Exploring I/O Management Performance in ZNS with ConfZNS++

Krijn Doekemeijer
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Dennis Maisenbacher
Western Digital
Copenhagen, Denmark

Zebin Ren
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Nick Tehrany*
BlueOne Business Software LLC
Beverly Hills, California, USA

Matias Bjørling
Western Digital
Copenhagen, Denmark

Animesh Trivedi*
IBM Research Europe
Zurich, Switzerland

ABSTRACT

Flash-based storage is known to suffer from performance unpredictability due to interference between host-issued I/O and device-side I/O management. SSDs with data placement capabilities, such as Zoned Namespaces (ZNS) and Flexible Data Placement (FDP), expose selective device-side I/O management operations to the host to provide predictable performance. In this paper, we demonstrate that these host-issued I/O management operations lead to performance interference with host-issued I/O. Indeed, we find that the I/O management operations introduced by ZNS and FDP create I/O interference, leading to significant performance losses. Despite the performance implications, we observe that ZNS research frequently uses emulators (over 20 recently published papers), but no emulator currently has function-realistic models for I/O management. To address this gap, we identify ten ZNS I/O management designs, explain how they interfere with I/O, and introduce ConfZNS++, a function-realistic emulator with native I/O management support, providing future research with the capability to explore these designs. Additionally, we introduce two actionable host-managed solutions to reduce ZNS management interference: ZINC, an I/O scheduler prioritizing I/O over I/O management, and the `softfinish` operation, a host-managed implementation of the `finish` operation. In our experiments, ZINC reduces `reset` interference by 56.9%, and `softfinish` reduces `finish` interference by 50.7%.

*Work done while the author was at the Vrije Universiteit Amsterdam.

SYSTOR '24, September 23–24, 2024, Virtual, Israel
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1181-7/24/09.
https://doi.org/10.1145/3688351.3689160

Table 1: *Performance models supported in ZNS emulators; models with "—" are incomplete.*

Model	*FEMU*	*NVMeVirt*	*ConfZNS*	*ConfZNS++*
`Read` *and* `write`	✗	✓	✓	✓
`Reset`	✗	—	—	✓
`Finish`	✗	✗	✗	✓
Zone mapping	✗	✗	✗	✓

CCS CONCEPTS

• **Information systems** → **Storage management**; *Flash memory*; • **Software and its engineering** → *Secondary storage*.

KEYWORDS

ZNS, Interference, NVMe Flash Storage, Emulation

1 INTRODUCTION

Solid-state drives (SSDs) have become the de facto standard for storing and processing data at high speeds. Today, SSDs can deliver microsecond access latencies, millions of I/O operations per second, and gigabytes of bandwidth per second [53]. However, delivering predictable SSD performance is challenging due to the significant required management effort for flash-based SSDs [2, 42] (e.g., garbage collection, parallelism management, wear-leveling). This flash management is traditionally hidden from the host behind the block interface, which exposes the SSD as a read/write anywhere device. To support this block-based interface, an SSD manages media transparently in the background but causes significant performance interference and, as a result, performance unpredictability in both latency and throughput [4, 15, 19, 20, 24, 33, 41].

To resolve this unpredictability, researchers have advocated for extending the conventional block-based SSD interface toward a more host-controlled interface. Examples of such interfaces include Software-Defined Flash (SDF), Open-Channel, Streams, and recently introduced Zoned Namespaces (ZNS) and Flexible Data Placement (FDP) [1, 4, 5, 29, 48]. We call SSDs, which support such interfaces, data placement SSDs.

Data placement SSDs give the host more control through explicit *I/O management operations* and typically group data in storage units, e.g., zones in ZNS.

Among these data placement SSDs, ZNS and FDP introduce I/O management operations to `finish` storage units, which allow the host to instruct the device to write out the unit with dummy data to release storage resources. This process of filling a unit is a key part of the reliability management of flash-based media, as it ensures that all media pages are programmed, either due to two-phase programming [6], program disturbs [10], or other media-related requirements [7]. In addition to `finish`, ZNS offers the ability to explicitly `reset` a zone, which invalidates any data associated with the zone and allows the host to reuse the zone for writing again.

While interference between I/O operations (i.e., read–write) is a well-studied phenomenon in data placement SSDs [3, 14, 41, 54, 58], interference caused by I/O management operations has yet to receive attention. This understudy is despite initial evidence demonstrating that I/O management operations (e.g., `finish`) have a non-negligible impact on I/O performance [45]. The source of such interference lies in how SSD-internal resources (e.g., NAND chips, dies, planes, channels, caches) are shared between storage units, I/O operations, and I/O management operations [59].

In this paper, we take a step back and systematically explore the design space of I/O management operations. We first demonstrate and quantify the performance (interference) of finish and `reset` on I/O operations on two types of commercially available SSDs: one with ZNS support, the Western Digital Ultrastar DC ZN540, and another with FDP support, which remains anonymous. We make eight observations, report three key findings, and introduce a novel quantitative model to quantify management interference, which together indicate that management interference is significant.

While I/O management interference is significant, we report that all currently available ZNS emulators, i.e., FEMU [40], NVMeVirt [35], and ConfZNS [59], do not have a function-realistic performance model for management operations. For example, they use a static cost for management operations, which has a dynamic cost. Further on, they assume a direct mapping of zones to physical resources, but mappings can be dynamic in ZNS [18, 45]. Despite this lack of performance realism, these emulators have been widely used in recently published literature [11–13, 18, 21, 22, 31, 35, 37–39, 43, 44, 46, 51, 55, 59, 62, 68, 69], which may not accurately reflect the performance when zones are finished.

Therefore, there is an explicit need to explore the design space of host-issued I/O management operations, incorporate them into emulators, and control I/O management interference. In this work, we take a two-pronged approach to address this need for ZNS. First, we identify ten key I/O management operation design choices and incorporate them into a new emulator, ConfZNS++. ConfZNS++ is implemented as an extension of the recently published, state-of-the-art ConfZNS emulator [59] and extends the emulator with explicit support for I/O management operations and zone mappings. In ConfZNS++, we incorporate seven of our discussed I/O management operation designs. As shown in Tab. 1, ConfZNS++ provides a more function-realistic performance model, allowing host-based software to explore I/O management designs. We verified ConfZNS++ by demonstrating similar performance trends to the ZN540 SSD and prior research [14, 28, 44].

Second, we design and implement two novel host-managed solutions to reduce I/O management interference: ZINC, a new I/O scheduler, and the `softfinish` operation, `finish` implemented on the host. ZINC (*ZNS interface-aware NVMe command scheduler*) relies on the assumption that I/O management operations typically run in background processes with lower urgency than foreground I/O. ZINC prioritizes I/O over I/O management and reduces `reset` interference by 56.9%. `Softfinish` implements `finish` on the host instead of the device to give the host more control over interference. Conventional `finish` transparently informs the SSD to fill a zone with writes, but when and how the zone is filled is beyond the host's control. As an alternative, `finish` can be implemented obliquely on the host by manually writing the unoccupied part of the zone. `Softfinish` uses this alternative; it uses host-managed `write` operations to fill zones and reduces `finish` interference by 50.7%.

We summarize our key contributions in this work as follows:

- We identify and quantify the performance interference of ZNS I/O management operations (i.e., `reset` and `finish`) on I/O operations for a commercially available NVMe ZNS device and showcase that `finish`'s latency model generalizes to ZNS and FDP SSDs.
- We introduce a quantitative model to quantify management on I/O performance interference.
- We design and implement ConfZNS++, a function-realistic emulator with performance (interference) models for management operations and zone mapping.
- We propose two methods to mitigate ZNS management performance interference on the host, I/O scheduler ZINC, which reduces `reset` interference by up to 56.9%; and `softfinish`, `finish` implemented in host software, which reduces `finish` interference by up to 50.7%.
- We publish all our code and data at https://github.com/stonet-research/systor-confznsplusplus-artifact to encourage reproducible research.

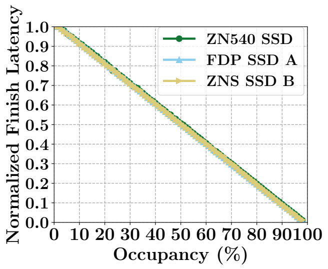


Figure 1: *Normalized ZNS `finish` and FDP `handle update` latency with increasing storage unit occupancy.*

2 MOTIVATION

In this section, we motivate addressing I/O management performance interference by giving background on I/O management operations, demonstrating that these operations are not endemic to ZNS but to flash management, and conducting an I/O management interference study on ZNS. Last, we introduce a quantitative I/O management interference model.

2.1 Background on I/O Management

SSDs with data placement support, such as ZNS and FDP, are designed to manage data placement efficiently. This efficiency is achieved by defining a storage unit that aligns with the physical characteristics of the underlying media. The result is an implicit contract that enables the host and device to collaborate on data placement. In the case of ZNS, this efficiency is demonstrated through the use of *zones*, where the storage capacity is divided into sequential-write-only zones with a set of LBAs managed as a single unit. Similarly, FDP introduces *reclaim units*, which the host can access through a *reclaim unit handle*. This handle serves as a reference to a writeable storage unit at any given time and is passed together with a specific write operation. The focus of this work is on two prevalent I/O management operations: `finish` and `reset`.

`Finish:` Due to the limited resources within an SSD and media-related requirements [6, 9], an SSD typically only supports a small number of *active* storage units. An active storage unit is a partially written unit that is yet to be filled. For example, for the ZN540 SSD, there is a maximum of 14 active/open zones, i.e., zones that can be in a partially written state.

Due to this limit, the host does not always have sufficient data to write into a specific storage unit and may, therefore, choose to "finish" a storage unit prematurely to increase the number of units where it can place its data. A finish is issued with `finish` in ZNS and `handle update` in FDP. Issuing this I/O management operation informs the device that it should free up resources by finishing up the unit. However, this is not a free operation.

Fig. 1 illustrates the costs of finishing storage units at different occupancies. The three lines represent the normalized time it takes to finish a storage unit (i.e., zone or reclaim unit) at a given occupancy for three commercially available SSDs: A ZN540 SSD, a TLC-based FDP SSD "A," and a TLC-based ZNS SSD "B." Interestingly, they each show the same behavior and the time to finish a storage unit directly correlates to its occupancy (a prior study showed similar results [14]). To provide a guiding reference to the normalized results, it takes approximately one second to finish a nearly empty zone on the ZN540. Further, this finishing time equals the cost of writing the zone on the ZN540. This equality is due to the finish internally filling up the unoccupied pages of the zone with writes, i.e., a finish is a series of device-issued writes.

