based on RocksDB and ZenFS well proofs the benefits of ZnH$_2$ in both performance and SSD lifespan. We believe that host-managed heterogeneous zones can bring more new opportunities and challenges to the storage system, and we hope that ZnH$_2$ can be the cornerstone of future research.

6 ACKNOWLEDGEMENTS

We thank all the reviewers for their constructive feedback. This work is supported by The Research Grants Council of Hong Kong SAR (Project No. CUHK14218522). Ming-Chang Yang is the corresponding author.

REFERENCES

[1] 2018. The Intel SSD 660p SSD Review : QLC NAND Arrives For Consumer SSDs. https://www.anandtech.com/show/13078/the-intel-ssd-660p-ssd-review-qlc-nand-arrives.

[2] 2019. How pseudo-SLC mode can make 3D NAND flash more reliable. https://www.hyperstone.com/zh/blog/How-pseudo-SLC-mode-can-make-3D-NAND-flash-more-reliable-2626.html.

[3] 2021. Enmotus FuzeDrive P200 M.2 NVMe SSD Review: AI Storage Beats SSD Caching. https://www.tomshardware.com/reviews/enmotus-fuzedrive-p200-m2-nvme-ssd-review/2.

[4] 2021. Intel SSD 670p M.2 NVMe SSD Review: Scaling QLC to New Heights. https://www.tomshardware.com/reviews/intel-ssd-670p-m-2-nvme-ssd-review/2.

[5] 2024. dm-zoned. https://docs.kernel.org/admin-guide/device-mapper/dm-zoned.html.

[6] 2024. NVMe Zoned Namespaces (ZNS) Command Set Specification. https://nvmexpress.org/specification/nvme-zoned-namespaces-zns-command-set-specification/.

[7] 2024. RocksDB: A persistent key-value store for fast storage environments. http://rocksdb.org/.

[8] 2024. Setup Options and Basic Tuning in RocksDB. https://github.com/facebook/rocksdb/wiki/Setup-Options-and-Basic-Tuning/.

[9] 2024. ZenFS: RocksDB Storage Backend for ZNS SSDs and SMR HDDs. https://github.com/westerndigitalcorporation/zenfs.

[10] Hanyeoreum Bae, Jiseon Kim, Miryeong Kwon, and Myoungsoo Jung. 2022. What you can't forget: exploiting parallelism for zoned namespaces. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*. 79–85.

[11] Matias Bjørling. 2023. ZNS SSDs: Achieving Large-Scale Deployment. In *Flash memory summit*.

[12] 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 *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 689–703.

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

[14] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.

[15] Hongyang Dang, Xiangyu Yao, Zheng Wan, and Qiao Li. 2023. A Study of Invalid Programming in 3D QLC NAND Flash Memories. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*. 73–79.

[16] Jin Yong Ha and Heon Young Yeom. 2023. zCeph: Achieving High Performance On Storage System Using Small Zoned ZNS SSD. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. 1342–1351.

[17] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. 2021. ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 147–162.

[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* 20, 3 (2023), 1–26.

[19] Shehbaz Jaffer, Kaveh Mahdaviani, and Bianca Schroeder. 2022. Improving the Reliability of Next Generation {SSDs} using {WOM-v} Codes. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. 117–132.

[20] Xavier Jimenez, David Novo, and Paolo Ienne. 2012. Software controlled cell bit-density to improve NAND flash lifetime. In *Proceedings of the 49th Annual Design Automation Conference*. 229–234.

[21] Jeeyoon Jung and Dongkun Shin. 2022. Lifetime-leveling LSM-tree compaction for ZNS SSD. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*. 100–105.

[22] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-streamed {Solid-State} Drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*.

[23] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. 2015. {F2FS}: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. 273–286.

[24] Sungjin Lee, Keonsoo Ha, Kangwon Zhang, Jihong Kim, and Junghwan Kim. 2009. FlexFS: A Flexible Flash File System for MLC NAND Flash Memory.. In *USENIX annual technical conference*. 1–14.

[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 *16th USENIX Conference on File and Storage Technologies (FAST 18)*. 83–90.

[26] Qiao Li, Hongyang Dang, Zheng Wan, Congming Gao, Min Ye, Jie Zhang, Tei-Wei Kuo, and Chun Jason Xue. 2024. Midas Touch: Invalid-Data Assisted Reliability and Performance Boost for 3d High-Density Flash. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 657–670.

