Don't Be a Blockhead: Zoned Namespaces Make Work on Conventional SSDs Obsolete

Theano Stavrinos
Princeton University

Daniel S. Berger
Microsoft Research

Ethan Katz-Bassett
Columbia University

Wyatt Lloyd
Princeton University

ABSTRACT

Research on flash devices almost exclusively focuses on conventional SSDs, which expose a block interface. Industry, however, has standardized and is adopting Zoned Namespaces (ZNS) SSDs, which offer a new storage interface that dominates conventional SSDs. Continued research on conventional SSDs is thus a missed opportunity to unlock a step-change improvement in system performance by building on ZNS SSDs. We argue for an immediate and complete shift in research to ZNS SSDs and discuss research directions.

ACM Reference Format:
Theano Stavrinos, Daniel S. Berger, Ethan Katz-Bassett, and Wyatt Lloyd. 2021. Don't Be a Blockhead: Zoned Namespaces Make Work on Conventional SSDs Obsolete. In *Workshop on Hot Topics in Operating Systems (HotOS '21), May 31–June 2, 2021, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3458336.3465300

1 INTRODUCTION

Cloud and datacenter services demand low-latency access to massive datasets. Flash-based solid-state drives (SSDs) have become critical to meeting these needs by offering orders-of-magnitude lower latency and higher I/O throughput than hard disk drives (HDDs), while providing more capacity at lower cost than DRAM. As a consequence, SSDs are used for latency-sensitive applications, e.g., caching [6, 11, 41, 47, 49] and key-value stores [1, 28, 30]. SSDs are now so common that even latency-insensitive applications like general-purpose filesystems [44] are built on top of SSDs, and cloud providers do not offer HDDs as part of general-purpose VMs (and typically use only SSDs in general-purpose server blades).

HotOS '21, May 31–June 2, 2021, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-8438-4/21/05.
https://doi.org/10.1145/3458336.3465300

Unfortunately, flash storage has significant physical limitations. Flash cells in an *erasure block* can only be re-written after the block is fully erased. Flash cells wear out with each write-and-erase cycle, eventually losing the ability to reliably store data, limiting cell *endurance*.

In *conventional* SSDs, flash cells and their peculiarities are hidden behind the traditional block interface. This interface is implemented via a sophisticated piece of firmware on the SSD, the *flash translation layer (FTL)* (§2). The block interface exposes to the host a flat address space that can be randomly written at a page granularity (typically 4 KB), similar to HDDs. This interface is familiar to application developers and is supported by major operating systems. However, because flash cells cannot be overwritten and must be erased at the erasure block granularity (typically several megabytes), random writes force the FTL to implement garbage collection to reclaim space from old data that was overwritten in the logical address space. Garbage collection copies forward valid data into *overprovisioned* (spare) flash space before erasing an erasure block. This causes *write amplification*, where a byte that is written to the logical address space once is written physically on flash multiple times. Write amplification reduces device lifetime by using excess write-and-erase cycles. Placing data together that will be invalidated around the same time is key for avoiding write amplification. Unfortunately, the FTL does not have access to the application-level information needed for such data placement, and applications have limited control over how the FTL arranges data on the device.

Significant research effort has gone toward managing the ill effects of conventional SSDs' block interface. This includes much work on managing the performance reduction and unpredictability caused by garbage collection and other FTL tasks [19, 29, 55, 56]. Prior work has reverse-engineered FTLs to find access patterns that work best with the FTL's internal operations [20, 62]. Systems also frequently throttle flash writes to extend the lifetimes of their flash devices as their workloads cause high write amplification [6, 16, 25].

This paper argues that the systems community should stop researching conventional SSDs today. Our efforts should shift to Zoned Namespaces (ZNS) SSDs [52]. ZNS is a new SSD interface that dominates the conventional block interface in

nearly every way (§2). The benefits of ZNS come from the interface matching how data is written to physical flash while abstracting the messy details of flash hardware. A ZNS SSD is divided into logical regions called *zones*. Writes can go to any zone but must be sequential within a zone. *ZNS is a reality*: the interface specification was added to the NVMe standard in 2020, ZNS SSDs are available from several vendors [38, 39, 51], and large cloud providers are adopting them. On the software side, a growing list of ZNS-compatible filesystems, key-value stores, and development frameworks is removing barriers to adoption (§2.5).