Furthermore, we performed the same benchmark on a QLC-based ZNS SSD "C," which utilizes a write-back SLC buffer (see [57]) before writing to QLC. In that specific case, there was no link between the finish time and occupancy. We assume this is because data is initially stored in the SLC buffer, thus hiding the late write of the QLC. We, therefore, limit our study to SSDs without such write-back buffers.

`Reset:` Specific to the ZNS interface, it offers the capability to "reset" a zone through a single atomic operation, `reset`. Resetting a zone invalidates all host data associated with that specific zone and enables it to be written again. The reset explicitly informs the device that the physical resources associated with the zone can be recycled and reused for other purposes (e.g., to serve writes to another zone). This type of operation is often metadata-heavy on the SSD, as it must perform significant updates to its mapping table [27] to unmap each logical block address to be invalid. Furthermore, recycling the media enables various designs, such as delaying block erasures or issuing them immediately. While our evaluated SSDs show that they delay erasures until new writes occur, we explore various other designs in §4.

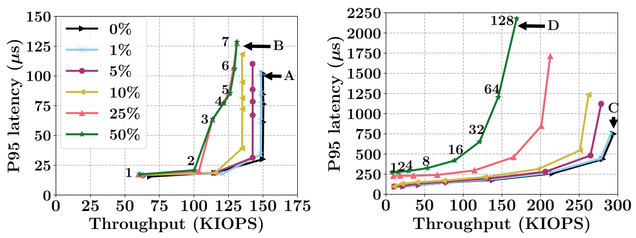
To summarize, both I/O management operations compete for the same resources as I/O. Therefore, we now explore the interference of these two operations on I/O performance.

Table 2: *Benchmarking environment.*

CPU	Dual socket Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz, 10 cores/socket, no hyper-threading
DRAM	256 GiB, DDR4
Storage	WD Ultrastar® DC ZN540 ZNS 1TB SSD. 904 zones with zone capacity 1,077 MiB; anonymous FDP SSD "A" and ZNS SSD "B" and "C"
Software	Ubuntu 22.04, kernel 6.3.8, modified fio with `append` and `finish` support (based on v3.32)

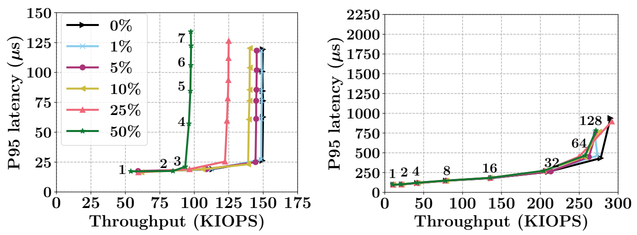
2.2 I/O Management Interference Study

Below, we demonstrate the quantitative evidence of interference between I/O and I/O management operations on ZNS.



(a) *On* `write` *with inter-zone concurrency.* (b) *On* `read` *with intra-zone concurrency.*

Figure 2: `Finish` *interference on (a)* `write`*; (b)* `read`*.*



(a) *On* `write` *with inter-zone concurrency.* (b) *On* `read` *with intra-zone concurrency.*

Figure 3: `Reset` *interference on (a)* `write`*; (b)* `read`*.*

Setup: Our benchmarking setup is shown in Tab. 2. We deploy all our benchmarking workloads on the ZN540 ZNS SSD. To confirm the generalizability of our `finish` interference results, we repeated our `finish` experiments on ZNS SSD "B" and made similar observations. We evaluate the interference of all I/O management (i.e., `finish` and `reset`) operations on all I/O operations exposed by ZNS. ZNS exposes two I/O operations for writing: `write` and `append`; and one operation for reading: `read`.

We benchmark I/O management interference by running a fio [25] process with two concurrent threads: a foreground thread running I/O and an interfering background thread running I/O management. Both threads are spatially separated (disjoint zones). We measure performance as the foreground thread's average throughput and P95 latency.

We scale the foreground thread's I/O using a setup similar to a previous study [14]. We define scalability in the *concurrency level* (CL), which is the number of concurrently issued operations. `Write` is inter-zone scalable (concurrent operations in disjoint zones), and `append` and `read` are intra-zone scalable (concurrent operations in a single zone). With inter-zone scalable operations we increase CL with the number of threads, each to a disjoint zone, and with intra-zone scalable operations we increase CL by increasing a single thread's queue depth to a single zone. For `write` and `append`, we use concurrency levels 1–7, and for `read` 1–128 in powers of 2 (based on each operation's saturation point).

We scale the background thread's management operations by increasing its *intensity*. Intensity is the maximum allowed throughput per second and is controlled by throttling an operation to a percentage of its peak performance (e.g., 50%). Before each workload, we first measure the operation's peak performance to determine the intensities to evaluate. In a workload, each `finish` is additionally preceded by a single page `write` as `finish` only affects zones with data; we assume this `write`'s interference is negligible.

For `reset` and `read` we prefill their assigned zones. We modify fio to support `finish` and `append` as workloads and exclusively use the *io_uring passthrough* mechanism [26]. *io_uring passthrough* delivers operations directly to the NVMe device driver, bypassing the block layer, which achieves performance close to the device hardware.

`Finish` interference: We now evaluate the interference of `finish` on I/O. Fig. 2 shows `finish` on I/O interference. The figure shows I/O performance in KIOPS (x-axis, higher is better) and P95 latency (y-axis, lower is better). The points on the lines represent the CL, and we observe that the throughput and latency increase monotonically with the CL (hence, one line is annotated). Each line is presented with a different percentage, indicating `finish` intensity. We measure peak `finish` throughput as 1.1 GiB/s (~1 IOPS); for example 25% equals approximately one operation every four seconds. In the plot, when an I/O operation saturates the device, a queuing effect takes place where the throughput remains stable, but latency increases sharply. We call this point the *saturation point*. For example, `write`'s saturation point is at the knee of the 0% line (CL=3, 149.3 KIOPS at 25.7 μs). Note that we do not plot `append` interference as it is comparable to `write` interference except for 50% past CL>2; `append` throughput decreases past this point (we do not know the reason for this anomaly). The similarity between `write` and `append` is expected as both issue the same operations to flash; they only differ in their implementation firmware (e.g., acceleration).

We observe that `finish` interferes significantly with all three I/O operations and make four observations. First (**Obs #1**), write operations (i.e., `write` and `append`) do not experience interference before their saturation point (CL=3 in Fig. 2a). Write performance at CL<3 is identical for all `finish` intensities. Second (**Obs #2**), write interference is significant beyond the saturation point and increases with the concurrency level. `Write` interference is highest at CL=7 (marked "A" and "B"), where it achieves 150.0 KIOPS and 101.9 μs in isolation and 131.1 KIOPS (12.6% lower) and 128.5 μs (20.7% higher) at 50% `finish`. Third (**Obs #3**), we observe that `finish` on `read` interference occurs irrespective of the saturation point (Fig. 2b), i.e., occurs at all concurrency levels. At CL=1, `read` achieves in isolation 11.3 KIOPS and 95.7 μs, and at 50% `finish`, 7.8 KIOPS (31.4% lower) and 272.4 μs (2.8× higher). Fourth (**Obs #4**), `finish` on `read` interference increases significantly with the concurrency level. At the highest concurrency level, CL=128 (marked "C" and "D"), `read` achieves in isolation 294.5 KIOPS and 749.6 μs and at 50% `finish` only 169.1 KIOPS (42.6% lower) and 2.2 ms (2.9× higher). For all I/O operations, `finish` interference is due to both operation types sharing the same hardware resources.

`Reset` interference: We now evaluate `reset` interference on I/O. Fig. 3 shows `reset` on I/O interference. Note that we do not plot `reset` on `append` as it interferes similarly to `reset` on `write`. The percentage indicates `reset` intensity as a percentage of the peak `reset` throughput. Note that the SSD performs `reset` faster than a typical media erase operation; hence, we assume that the `reset` operation is solely performing internal book-keeping operations, e.g., invalidating LBA entries. We measure its peak throughput (100% `reset`) to be 62 IOPS with an average latency of 16 ms (>62 GiB/s). This peak performance is confirmed by earlier research, which evaluated ZN540's `reset` latency to be high (i.e., 15–30 ms) [14].

We make four observations. First (**Obs #5**), `reset` on `write` interference (Fig. 3a) is significant, but only at high intensities (i.e., 25% and beyond). At the highest CL from 0% `reset` intensity to 25% and to 50% rates, the peak throughput degrades from 149.3 KIOPS to 124.8 KIOPS (16.5% lower) and 97.6 KIOPS (34.6% lower than 0%), respectively. Second (**Obs #6**), `reset` has a significantly greater impact on `write` throughput than tail latency. Third (**Obs #7**), there is only `reset` interference past the saturation point (CL=3). Fourth (**Obs #8**), unlike ZNS `write` operations, `reset` has negligible interference on `read` (Fig. 3b). This implies that `reset` and data writing commands share either ZNS internal flash channels, dies, or metadata structures in spite of the fact that these commands are issued to disjoint sets of zones; hence competing for resources. In the case of `read`, no such bottleneck is seen, with a slight performance difference due to command scheduling overheads (as indicated by the slightly diverging lines at high concurrency levels in Fig. 3b).