[27] Shicheng Li, Longfei Luo, Yina Lv, and Liang Shi. 2022. Latency aware page migration for read performance optimization on hybrid SSDs. In *2022 IEEE 11th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 33–38.

[28] Yoohyuk Lim, Jaemin Lee, Cassiano Campes, and Euiseong Seo. 2017. {Parity-Stream} Separation and {SLC/MLC} Convertible Programming for Life Span and Performance Improvement of {SSD}{RAIDs}. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*.

[29] Chun-Yi Liu, Jagadish Kotra, Myoungsoo Jung, and Mahmut Kandemir. 2018. {PEN}: Design and Evaluation of {Partial-Erase} for 3D {NAND-Based} High Density {SSDs}. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. 67–82.

[30] Umesh Maheshwari. 2021. From blocks to rocks: A natural extension of zoned namespaces. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*. 21–27.

[31] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. 2023. {eZNS}: An elastic zoned namespace for commodity {ZNS}{SSDs}. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 461–477.

[32] Rekha Pitchumani, James Hughes, and Ethan L Miller. 2015. SMRDB: key-value data store for shingled magnetic recording disks. In *Proceedings of the 8th ACM International Systems and Storage Conference*. 1–11.

[33] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. BTRFS: The Linux B-tree filesystem. *ACM Transactions on Storage (TOS)* 9, 3 (2013), 1–32.

[34] Liang Shi, Longfei Luo, Yina Lv, Shicheng Li, Changlong Li, and Edwin Hsing-Mean Sha. 2021. Understanding and optimizing hybrid ssd with high-density and low-cost flash memory. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 236–243.

[35] Selvakumar Somasundaram. 2021. Towards Copy-Offload in Linux NVMe. In *Storage Developer Conference (SDC)*.

[36] Radu Stoica, Roman Pletka, Nikolas Ioannou, Nikolaos Papandreou, Sasa Tomic, and Haris Pozidis. 2019. Understanding the design trade-offs of hybrid flash controllers. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 152–164.

[37] Yu Wang, You Zhou, Zhonghai Lu, Xiaoyi Zhang, Kun Wang, Feng Zhu, Shu Li, Changsheng Xie, and Fei Wu. 2023. FlexZNS: Building High-Performance ZNS SSDs with Size-Flexible and Parity-Protected Zones. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 291–299.

[38] Yingjia Wang, You Zhou, Fei Wu, Jie Zhang, and Ming-Chang Yang. 2024. Land of Oz: Resolving Orderless Writes in Zoned Namespace SSDs. *IEEE Trans. Comput.* (2024).

[39] Qian Wei, Yi Li, Zhiping Jia, Mengying Zhao, Zhaoyan Shen, and Bingzhe Li. 2023. Reinforcement Learning-Assisted Management for Convertible SSDs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[40] Denghui Wu, Biyong Liu, Wei Zhao, and Wei Tong. 2022. ZNSKV: Reducing Data Migration in LSMT-Based KV Stores on ZNS SSDs. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 411–414.

[41] Ming-Chang Yang, Yuan-Hao Chang, Chei-Wei Tsao, and Chung-Yu Liu. 2016. Utilization-aware self-tuning design for TLC flash storage devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 10 (2016), 3132–3144.

[42] Vic Ye. 2018. A Graphical Journey into 3D NAND Program Operations. In *Flash memory summit*.

[43] Sangjin Yoo and Dongkun Shin. 2020. Reinforcement {Learning-Based}{SLC} Cache Technique for Enhancing {SSD} Write Performance. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*.

[44] Wenhui Zhang, Qiang Cao, Hong Jiang, Jie Yao, Yuanyuan Dong, and Puyuan Yang. 2019. SPA-SSD: Exploit heterogeneity and parallelism of 3D SLC-TLC hybrid SSD to improve write performance. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 613–621.

[45] Yanbo Zhou, Erci Xu, Li Zhang, Kapil Karkra, Mariusz Barczak, Wayne Gao, Wojciech Malikowski, Mateusz Kozlowski, Łukasz Łasek, Ruiming Lu, et al. 2024. CSAL: the Next-Gen Local Disks for the Cloud. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 608–623.