Because the ZNS interface more closely matches how data is written, the FTL is thinner: compared to conventional FTLs, it does coarser-grained address translation (i.e., at the erasure block granularity instead of the much smaller page granularity), and it does not do garbage collection. As a result, performance variability and other impacts of garbage collection are eliminated; performance modeling and reverse-engineering the FTL are unnecessary. Hosts control write amplification and can make application-aware data placement and I/O scheduling decisions. Devices are cheaper because overprovisioned flash capacity for garbage collection is unnecessary, and a fraction of the DRAM is required for address translation. Some ZNS use cases may need host resources for tasks traditionally carried out by the FTL. Crucially, with ZNS application developers can choose how many system resources (if any) to devote to flash-related tasks (§2.3). In conventional SSDs, DRAM and overprovisioned flash capacity are fixed costs, whether they are needed or not.

We conduct a survey of recent systems literature on SSDs and find that only 18% of papers will not be affected by the shift to ZNS SSDs; 23% of papers address problems that are simplified or solved by ZNS. The remaining 59% of papers will be affected or need revisiting once industry inevitably shifts to using ZNS SSDs for their cost and performance benefits (§3).

Instead of conventional SSD problems that are obsolete, research going forward should focus on making our systems cheaper, faster, and more efficient with ZNS SSDs. We propose several broad research directions (§4) that will improve system performance and target the specific limitations and challenges introduced by ZNS SSDs.

2 ZNS DEVICES DOMINATE!

Zoned Namespaces SSDs dominate conventional devices in nearly every way: they are cheaper, expose a more useful abstraction to hosts, and may get better performance.

2.1 A Quick Flash Primer

Flash architecture. Flash is composed of NAND cells. A cell can store one (SLC), two (MLC), three (TLC), four (QLC), or five (PLC) bits, depending on the number of programmed and retained voltage levels.

To achieve high capacity, flash cells are arranged in three dimensions: two-dimensionally in arrays, and then stacked. To achieve high throughput, read and write operations exploit parallelism across thousands of cells.

This architecture forms a deep multi-layer hierarchy. Multiple cells form a page, which includes significant redundancy to store error-correcting codes. Multiple pages form an erasure block, which in turn form planes. Multiple planes finally form a channel (or die).

Reading and writing flash. Flash exposes a read/write/erase interface. Reads can happen at page granularity (often 4 KB). Before a page can be written (programmed), it must be erased. Erasing takes several times longer than programming (~6× for TLC) [54]. To hide this latency, erasures are batched at block granularity (tens of MB). Within an erasure block, pages are programmed sequentially until no empty pages remain.

Multiple read/write operations are typically scheduled to happen in parallel across multiple planes in each channel. Programming multi-level cells involves significantly more steps, and manufacturers often implement variants of this geometry to further improve parallelism, which increases erasure block sizes to hundreds of megabytes [9, 12].

Conventional SSDs. In a conventional SSD, the flash translation layer (FTL) hides the intricacies of device geometry and the presence of erasure blocks. If adjacent logical pages are written at different times, they may be mapped to different block offsets, or even different blocks and planes. The FTL is responsible for:

- translating each logical address into the hierarchy of channels, planes, erasure blocks, and pages.
- garbage collection: when erasing an erasure block that contains a mixture of *valid* and *invalid* pages (i.e., pages that have been logically overwritten on flash), the FTL copies forward the valid pages into an erasure block with free pages remaining, then erases the old block.
- storing FTL data structures durably and in a consistent state to prepare for power-off events.
- wear leveling: ensuring erasure blocks wear as evenly as possible by balancing erasures across all blocks.

Zoned Namespaces SSDs. ZNS SSDs partition the address space into *zones*, which behave similarly to erasure blocks: a zone can only be written sequentially. When full, it must be reset (erased) before it can accept new data. The current write position is tracked with a *write pointer*.