2.3 I/O Management Interference Model

Since no standardized model to *quantify* the interference level exists, we define a first-order quantifiable model to reason about interference *levels* among I/O management operations. This model is inspired by the earth mover's distance in a 2D space. We denote management interference as M^{Inter}. We model M^{Inter} in terms of two variables, P95 latency, and IOPS throughput, as we increase the concurrency level. We define M^{Inter} as the throughput/latency deviation between an operation running in isolation (*iso*) and an operation running concurrently with other operation(s) (*int*)—larger deviations indicate more interference. The model's intuition is that for

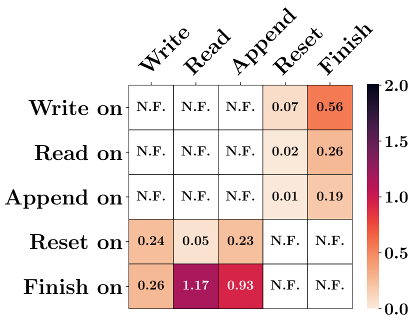


Figure 4: *M^{Inter} heatmap for one-on-one combinations of ZNS operations between isolated runs (at 0%) and interference runs (at 50%) on the ZN540; "N.F." is Not-The-Focus.*

two curves (with and without interference), we calculate the distance between each concurrency level's two points in a normalized 2D space. We model M^{Inter} as:

$$M^{Inter} = \frac{1}{n}\sum_{i=1}^{n}\sqrt{\alpha \times (\Delta T_i)^2 + \beta \times (\Delta L_i)^2} \tag{1}$$

With:

$$[\alpha + \beta = 1,\ 0 \leq \alpha \leq 1,\ 0 \leq \beta \leq 1]$$

$$\Delta T_i = (\frac{T_i^{\,int} - T_i^{\,iso}}{T_i^{\,iso}}) \tag{2}$$

$$\Delta L_i = (\frac{L_i^{\,int} - L_i^{\,iso}}{L_i^{\,iso}}) \tag{3}$$

In our model, T constitutes throughput and L latency. The ΔT (Equation 2) and ΔL (Equation 3) capture the relative difference in throughput and latency between *iso* (0% interference) and *int* (>0% interference) for a given concurrency level. Integer i indicates this level and scales up to user-configured maximum of n. Equation 1 takes the root mean square sum of the relative shift in throughput and latency with α and β weights (currently: both 0.5). Workloads can configure the weights based on their affinity for throughput or latency.

In this work, we bound our workloads between 0–50% interference, with interference having a decreasing monotonic impact on the isolated performance run (i.e., 50% interference having more impact than 25% interference). For example, we show two points in Fig. 2b, **C** (T: 294.50 KIOPS, L: 749.57 μs) on 0% interference, and **D** (T: 169.09 KIOPS, L: 2,179.02 μs) on 50% interference. Based on these numbers, we calculate the single interference level for CL=128 as: $\sqrt{0.5 \times (\frac{169.01-294.50}{294.50})^2 + 0.5 \times (\frac{2179.02-749.57}{749.57})^2} = 1.38$. The α and β values significantly impact the resulting M^{Inter} value. For example, interference at CL=128 is latency-dominant, and reducing its interference is thus more important for latency-sensitive applications (e.g., write-ahead logs). Setting the α to 0 and the β to 1 leads to an M^{Inter} of 1.91, and setting the α to 1 and the β to 0 leads to 0.43. In our setup, we have not explored this scenario and keep both values equal; however, we envision an online optimized/explorer using model equations with α and β values emphasized for throughput or latency.

Apart from the graphs shown thus far (Fig. 2, Fig. 3), we run additional combinations of I/O and I/O management operations. Fig. 4 shows our results for all I/O and management combinations for the interference quantification between 0% and 50% interference. We report that the interference values are different for each combination but omnipresent. There are combinations that have high interference (`finish` on `read`, 1.17) and combinations with low interference (`reset` on `read`, 0.05). Note that `finish` on `append` is higher than `finish` on `write` due to the earlier-mentioned anomaly. Based on the differing M^{Inter} values, the host should treat I/O management operations accordingly.

2.4 A case for Addressing I/O Management Interference

Based on our observations, we have three key findings:

(1) `Finish` has a significant impact on the tail latency and throughput of all concurrent I/O operations.
(2) `Reset` has an impact on the tail latency and throughput of concurrent `writes` and `appends` at high `reset` intensity (faster than a zone can be written).
(3) Operations that fill up resources like `finish` are present in multiple flash interfaces, e.g., in FDP, showcasing a need for addressing interference beyond ZNS.

We have also evaluated if NVMe namespaces provide performance isolation by repeating our experiments across multiple namespaces. We still observed I/O management interference across namespaces; hence, namespaces do not provide I/O management performance isolation. Together, these findings illustrate that (ZNS) SSDs have significant I/O management interference that needs to be addressed.

3 HOST-MANAGED SOLUTIONS FOR ADDRESSING INTERFERENCE

In this section, we showcase two actionable host-managed solutions to control I/O management interference: `softfinish`, an implementation of `finish` on the host, and ZINC, a new I/O scheduler. Both are visualized in Fig. 5.

3.1 `Softfinish`

`Finish` fills a zone by writing the unoccupied pages in a zone (§2.2) but does not give the host control over this writing. The performance model is thus static and etched into the SSD's firmware. This inflexibility leads to a semantic gap between the host and the device. This gap is due to the host being oblivious to the writing (design options detailed in §4.1) and the device being oblivious to the host's demands. Due to this gap, the host can not control when and how `finish` is executed. For example, when a `finish` is critical, it can not be prioritized, and when concurrent I/O is more critical, `finish` can not be slowed down. Further on, a `finish` can not be controlled by I/O schedulers on the host. Therefore, we propose `softfinish`, which replaces `finish` with a series of host-managed `writes`.

`Softfinish` is host-managed and provides semantics to control `finish` intensity, addressing the semantic gap. There are two parameters to control this intensity: the granularity of `write` requests and pauses between subsequent `writes`. A `softfinish` with large `writes` completes faster, reducing latency but increasing M^{Inter} on concurrent I/O. Similarly, M^{Inter} can be reduced by increasing the pause between `writes`. Another benefit of using `writes` is that any host-managed I/O scheduler, e.g., ZINC, can schedule these `writes`. I/O schedulers thus (re)gain control over `finish`. Disadvantages of `softfinish` are that each `write` adds round trip time to the `finish` latency, and `softfinish` can not be hardware-accelerated on the SSD. Further on, every `write` adds additional DMA traffic, increasing the performance cost for `softfinish`, especially as the `write` granularity decreases. `Softfinish` should thus be used when I/O is of higher urgency than I/O management performance.

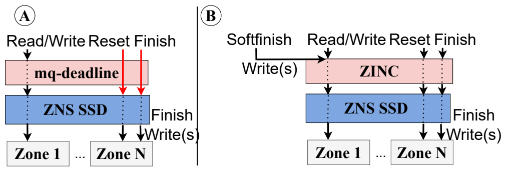


Figure 5: *Host-managed solutions for ZNS: (A) state-of-the-practice; (B) Combination of* ZINC *and* `softfinish`.

3.2 ZINC I/O Scheduler

I/O interference on the host is commonly controlled with I/O schedulers. These allow for improving performance QoS without application changes. Instead, practitioners have to swap the SSD's attached scheduler. This transparency is beneficial for addressing I/O management as the interference model differs between SSDs (see §4), and altering applications for each SSD is costly. Previous research has shown that I/O schedulers effectively address flash I/O interference, e.g., write-on read-interference [23, 61, 66]. Therefore, we postulate addressing I/O management interference with I/O schedulers.

To this end, we design a scheduler, ZINC, that (1) accounts for the highly asymmetric performance of I/O management operations (e.g., up to 4 orders of magnitude differences from

I/O); and (2) reduces I/O management interference for concurrent I/O. We design ZINC for generic I/O management operations but verify its applicability on the `reset` and `finish` operations of the ZN540 SSD (§2.2). ZINC is designed as a plug-and-play scheduler to allow switching the scheduler during runtime and uses the Linux kernel I/O scheduler interface. We base its `read` and `write` functionality on the state-of-the-practice mq-deadline scheduler. The mq-deadline scheduler ensures that only one `write` is issued to a zone concurrently and is, therefore, frequently an application requirement (e.g., ZenFS [65]). A disadvantage of mq-deadline's design is that it does not schedule I/O management operations but immediately dispatches them (see Fig. 5).

ZINC dispatches I/O and I/O management operations from software queues to hardware queues and prioritizes I/O over management I/O operations. ZINC achieves prioritization by sending I/O management operations at low intensity and throttling them when necessary. Intensity is controlled with the *reset epoch* parameter, which presents a static timing window between issuing consecutive operations. I/O management operations are always sent serially and are only dispatched at the end of each window to reduce their intensity to one concurrent operation. At the end of each window, ZINC analyzes the intensity of the concurrent workload—in particular, it considers (1) the amount of I/O completed since the last issued management operation; and (2) the latency cost of the issued I/O management operation compared to the ongoing I/O operation (i.e., their relative cost). ZINC uses this analysis to decide whether to dispatch or retain an I/O management operation. A management operation is dispatched if the number of issued (past and present) I/O operations has passed a configurable threshold, *write tokens*, to ensure fairness. If this threshold is not reached, the management operation is delayed and retained in the management queue. To prevent starvation, management operations are immediately dispatched after they are retained for a fixed amount of windows, defined by *maximum epoch holds*. Lastly, since management operations do not lead to interference at low concurrency levels (see §2.2), management operations are, therefore, only retained in the queue if the workload intensity is high, defined by a *minimum concurrency threshold*. If the I/O intensity is lower than this threshold, management operations are always dispatched immediately. As a result, this threshold can also be used for high priority I/O management operations (e.g., no more active zones) because if the amount of concurrently-issued I/O reduces, which typically happens if I/O needs to wait for high priority operations, the threshold is no longer reached.

Note that regardless of ZINC's configurations, it is not designed to be the best scheduler in every scenario; it is designed to showcase that management interference can be reduced for an application's needs at the cost of the performance of management operations. If this is not desirable, it can be configured to be identical to no scheduler.

4 DESIGN AND IMPLEMENTATION OF CONFZNS++

In this section, we discuss the design choices and their performance implications for `reset`, `finish`, and zone mapping, as well as how we implemented seven of them in ConfZNS++.

4.1 Finish

`Finish` informs an SSD to release a zone's resources by filling the zone. A zone is filled by writing the unoccupied pages of the zone with dummy data (see §2.1). A `finish` is thus equivalent to a series of subsequent `writes` but device-issued instead of host-issued. When designing ZNS SSDs, it should be specified when and how to issue these device-issued `writes`. This design choice has direct performance implications not only for the performance of `finish`, but also for concurrently running I/O. It affects I/O because device- and host-issued I/O contend for the same flash resources. Current literature does not detail `finish` designs; thus, we introduce three different `finish` designs (in §5.2, we show their similarities to physical ZNS SSDs in ConfZNS++). All these designs are realistic methods of filling a zone.

We discuss the three `finish` designs around the *intensity* of their `writes`. Higher intensity leads to more interference with concurrent I/O but lower `finish` latency. We discuss two method types to control intensity: static (i.e., fixed) and dynamic. Static methods concern the `write` request granularity and pause time between subsequent `write` requests. Larger requests lead to higher intensity as larger requests block underlying resources for an extended duration. Shorter pauses lead to higher intensity. Dynamic methods alter `finish` behavior based on concurrent I/O. For example, decreasing `finish`'s `write` intensity if there is concurrent I/O. Intensity can be decreased by stalling or preempting `finish`' `writes` during heavy I/O load. Prioritizing one I/O operation is a common optimization in I/O schedulers. For example, Kyber and FIOS prefer `read` over `write` [49, 52].

4.2 Reset

As explained in §2.1, `reset` informs the SSD that a zone's physical resources are ready to be reused, and its flash blocks should *eventually* be erased. The host, however, has no control over when and how these erasures happen; the device dictates these. In this section, we discuss five different `reset` designs, their relation to erasures, and their implications for the host.

The first design choice is *when* to erase. This choice involves two types of `reset`: synchronous and asynchronous. A synchronous `reset` immediately erases a zone's blocks, giving the host control over block erasure. This design has the

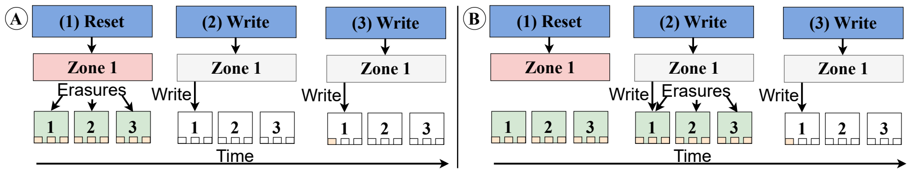


Figure 6: *Storage operations for* `reset` *followed by two* `writes`*: (A) direct mapping; (B) dynamic lazy mapping.*

advantage that the host can reduce erasure interference, e.g., by not concurrently issuing `reset` and I/O. A disadvantage of synchronous `reset` is that it is stressful for NAND to be in a prolonged erased state as it leads to more hardware errors as time progresses [8]. Additionally, synchronous `reset` leads to high I/O management latency as flash erasures are expensive, i.e., a representative TLC NAND [30] takes 3.5 ms to be erased compared to 700 μs for a write (program). Asynchronous `reset` defers erasures and only updates metadata. Typically, for asynchronous `reset` blocks are only erased when they need to be reused (reduces NAND stress by shortening the erased state). For example, the first `write` to a zone erases all of the zone's blocks, or the first `write` to a block erases the respective block (see §4.3).

The second design choice is *how* to erase. This choice is important because a zone's blocks can be erased independently. Further on, similar to reads and writes, each erasure blocks the underlying flash resources until it is completed, leading to interference. Thus, erasing a zone constitutes multiple independent erasures that each interfere with concurrent I/O. An optimization that addresses this choice is *partial erasures* [28, 44]. A partial erasure only erases a zone's (partially) occupied blocks; all empty blocks are ignored. This design is useful if the host `resets` partially filled zones as it reduces the number of erasures. Partial erasures lead to reduced latency and longer flash endurance as erasing is expensive. To implement partial erasures, an SSD needs to know the write pointer of a zone and a zone's block mapping. The alternative to partial erasures is *full erasures*, erasing all blocks unconditionally. The advantage of full erasures is lower complexity; the disadvantage is an increased number of erasures. A second `reset` design that addresses how to erase is *preemptive zone* `reset` [28]. This design recognizes that a `reset` is composed of multiple erasures and that a `reset` can be paused between erasures. Pausing is beneficial if there is a concurrent I/O workload. This design pauses `resets` and preempts erasures if there is concurrent I/O. Preemption reduces interference on concurrent I/O by decreasing `reset` intensity. However, its disadvantage is that it increases the time it takes for a block to be ready for writing (for both synchronous and asynchronous `reset`).

4.3 Zone Mapping

Another ZNS design choice is the mapping of zones to physical resources. The ZNS standard allows (re)mapping a zone to different flash resources over an SSD's lifetime; hence, the host is oblivious to a zone's flash resources. For example, a zone can be remapped to different blocks after it is `reset`. Remapping aids wear-leveling and allows asynchronous I/O management operations to be implemented without blocking the next I/O (i.e., no need to wait for a zone's blocks to be erased). We define two forms of zone mapping in ZNS: *direct* and *dynamic*. We illustrate examples of both types in Fig. 6.

Direct mapping is a static mapping of zones to physical resources (i.e., channels, ways, planes). Synchronous `reset` for direct mapping is visualized in Ⓐ (Fig. 6); `reset` erases the zone's blocks, and the subsequent `write` uses the same flash resources as before the `reset`. Direct mapping allows for a predictable performance model but has wear-leveling challenges as `writes` are not evenly distributed.

Dynamic mapping addresses the wear-leveling challenge as its mapping of zones to physical resources is not fixed [44]. Dynamic mapping also allows hiding the latency cost of I/O management operations with asynchronous `resets`. For example, if zone "X" is `reset`, the SSD can invalidate zone "X" and transparently map "X" to new resources (while the originally mapped resources are erased in the background). A prominent form of dynamic mapping is *lazy mapping* [18, 44]. In lazy mapping, a zone is not mapped to physical resources until it receives its first write. On the first zone write, the SSD checks if empty blocks are available. If there are, the zone is mapped to these blocks. If not, the zone is mapped to non-empty (invalidated) blocks, and all of these blocks are erased *before* the issued `write`. Lazy mapping's procedure of a `reset` followed by a `write` is visualized in Ⓑ. Due to postponing erasures, lazy mapping leads to erasure latency on the first zone write. Alternatively, lazy mapping can be implemented to only erase on the first `write` to a block, spreading erasures over a zone's `writes`.

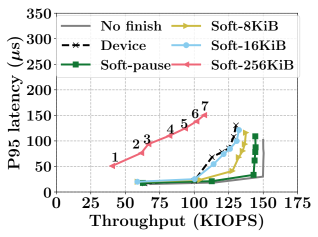
4.4 ConfZNS++ Implementation

We have detailed ten designs for `reset`, `finish` and zone mapping. Despite their performance implications, none of these designs are implemented in the state-of-the-art emulators for ZNS. We detail the current state of function-realistic ZNS emulation in Tab. 1. Both ConfZNS [59] and NVMeVirt [35] have a function-realistic model for I/O operations. They support direct mapping from zones to flash resources, which includes channels, planes, and dies. `Read` and `write` completion latency is implemented as the next time a resource will be available (i.e., queues), and each operation adds its respective configurable (e.g., SLC, TLC) latency. However, as of writing, both emulators do not emulate `finish`'s latency. They also do not support asynchronous `reset`, partial erasures, or resource blocking erasures. Lastly, they do not support dynamic zone mapping.

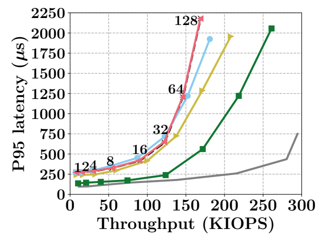
To address this gap, we introduce the ConfZNS++ emulator, which supports seven of our discussed designs (we do not implement preemptive `reset`, pauses in `finish`, or dynamic `finish`). ConfZNS++ is built on top of the ConfZNS emulator because it is state-of-the-art; hence, ConfZNS++ has all of its I/O functionalities. ConfZNS++ additionally supports synchronous `finish`, implemented as a sequence of sequential `writes` (same latency model and implementation). These `writes` are issued one at a time and at a configurable granularity, exposing the `finish` intensity design choice. On `write` completion, ConfZNS++ issues a new `write` unless the zone is full (respecting ZNS' sequential write constraint). ConfZNS++ implements resource-blocking erasures, synchronous and asynchronous `reset`, and partial and full zone erasures. In order to support `reset`, we add block mapping (apart from already extant plane, channel and die mapping). With block mapping, each zone is linked to blocks, parallelized, and striped over its channels, ways, and planes. On a `reset`, a zone's blocks are erased according to the specified latency model, and their combined latencies are added to the completion time of the respective channels and planes. Synchronous `reset` adds latency to the underlying resources immediately upon issuing a `reset`. With the asynchronous `reset` model, the `reset` latency equals the round trip time, and the erasures are fused with a later `write` or `append` to a zone, see §4.3. In order to implement partial erasures, we only add the erasure latency of (partially) occupied blocks, i.e., empty blocks add no latency. Lastly, ConfZNS++ supports lazy mapping, which is implemented as a queue of virtual and physical zones. The host interacts with virtual zones; physical zones are assigned on the first write to a zone. If the physical zone has data, it is first erased.

5 EVALUATION

In this section, we establish the efficacy of our host-managed solutions and showcase the implications of I/O management designs using our ConfZNS++ emulator.



(a) *On `write` with inter-zone concurrency.*



(b) *On `read` with intra-zone concurrency.*

Figure 7: *Device- and software `finish` interference on I/O.*

5.1 Host-managed Solutions for Addressing Interference

In this section, we evaluate if we can reduce management interference on the host for `finish` with `softfinish` and `reset` with ZINC. The benchmarking setup is shown in Tab. 2, and our workloads are identical to §2.2. We evaluate using the commercially available ZN540 SSD.

Reducing `finish` interference with `softfinish`: To evaluate if `softfinish` can reduce interference effectively, we compare the operation against `finish` and establish the trade-offs its configurations provide. We implement `softfinish` directly in fio and expose configurations for `write` granularity and pause (§3). We repeat the `finish` on `write` and `read` experiments from §2.2. We evaluate `softfinish`'s configurations, at different granularities and pause time. We configure `softfinish` with granularities of 8 KiB (similar to `write` in §2.2), 16 KiB, and 256 KiB and without a pause. We use these granularities to show the impact of both small and large `writes`. Additionally, we repeat the 8 KiB configuration with 50 μs pauses.

Fig. 7a shows `finish` on `write` interference. The lines with "Soft" use `softfinish`, "Device" uses `finish`, and "pause" indicates 8 KiB with 50 μs pauses. We make three observations. First, `finish`'s interference model can be accurately reproduced with `softfinish` on this SSD. 16 KiB `softfinish` has the same interference on `write` as `finish` (0.26 M^{Inter}). Second, `write` granularity has a significant impact on interference. 8 KiB at highest concurrency reaches 137.5 KIOPS and 116.2 μs (0.13 M^{Inter}), compared to 107.2 KIOPS and 150.5 μs for 256 KiB (0.68 M^{Inter}, 5.4× higher). Third, pause reduces interference significantly. 8 KiB with pause reach 144.3 KIOPS and 109.1 μs (0.17 M^{Inter}).

Fig. 7b shows `finish` on `read` interference. We have similar observations to `finish` on `write` interference. 256 KiB interference on `read` is similar to `finish` on `read` interference (1.17 M^{Inter}). 8 KiB interference is significantly less than `finish`'s interference (1.06 M^{Inter}, 9.2% lower), with a `read` throughput of 208.1 KIOPS (23.1% higher) and latency of 1,957.9 μs (10.15% lower). 8 KiB with

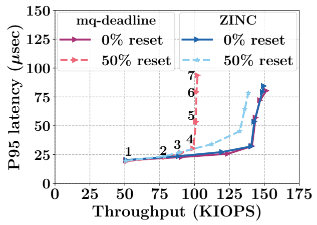


Figure 8: *`Reset` on `write` interference for I/O schedulers.*

pause achieves the lowest interference compared to `finish` 0.58 M^{Inter}, 50.7% lower) with 260.9 KIOPS throughput (54.3% higher) and 2,056.2 μs latency (5.6% lower). Notably, 256 KiB on `read` interference is similar to `finish`, whereas 16 KiB on `write` interference is similar to `finish`. For all configurations, the impact on throughput is higher than on latency. Thus, for latency-sensitive host workloads, this solution does not significantly reduce the interference (e.g., for 8 KiB with pauses, $\alpha = 0$ and a $\beta = 1$, the M^{Inter} is 0.06).

We also evaluated the impact of `softfinish`'s configurations on `softfinish`'s performance (not visualized). Note that `softfinish` adds additional DMA traffic. The overheads of this additional traffic should be benchmarked to understand the effectiveness of `softfinish` on different hardware platforms; we leave this analysis to future work. In our experiments, all reported numbers are inclusive of this overhead. We observe that decreasing the granularity or increasing the pause time decreases `softfinish` performance. If running the `write` experiment for `softfinish` with no interference (no finish), throughput degrades from 1.1 IOPS for 256 KiB (identical to device-level `finish`) to 0.23 IOPS for 8 KiB (78.3% lower) and to 0.1 IOPS (95.1% lower) for 8 KiB with pause. When running the experiment with concurrent `writes` (CL=7, 50%), the performance of `softfinish` performance decreases (expected), for example, to 0.30 for 256 KiB (0.14 for `finish`). Similar to what was observed in §2.2(not plotted), both `finish` and `softfinish` latency decreases with the zone occupancy. `Softfinish` thus presents a clear trade-off between `finish` and I/O performance. If completing `softfinish` is urgent (e.g., in the foreground), it should be configured with large granularity and no pause. If concurrent I/O has higher urgency (e.g., in the background), small granularity and pauses should be used.

In short, `softfinish` reduces M^{Inter} from 0.26 to 0.17 for `write` and from 1.17 to 0.58 for `read` compared to `finish` (up to 50.7% lower). `Softfinish` enables the host to make a trade-off between `finish` and I/O performance, a trade-off that `finish` does not provide.

Reducing `reset` interference with ZINC: We designed the ZINC scheduler to reduce management interference; below, we evaluate its efficacy. We implement ZINC as a kernel module by extending the mq-deadline scheduler. To assess if ZINC reduces `reset` interference in practice, we repeat the benchmark from §2.2 and use the Linux storage stack instead of passthrough to enable Linux I/O scheduler functionality. The Linux storage stack leads to lower performance than passthrough [26] but does not change the storage saturation point or interference patterns. To ensure a fair comparison, we compare ZINC against the state-of-the-practice mq-deadline. The ZINC scheduler is configured with a *minimum concurrency threshold* and *maximum epoch holds* of 3, *reset epoch* of 64 ms, and *write tokens* of 20,000. This configuration leads to the lowest interference and bounds the `reset` dispatch latency to a maximum of 192 ms. We obtained this configuration through grid search and used multiples of the `reset` latency as a stepsize for the *reset epoch*. In our grid search, we consider throughput and latency interference equally important and evaluated with an M^{Inter} model with an α and β value of 0.5 (see §2.2). Dependending on the host's needs, only these values need to be changed to repeat the search.

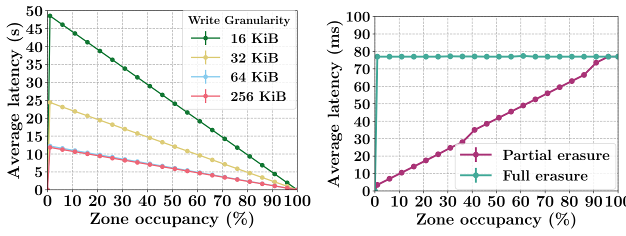
Fig. 8 shows `reset` on `write` interference for both I/O schedulers (we only plot 0% and 50%). We observe `reset` interference to be significantly less for ZINC (0.11 M^{Inter}) than for mq-deadline (0.17 M^{Inter}, 56.9% lower). At 50% interference, ZINC has up to 35.5% higher throughput and 16.4% lower P95 tail latency (evaluated at CL=7) than mq-deadline. On the other hand, ZINC has higher `reset` latency. We evaluated (not visualized) ZINC's P95 `reset` latency to be 20.1 ms at CL=1 and 210.8 ms at CL=7 and mq-deadline's to be 19.8 ms at CL=1 and 20.6 ms at CL=7. When there are no concurrent `resets`, ZINC also has slightly lower throughput but no significant latency difference. For example, at CL=7, throughput is 1.7% lower. Note that we do not plot `reset` on `read` interference as this interference is negligible for this device (see §2.2).

Last, we evaluate the software overhead of ZINC on the host by repeating prior interference experiments with ZINC and mq-deadline and compare their CPU usage. We do not observe (not visualized) statistically significant differences in CPU utilization; hence, ZINC does not lead to an increase in CPU overhead over mq-deadline.

5.2 ConfZNS++

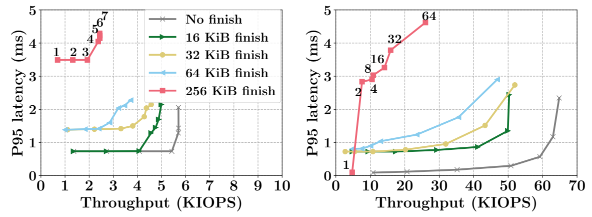
Below, we evaluate ConfZNS++ and showcase the performance implications of its various I/O management designs. We determine the relation between zone occupancy and I/O management latency, evaluate `finish` and `reset` interference, and showcase the performance impact of lazy mapping.

Setup: We evaluate ConfZNS++ with a configuration based on the large zone model of the ZN540 and a latency model based on a 3D NAND flash SSD [30]. This configuration uses 768 16 KiB page blocks, a zone size of 2 GiB, a zone capacity of 1056 MiB, and 48 zones (96 GiB SSD). Additionally, we



(a) `Finish`. **(b)** *Synchronous* `reset`.

Figure 9: *ConfZNS++ I/O management latency.*



(a) *On* `write` *with inter-zone concurrency.* **(b)** *On* `read` *with intra-zone concurrency.*

Figure 10: *ConfZNS++* `finish` *interference on I/O.*

use a parallelism model with an intra-zone scalability of four: four channels, one way, one die, and one plane. We use this model to simplify the explanation of our interference results. We use a latency of 700 μs, 60 μs, and 3.5 ms for flash write, read, and erasure, respectively. Note that in our experiments, throughput is lower than in §2.2 as we evaluate without SSD buffers or caches, which directly exposes flash latency.

Zone occupancy: As discussed in §2.1 and §4, I/O management latency depends on zone occupancy. We evaluate this dependency in ConfZNS++ by repeating the occupancy experiments from §2.1 for both `finish` and `reset` under various configurations. We configure `finish` with granularities of 16 KiB (page size), 32 KiB, 64 KiB, and 256 KiB to establish the impact of `finish` intensity. We configure `reset` with synchronous `reset` (§4) to directly measure erasure latency and use both full- and partial erasure.

Fig. 9a shows the average `finish` latency (seconds) as a function of zone occupancy (percentage of capacity). We observe a linear relation between occupancy and latency (expected), from 48.6 s for one-page occupancy to 32.1 μs for close to full occupancy (6 orders of magnitude). These observations show similar trends to the physical devices in §2.1. We also observe higher `finish` intensity to reduce `finish` latency; one-page occupancy with 256 KiB achieves 4.1× lower latency than 8 KiB.

Fig. 9b shows average `reset` latency (milliseconds) as a function of occupancy. *Full erasures* require, regardless of occupancy, 77.0 ms, which is equal to the number of blocks a zone has (88) times the block erasure latency (3.5 ms) divided by the intra-zone scalability (4). The scalability matters as it allows multiple erasures to proceed in parallel. *Partial erasures* show an increasing relation between occupancy and latency, confirming prior research [28]. For example, a `reset` at 50% occupancy takes 38.5 ms (50% of 77.0 ms).

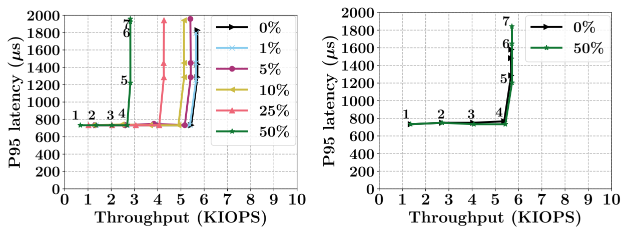
`Finish` interference: We evaluate `finish` interference using the experiments from §2.2. We change the I/O request size to 16 KiB (i.e., page size). We configure `finish` with the granularities used in our occupancy experiment.

Fig. 10a shows `finish` on `write` interference; apart from "no `finish`," all lines are for 50% `finish` interference (percentage based on highest throughput, 256 KiB). Similar to §2.2, we observe significant `finish` on `write` interference. Further on, we observe interference to increase with higher `finish` granularity; this increase is similar to `write` granularity in `softfinish` (see §5.1), confirming that ConfZNS++'s `finish` behaves similarly to `softfinish`. At the highest concurrency, `write` in isolation achieves 5.7 KIOPS and 2.1 ms, whereas `write` with concurrent 16 KiB `finish` reaches 5.0 KIOPS and 2.5 ms (0.14 M^{Inter}), and `write` with concurrent 256 KiB `finish` reaches 2.4 KIOPS and 4.3 ms (2.17 M^{Inter}). 256 KiB leads to high interference as the flash page size is 16 KiB, and intra-zone scalability is 4; thus, large (e.g., 256 KiB) requests block the underlying resources multiple times (i.e., four times). For similar reasons, we observe no interference at 16 KiB granularity and low CL (<4) because at this granularity, `finish` and `write` combined do not use the full scalability. We observed similar behavior in §2.2, where a CL below the saturation point does not lead to interference (see **Obs #1–2**).

Fig. 10b shows `finish` on `read` interference. Observations are similar to §2.2; `finish` has significant interference on `read` and occurs at all concurrency levels (**Obs #3**). `read` increases significantly with the concurrency level (**Obs #4**). `Read` in isolation achieves 64.8 KIOPS and 2.3 ms, whereas `read` with concurrent 16 KiB `finish` achieves 50.3 KIOPS and 2.4 ms (1.81 M^{Inter}), and `read` with concurrent 256 KiB `finish` achieves 26.0 KIOPS and 4.6 ms (5.66 M^{Inter}).

`Reset` interference: We evaluate ConfZNS++'s synchronous and asynchronous `reset` interference by repeating the `reset` on `write` experiments from §2.2. We exclude `reset` on `read` as we are interested in I/O interference in general. ConfZNS++'s asynchronous `reset` uses lazy zone mapping. This mapping is dynamic; therefore, we restart ConfZNS++ before each run to ensure reproducibility. In a later experiment, we evaluate this remapping in more detail.

Fig. 11a shows synchronous `reset` on `write` interference. We observe significant interference on `write`. `Write` achieves 5.7 KIOPS and 1.8 ms latency in isolation and 2.8 KIOPS and 2.0 ms latency with 50% `reset` (0.37 M^{Inter}). This interference is due to (1) erasures blocking flash resources, and (2) erasures having higher latency than writes



(a) *Synchronous* `reset`. **(b)** *Asynchronous* `reset`.

Figure 11: *ConfZNS++* `reset` *interference on* `write`.

(i.e., longer blocking). By directly exposing erasure interference in `reset`, synchronous `reset` thus enables the host to optimize by deciding when to issue `reset`, e.g., with ZINC.

Fig. 11b shows asynchronous `reset` interference and is, unlike synchronous `reset`, insignificant. This lack of interference is due to asynchronous `reset` not issuing erasures (or other flash operations) directly. These erasures are caused by other operations (e.g., first `write` to a zone). Thus, asynchronous `reset` is limited to metadata operations, and therefore, it is generally ineffective for the host to schedule `reset` to mitigate interference. There is an exception to this rule, however. We assume the evaluated ZN540 uses asynchronous `resets`, although it does have observable interference at high `reset` intensity because of potential resource contention on non-flash related SSD structures (e.g., metadata). We do not emulate such behavior in ConfZNS++ as it is unrelated to flash management. Note that regardless of what causes interference, host-managed solutions such as ZINC are effective for reducing `reset` interference (see §5.1).

Erasures in lazy mapping: Zone mapping significantly impacts the performance of both I/O management and I/O operations. We demonstrate this impact by evaluating the dynamic lazy zone mapping in ConfZNS++. This mapping issues erasures with the first write to a zone (or block), not with `reset`. This erasure causes the first zone `write` to take significantly longer to complete, which we demonstrate by running two workloads on both direct (synchronous `reset`) and lazy mapping (asynchronous `reset`) and measuring the latency of the first zone `write`. We reboot ConfZNS++ before each run. The first workload is *sequential fill*, which sequentially fills the device twice with 16 KiB (page size) `writes`. After the first fill, all zone mappings in both mapping algorithms are exhausted (i.e., no unallocated storage), and therefore, blocks need to be erased before each zone allocation. The second workload is *repeat zone*, which repeatedly fills one zone until all physical blocks are at least mapped twice in lazy mapping (i.e., we fill the device twice).

For direct mapping, we observe (not visualized) that all a zone's `writes` (also the first) have a P95 latency of 733.2 μs. For lazy mapping, on the other hand, there is a significant difference between a zone's first and subsequent `writes`. Similar to direct mapping, the P95 latency for `writes` is 733.2 μs, but the first `write` has a higher latency. If there are flash resources to allocate, i.e., the first fill in *sequential fill* or the first number of zone repeats in *repeat zone*, latency is identical to subsequent `writes`. However, if there are no more resources to allocate, the first `write`'s latency is 77.7 ms on average. This latency is equal to the combination of a `write` full erasure (77 ms, see Fig. 9b). With lazy mapping, the host can optimize interference based on a zone's first `write`, i.e., the host needs to decide when to open a zone for writing.

6 DISCUSSION

In this paper, we demonstrated I/O management interference on ZNS with micro-benchmarks. Below, we discuss the generalizability of our experiments in relation to applications.

`Finish`: In our experiments, we issue `finish` at a zone occupancy of a few pages (see §2.2 and §5.1). However, it is generally more beneficial to `finish` zones that are close to full as it reduces space amplification and increases drive endurance (less `writes`). Further on, instead of filling the unoccupied pages with dummy data, operations such as `softfinish` can also be designed to fill with application data, e.g., read caches [69]. Therefore, ZNS-friendly file systems/applications such as ZenFS [65] prefer to `finish` zones closer to full (not always possible). Regardless of this preference, the `writes` issued by `finish` lead to interference. Instead, `finish` interference concerns a shorter period. In §2.2, we demonstrate a direct correlation between unit occupancy and `finish` latency; thus, `finish` on "fuller" zones has reduced latency and fewer `writes`. Applications that fill such zones are thus given control over a shorter sequence of `writes` to fill a zone but do have to make the same design choices (e.g., `softfinish` intensity).

`Reset`: In our motivational `reset` benchmark (see §2.2), we issue `reset` at a high number of requests per second. For example, with the ZN540 SSD, 50% `resets` amounts to roughly 62 IOPS (62 GiB/s). This bandwidth is higher than that zones can be erased; thus, we hypothesize it uses asynchronous `reset`. Zones can not be filled at that speed; hence, such `reset` behavior is reserved for bulk `resets`, i.e., if there are large deletes. Therefore, on SSDs with asynchronous `reset`, we recommend addressing `reset` interference for applications that issue `reset` in bulk. However, if an SSD issues a synchronous `reset`, `reset` throughput will be significantly lower, and these `resets` lead to interference also for sporadic usage (see §4). For example, applications such as the file systems ZenFS [65] and F2FS [36] send small bursts of `reset`. For synchronous `reset`, we thus recommend addressing `reset` interference independent of the workload, for example, with ZINC. Such differences in recommendations exemplify that ZNS software should be tested on multiple designs (e.g., with ConfZNS++).

7 THREAT TO VALIDITY

The results in this work largely confirm ZNS management behavior, which is hypothesized in its development. Our experiments and benchmarks are *empirically driven, user-observed* behavior for multiple ZNS- and one FDP device. We are aware that such benchmarking has risks. Therefore, we have consulted and verified our observations with a global SSD manufacturer for the SSDs used in §2.2 and §5.1. However, we are aware that (ZNS) implementations differ between devices and SSD internals are typically hidden, making it challenging to generalize our findings. Nonetheless, we believe this paper makes strong contributions by performing a first-of-its-kind characterization of I/O management interference, exposing this behavior in an emulator, and providing workable solutions to address such interference on the application level for ZNS. We have open-sourced all of our code and data sets to ensure that other research can use our code and our research can be reproduced.

8 RELATED WORK

I/O interference on flash-based SSDs is a well-studied but complex phenomenon [24, 49] and becomes ever more important as storage is shared by multiple tenants [67]. In this paper, we investigate I/O management interference with an in-depth characterization of ZNS SSDs. Our characterization extends on top of prior ZNS characterizations [3, 14, 45, 58, 63] and SSD performance (interference) characterizations [17, 24, 32, 34, 41, 42, 64]. Bae et al. [3] and Min et al. [45] study inter-zone I/O interference and observe that `reset` interferes with concurrent I/O. Doekemeijer et al. [14] establish the relation between zone occupancy and management I/O latency and identify I/O on `reset` interference. We establish why, when, and how I/O management operations interfere with I/O.

ConfZNS++ extends on prior emulator research. FEMU is a state-of-the-practice SSD emulator with native support for ZNS [40]. ConfZNS continues on FEMU and allows exploring the trade-offs between ZNS parallelism and isolation for I/O operations [59]. NVMeVirt is a state-of-the-art emulator with function-realistic ZNS support for I/O operations [35]. Various SSD simulators similarly allow exploring SSD design choices. MQSim [60] allows for studying multi-queue SSDs, and Amber [16] models the CPU, DRAM, and flash of SSDs.

Various researches discuss zone mapping and I/O management operation design [18, 28, 44, 45, 69]. Long et al. discuss reducing wear leveling with state-of-the-art zone mapping designs [44]. ZNS+ extends ZNS with additional semantics, e.g., copy operations, to provide more host-side control [18]. Jung et al. introduce preemptive `reset` [28], and multiple works discuss partial erasures [28, 44]. Zhu et al. propose finishing a zone by using unoccupied pages as a read cache to reduce inter-zone I/O interference [69].

Similar to ZINC, there are plenty of I/O schedulers that address latency-asymmetric operations by prioritizing one operation over another (e.g., `read` over `write`) [32, 49, 56]. No I/O management-aware scheduler currently exists for ZNS, but ZNS does provide orthogonal schedulers. eZNS and the scheduler by Bae et al. [3, 45] reduce inter-zone I/O interference. Fair-ZNS [43] addresses I/O fairness with a scheduler on the device-level instead of the host. Note that our proposed solutions do not apply for traditional SSDs as these do not have an API for I/O management.

Various works reduce ZNS I/O interference on the application-level [46, 47, 50, 54], albeit none on the management level. ZenFS+ extends the *ZenFS* file-system to schedule its I/O based on inter-zone I/O interference patterns [47]. Other works generally address interference by reducing the overhead of application issued GC (i.e., data migration between zones) [39, 46, 50, 54].

9 CONCLUSION

Data placement interfaces, such as ZNS and FDP, promise better interference management with increased host control. However, they simultaneously expose several new I/O and I/O management operations to the host. Using commercially available ZNS and FDP SSDs, we demonstrate that these new I/O management operations lead to significant interference on concurrent I/O and provide a first-of-its-kind interference model to quantify this interference.

To give the host the ability to address these challenges effectively, support from emulators and actionable solutions need to be provided. To this end, we introduce and validate ConfZNS++, a function-realistic emulator for ZNS with I/O management support. We also give concrete solutions to reduce management interference with ZINC for `reset` and `softfinish` for `reset`, observing interference reductions of 56.9% for `reset` and 50.7% for `finish`, respectively.

ACKNOWLEDGMENTS

We thank the SYSTOR'24 reviewers and our shepherd, Youjip Won, for their helpful and constructive feedback. We also want to thank the AtLarge group at VU Amsterdam and Bala Chandrasekaran for their input. This work is supported by generous hardware donations from Western Digital and partially supported by Netherlands-funded projects (NWO OffSense, NWO OCENW.KLEIN.561, and GFP 6G FNS), and EU-funded projects MSCA CloudStars (g.a. 101086248) and Horizon Graph Massivizer (g.a. 101093202). Krijn Doekemeijer is funded by the VU PhD innovation program.

REFERENCES

[1] Accessed: 2024-05-20. NVMe 2.0 - TP 4146, FDP. https://nvmexpress.org/wpcontent/uploads/NVM-Express-2.0-RatifiedTPs_12152022.zip.

[2] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference (ATC'08)*. USENIX Association, USA, 57–70. https://www.usenix.org/legacy/event/usenix08/tech/full_papers/agrawal/agrawal.pdf

[3] 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 (HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 79–85. https://doi.org/10.1145/3538643.3539744

[4] 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 the 2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. USENIX Association, 689–703. https://www.usenix.org/conference/atc21/presentation/bjorling

[5] Matias Bjørling, Javier González, and Philippe Bonnet. 2017. LightNVM: the Linux Open-Channel SSD Subsystem. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST'17)*. USENIX Association, USA, 359–373. https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling

[6] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch. 2017. Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*. IEEE Computer Society, 49–60. https://doi.org/10.1109/HPCA.2017.61

[7] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. 2012. Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '12)*. EDA Consortium, San Jose, CA, USA, 521–526. https://doi.org/10.1109/DATE.2012.6176524

[8] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. 2015. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015*. IEEE Computer Society, 438–449. https://doi.org/10.1109/DSN.2015.49

[9] Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu. 2015. Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*. IEEE Computer Society, 551–563. https://doi.org/10.1109/HPCA.2015.7056062

[10] Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai. 2013. Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation. In *2013 IEEE 31st International Conference on Computer Design, ICCD 2013, Asheville, NC, USA, October 6-9, 2013*. IEEE Computer Society, 123–130. https://doi.org/10.1109/ICCD.2013.6657034

[11] Yun-Chih Chen, Chun-Feng Wu, Yuan-Hao Chang, and Tei-Wei Kuo. 2023. ZoneLife: How to Utilize Data Lifetime Semantics to Make SSDs Smarter. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 42, 8 (2023), 2488–2499. https://doi.org/10.1109/TCAD.2022.3224898

[12] Gyuseok Choe, Youngmin Lee, and Won Woo Ro. 2022. Analysis of SSD with Logical to Physical Address Mapping of Hot Data to Single Level Cell Area. In *2022 37th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. 1–4. https://doi.org/10.1109/ITC-CSCC55581.2022.9894873

[13] Krijn Doekemeijer, Zebin Ren, Nick Tehrany, and Animesh Trivedi. 2024. ZWAL: Rethinking Write-ahead Logs for ZNS SSDs with Zone Appends. In *Proceedings of the 4th Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '24)*. Association for Computing Machinery, New York, NY, USA, 9–16. https://doi.org/10.1145/3642963.3652203

[14] Krijn Doekemeijer, Nick Tehrany, Balakrishnan Chandrasekaran, Matias Bjørling, and Animesh Trivedi. 2023. Performance Characterization of NVMe Flash Devices with Zoned Namespaces (ZNS). In *IEEE International Conference on Cluster Computing, CLUSTER 2023, Santa Fe, NM, USA, October 31 - Nov. 3, 2023*. IEEE, 118–131. https://doi.org/10.1109/CLUSTER52292.2023.00018

[15] Nima Elyasi, Changho Choi, Anand Sivasubramaniam, Jingpei Yang, and Vijay Balakrishnan. 2019. Trimming the Tail for Deterministic Read Performance in SSDs. In *IEEE International Symposium on Workload Characterization, IISWC 2019, Orlando, FL, USA, November 3-5, 2019*. IEEE, 49–58. https://doi.org/10.1109/IISWC47752.2019.9042073

[16] Donghyun Gouk, Miryeong Kwon, Jie Zhang, Sungjoon Koh, Wonil Choi, Nam Sung Kim, Mahmut T. Kandemir, and Myoungsoo Jung. 2018. Amber: Enabling Precise Full-System Simulation with Detailed Modeling of All SSD Resources. In *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018*. IEEE Computer Society, 469–481. https://doi.org/10.1109/MICRO.2018.00045

[17] Keonsoo Ha, Jaeyong Jeong, and Jihong Kim. 2016. An Integrated Approach for Managing Read Disturbs in High-Density NAND Flash Memory. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 35, 7 (2016), 1079–1091. https://doi.org/10.1109/TCAD.2015.2504868

[18] 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 2021, July 14-16, 2021*. USENIX Association, 147–162. https://www.usenix.org/conference/osdi21/presentation/han

[19] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. 2020. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, Article 10, 18 pages. https://www.usenix.org/conference/osdi20/presentation/hao

[20] Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2017. The Unwritten Contract of Solid State Drives. In *Proceedings of the Twelfth European Conference on Computer Systems (EuroSys '17)*. Association for Computing Machinery, New York, NY, USA, 127–144. https://doi.org/10.1145/3064176.3064187

[21] Yun-Shan Hsieh, Bo-Jun Chen, Po-Chun Huang, and Yuan-Hao Chang. 2024. PRESS: Persistence Relaxation for Efficient and Secure Data Sanitization on Zoned Namespace Storage. In *Proceedings of the 29th Asia and South Pacific Design Automation Conference (ASPDAC '24)*. IEEE Press, 341–348. https://doi.org/10.1109/ASP-DAC58780.2024.10473986

[22] 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 Trans. Archit. Code Optim.* 20, 3, Article 45 (aug 2023), 26 pages. https://doi.org/10.1145/3608476

[23] Jaehyun Hwang, Midhul Vuppalapati, Simon Peter, and Rachit Agarwal. 2021. Rearchitecting Linux Storage Stack for μs Latency and High Throughput. In *15th USENIX Symposium on Operating Systems*

Design and Implementation, OSDI 2021, July 14-16, 2021. USENIX Association, 113–128. https://www.usenix.org/conference/osdi21/presentation/hwang

[24] Lokesh N. Jaliminche, Chandranil Nil Chakraborttii, Changho Choi, and Heiner Litz. 2023. Enabling Multi-tenancy on SSDs with Accurate IO Interference Modeling. In *Proceedings of the 2023 ACM Symposium on Cloud Computing (SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 216–232. https://doi.org/10.1145/3620678.3624657

[25] Jens Axboe. Accessed: 2024-06-18. Fio. https://github.com/axboe/fio.

[26] Kanchan Joshi, Anuj Gupta, Javier Gonz´lez, Ankit Kumar, Krishna Kanth Reddy, Arun George, Simon Lund, and Jens Axboe. 2024. I/O Passthru: Upstreaming a Flexible and Efficient I/O Path in Linux. In *Proceedings of the 22nd USENIX Conference on File and Storage Technologies (FAST '24)*. USENIX Association, USA, Article 7, 16 pages. https://www.usenix.org/conference/fast24/presentation/joshi

[27] Myoungsoo Jung and Mahmut Kandemir. 2013. Revisiting Widely Held SSD Expectations and Rethinking System-Level Implications. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '13)*. Association for Computing Machinery, New York, NY, USA, 203–216. https://doi.org/10.1145/2465529.2465548

[28] Siu Jung, Seungjin Lee, Jungwook Han, and Youngjae Kim. 2023. Preemptive Zone Reset Design within Zoned Namespace SSD Firmware. *Electronics* 12, 4 (2023). https://doi.org/10.3390/electronics12040798

[29] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-Streamed Solid-State Drive. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'14)*. USENIX Association, USA, 13. https://www.usenix.org/conference/hotstorage14/workshop-program/presentation/kang

[30] Chulbum Kim, Doo-Hyun Kim, Woopyo Jeong, Hyun-Jin Kim, Il Han Park, Hyun-Wook Park, JongHoon Lee, JiYoon Park, Yang-Lo Ahn, Ji Young Lee, Seung-Bum Kim, Hyunjun Yoon, Jae Doeg Yu, Nayoung Choi, NaHyun Kim, Hwajun Jang, JongHoon Park, Seunghwan Song, YongHa Park, Jinbae Bang, Sanggi Hong, Youngdon Choi, Moo-Sung Kim, Hyunggon Kim, Pansuk Kwak, Jeong-Don Ihm, Dae Seok Byeon, Jin-Yub Lee, Ki-Tae Park, and Kye-Hyun Kyung. 2018. A 512-Gb 3-b/Cell 64-Stacked WL 3-D-NAND Flash Memory. *IEEE Journal of Solid-State Circuits* 53, 1 (2018), 124–133. https://doi.org/10.1109/JSSC.2017.2731813

[31] Hyungjin Kim and Seokin Hong. 2023. Why Address Translation Matter?: Analyzing Page Access Patterns in NAND Flash-Based SSDs. In *2023 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*. 1–5. https://doi.org/10.1109/ITC-CSCC58803.2023.10212538

[32] Jieun Kim, Dohyun Kim, and Youjip Won. 2022. Fair I/O Scheduler for Alleviating Read/Write Interference by Forced Unit Access in Flash Memory. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 86–92. https://doi.org/10.1145/3538643.3539753

[33] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs Through OPS Isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*. USENIX Association, USA, 183–189. https://www.usenix.org/conference/fast15/technical-sessions/presentation/kim_jaeho

[34] Jaeho Kim, Kwanghyun Lim, Young-Don Jung, Sungjin Lee, Changwoo Min, and Sam H. Noh. 2019. Alleviating Grbage Collection Interference Through Spatial Separation in All Flash Arrays. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19)*. USENIX Association, USA, 799–812. https://www.usenix.org/conference/atc19/presentation/kim-jaeho

[35] Sang-Hoon Kim, Jaehoon Shim, Euidong Lee, Seongyeop Jeong, Ilkueon Kang, and Jin-Soo Kim. 2023. NVMeVirt: A Versatile Software-Defined Virtual NVMe Device. In *Proceedings of the 21st USENIX Conference on File and Storage Technologies (FAST'23)*. USENIX Association, USA, Article 24, 15 pages. https://www.usenix.org/conference/fast23/presentation/kim-sang-hoon

[36] Changman Lee, Dongho Sim, Joo-Young Hwang, and Sangyeun Cho. 2015. F2FS: A New File System for Flash Storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*. USENIX Association, USA, 273–286. https://www.usenix.org/conference/fast15/technical-sessions/presentation/lee

[37] Euidong Lee, Ikjoon Son, and Jin-Soo Kim. 2023. An Efficient Order-Preserving Recovery for F2FS with ZNS SSD. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23)*. Association for Computing Machinery, New York, NY, USA, 116–122. https://doi.org/10.1145/3599691.3603416

[38] 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 the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 93–99. https://doi.org/10.1145/3538643.3539743

[39] Manjong Lee, Jonggyu Park, and Young Ik Eom. 2023. An Efficient F2FS GC Scheme for Improving I/O Latency of Foreground Applications. In *IEEE International Conference on Consumer Electronics, ICCE 2023, Las Vegas, NV, USA, January 6-8, 2023*. IEEE, 1–3. https://doi.org/10.1109/ICCE56470.2023.10043469

[40] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminatahan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. 2018. The Case of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*. USENIX Association, USA, 83–90. https://www.usenix.org/conference/fast18/presentation/li

[41] Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. 2021. IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 263–279. https://doi.org/10.1145/3477132.3483573

[42] Nanqinqin Li, Mingzhe Hao, Huaicheng Li, Xing Lin, Tim Emami, and Haryadi S. Gunawi. 2022. Fantastic SSD Internals and How to Learn and Use Them. In *Proceedings of the 15th ACM International Conference on Systems and Storage (SYSTOR '22)*. Association for Computing Machinery, New York, NY, USA, 72–84. https://doi.org/10.1145/3534056.3534940

[43] Renping Liu, Zhenhua Tan, Yan Shen, Linbo Long, and Duo Liu. 2024. Fair-ZNS: Enhancing Fairness in ZNS SSDs Through Self-Balancing I/O Scheduling. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 43, 7 (2024), 2012–2022. https://doi.org/10.1109/TCAD.2022.3232997

[44] Linbo Long, Shuiyong He, Jingcheng Shen, Renping Liu, Zhenhua Tan, Congming Gao, Duo Liu, Kan Zhong, and Yi Jiang. 2024. WA-Zone: Wear-Aware Zone Management Optimization for LSM-Tree on ZNS SSDs. *ACM Trans. Archit. Code Optim.* 21, 1, Article 16 (jan 2024), 23 pages. https://doi.org/10.1145/3637488

[45] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. 2024. eZNS: Elastic Zoned Namespace for Enhanced Performance Isolation and Device Utilization. *ACM Trans. Storage* 20, 3, Article 16 (jun 2024), 41 pages. https://doi.org/10.1145/3653716

[46] Gijun Oh, Junseok Yang, and Sungyong Ahn. 2021. Efficient Key-Value Data Placement for ZNS SSD. *Applied Sciences* 11, 24 (2021).

https://doi.org/10.3390/app112411842

[47] Myounghoon Oh, Seehwan Yoo, Jongmoo Choi, Jeongsu Park, and Chang-Eun Choi. 2023. ZenFS+: Nurturing Performance and Isolation to ZenFS. *IEEE Access* 11 (2023), 26344–26357. https://doi.org/10.1109/ACCESS.2023.3257354

[48] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. 2014. SDF: Software-Defined Flash for Web-Scale Internet Storage Systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 471–484. https://doi.org/10.1145/2541940.2541959

[49] Stan Park and Kai Shen. 2012. FIOS: A Fair, Efficient Flash I/O Scheduler. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. USENIX Association, USA, 13. https://www.usenix.org/conference/fast12/fios-fair-efficient-flash-io-scheduler

[50] Devashish R. Purandare, Sam Schmidt, and Ethan L. Miller. 2023. Persimmon: An Append-Only ZNS-First Filesystem. In *41st IEEE International Conference on Computer Design, ICCD 2023, Washington, DC, USA, November 6-8, 2023*. IEEE, 308–315. https://doi.org/10.1109/ICCD58817.2023.00054

[51] Wenjie Qi, Zhipeng Tan, Jicheng Shao, Lihua Yang, and Yang Xiao. 2022. InDeF: An Advanced Defragmenter Supporting Migration Offloading on ZNS SSD. In *IEEE 40th International Conference on Computer Design, ICCD 2022, Olympic Valley, CA, USA, October 23-26, 2022*. IEEE, 307–314. https://doi.org/10.1109/ICCD56317.2022.00052

[52] Zebin Ren, Krijn Doekemeijer, and Animesh Trivedi. 2024. A Systematic Configuration Space Exploration of the Linux Kyber I/O Scheduler. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion)*. Association for Computing Machinery, New York, NY, USA, 167–173. https://doi.org/10.1145/3629527.3651416

[53] Samsung. Accessed: 2024-08-21. Samsung 990 PRO PCIe® 4.0 NVMe® SSD 1TB. https://www.samsung.com/us/computing/memory-storage/solid-state-drives/990-pro-pcie–4-0-nvme–ssd-1tb-mz-v9p1t0b-am/.

[54] Dongjoo Seo, Ping-Xiang Chen, Huaicheng Li, Matias Bjørling, and Nikil Dutt. 2023. Is Garbage Collection Overhead Gone? Case study of F2FS on ZNS SSDs. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23)*. Association for Computing Machinery, New York, NY, USA, 102–108. https://doi.org/10.1145/3599691.3603409

[55] Jingcheng Shen, Lang Yang, Linbo Long, Renping Liu, Zhenhua Tan, Congming Gao, and Yi Jiang. 2024. Overlapping Aware Zone Allocation for LSM Tree-Based Store on ZNS SSDs. In *Proceedings of the 29th Asia and South Pacific Design Automation Conference (ASPDAC '24)*. IEEE Press, 448–453. https://doi.org/10.1109/ASP-DAC58780.2024.10473944

[56] Kai Shen and Stan Park. 2013. FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC'13)*. USENIX Association, USA, 67–78. https://www.usenix.org/conference/atc13/technical-sessions/presentation/shen

[57] 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. https://doi.org/10.1109/ICCD53106.2021.00046

[58] Hojin Shin, Myunghoon Oh, Gunhee Choi, and Jongmoo Choi. 2020. Exploring Performance Characteristics of ZNS SSDs: Observation and Implication. In *9th Non-Volatile Memory Systems and Applications Symposium, NVMSA 2020, Seoul, South Korea, August 19-21, 2020*. IEEE, 1–5. https://doi.org/10.1109/NVMSA51238.2020.9188086

[59] Inho Song, Myounghoon Oh, Bryan Suk Joon Kim, Seehwan Yoo, Jaedong Lee, and Jongmoo Choi. 2023. ConfZNS: A Novel Emulator for Exploring Design Space of ZNS SSDs. In *Proceedings of the 16th ACM International Conference on Systems and Storage (SYSTOR '23)*. Association for Computing Machinery, New York, NY, USA, 71–82. https://doi.org/10.1145/3579370.3594772

[60] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQsim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*. USENIX Association, USA, 49–65. https://www.usenix.org/conference/fast18/presentation/tavakkol

[61] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie S. Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gómez-Luna, and Onur Mutlu. 2018. FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18)*. IEEE Press, 397–410. https://doi.org/10.1109/ISCA.2018.00041

[62] Nick Tehrany, Krijn Doekemeijer, and Animesh Trivedi. 2024. zns-tools: An eBPF-powered, Cross-Layer Storage Profiling Tool for NVMe ZNS SSDs. In *Proceedings of the 4th Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '24)*. Association for Computing Machinery, New York, NY, USA, 23–32. https://doi.org/10.1145/3642963.3652205

[63] Nick Tehrany and Animesh Trivedi. 2022. Understanding NVMe Zoned Namespace (ZNS) Flash SSD Storage Devices. *CoRR* abs/2206.01547 (2022). https://doi.org/10.48550/arXiv.2206.01547 arXiv:2206.01547

[64] Shucheng Wang, Kaiye Zhou, Zhandong Guo, Qiang Cao, Jun Xu, and Jie Yao. 2024. SIndex: An SSD-based Large-scale Indexing with Deterministic Latency for Cloud Block Storage. In *Proceedings of the 53rd International Conference on Parallel Processing (ICPP '24)*. Association for Computing Machinery, New York, NY, USA, 1237–1246. https://doi.org/10.1145/3673038.3673041

[65] Western Digital. Accessed: 2024-06-18. ZenFS: RocksDB Storage Backend for ZNS SSDs and SMR HDDs. https://github.com/westerndigitalcorporation/zenfs.

[66] Jiwon Woo, Minwoo Ahn, Gyusun Lee, and Jinkyu Jeong. 2021. D2FQ: Device-Direct Fair Queueing for NVMe SSDs. In *19th USENIX Conference on File and Storage Technologies, FAST 2021, February 23-25, 2021*. USENIX Association, 403–415. https://www.usenix.org/conference/fast21/presentation/woo

[67] Miguel G. Xavier, Israel C. De Oliveira, Fábio D. Rossi, Robson D. Dos Passos, Kassiano J. Matteussi, and César A. F. De Rose. 2015. A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds. In *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015, Turku, Finland, March 4-6, 2015*. IEEE Computer Society, 253–260. https://doi.org/10.1109/PDP.2015.67

[68] Junseok Yang, Seokjun Lee, and Sungyong Ahn. 2022. Selective Power-Loss-Protection Method for Write Buffer in ZNS SSDs. *Electronics* 11, 7 (2022). https://doi.org/10.3390/electronics11071086

[69] Weilin Zhu and Wei Tong. 2023. Turn Waste Into Wealth: Alleviating Read/Write Interference in ZNS SSDs. In *41st IEEE International Conference on Computer Design, ICCD 2023, Washington, DC, USA, November 6-8, 2023*. IEEE, 316–319. https://doi.org/10.1109/ICCD58817.2023.00055