Zones can be in six different *states*: empty, open, closed, full, read-only, and offline. Zones start empty, are opened and written to until full, and then are reset to the empty state, after which they can be opened again. Only a limited number of

zones can be active at once, since each active zone consumes resources (from, e.g., write buffers [52]). However, all zones can be in read-only mode at the same time. Flash cell failures are handled transparently by decreasing the length of a zone after a reset, or by marking a zone as offline.

Zones are at least as large as erasure blocks. For instance, a device evaluated in recent work uses 1GB zones and supports 14 active zones [10].

2.2 ZNS Costs Less per Gigabyte

ZNS SSDs need fewer dedicated resources on each device, which significantly reduces their cost.

Less on-board DRAM. In conventional SSDs, the FTL keeps the address mapping table (logical to physical) and garbage collection metadata in the device's on-board DRAM. In ZNS SSDs, the FTL maintains address translations at the granularity of zones, which requires minimal DRAM.[1]

To estimate DRAM overhead, we assume pages are 4 KB. An optimized mapping table in a conventional SSD requires about 4 bytes per page [14]. This is around 1 GB of on-board DRAM per TB of flash on current devices [7]. In ZNS SSDs, the FTL maps zones to erasure blocks. Assuming a similar 4-byte overhead per block and 16 MB erasure blocks, it requires only ~256 KB of on-board DRAM. Constructing a more complex interface over ZNS requires some host resources (§2.3).

Less overprovisioning. Conventional SSDs reserve a large fraction of flash cells as spare capacity. This overprovisioning is typically 7–28 of the usable capacity% [10, 32]. The FTL uses this spare capacity to reduce garbage collection overheads. Rather than needing to reclaim space immediately when overwriting a logical address, the SSD can simply write the new data to a clean flash page and update the address mapping. Delaying garbage collection also increases the likelihood that data in an erasure block will be invalidated before the block's space needs to be reclaimed. In our lab experiments with random write workloads and a variable overprovisioning factor, the write amplification (additional bytes written) from garbage collection improves from 15× with no overprovisioning to about 2.5× with ~25% overprovisioning.

Overprovisioning inflates SSD prices, as flash cells are the most costly part of a device [46]. ZNS SSDs do not do garbage collection, since the erasure blocks in a zone are always completely invalidated and erased when the zone is reset. The host therefore has access to nearly all of the flash capacity (some is reserved to replace bad flash blocks). Although some applications will do host-side garbage collection, moving this responsibility to the host means that system resources can be allocated according to an application's needs, rather than being fixed for all applications.

2.3 ZNS Is a More Useful Abstraction

The block interface exposed by conventional SSDs is familiar to anyone who has developed software that reads or writes to a hard disk drive. However, the block interface abstracts too much of flash's write constraints. This makes it extremely difficult for applications to tune their I/O patterns to maximize performance and device endurance. The interface exposed by ZNS SSDs hits a sweet spot that balances being easy to use and faithfully representing how the device handles data. Writers may write to any zone, but each zone must be written sequentially. This thin interface gives applications more control over data placement on flash, and hence more control over write amplification.

The zoned interface is amenable to building other abstractions on top. For example, it was straightforward to implement the block interface on the host using ZNS SSDs (§2.5). This task is aided by the *simple copy* command in the NVMe standard [36], which allows the host to issue device controller-managed data copy operations. With this command, copying forward valid data before erasing a zone does not use any PCIe bandwidth, enabling performance comparable to conventional SSDs.

Building an abstraction over the device's interface shifts some computational and DRAM burden to the host, one of the few drawbacks of ZNS SSDs. However, buying a large DRAM DIMM for a host is cheaper than multiple small embedded DRAM chips in each SSD.[2] Additionally, since the host has insight into application specifics, it can manage garbage collection and other data management tasks better than an on-board FTL (§4.1). Hosts can choose whether to pay the additional host resource cost to construct a different interface.

ZNS is a refinement of two existing abstractions. Open-channel SSDs expose device geometry to the host, enabling fine-grained control over data placement [9, 37]. Zoned Namespaces standardizes a slightly higher-level interface based on lessons learned from Open-Channel SSDs [7]. The *multi-stream writes* NVMe directive is conceptually similar to ZNS [34]. Hosts label related writes with the same stream ID, and the device writes each stream to its own set of erasure blocks. Multi-streams are a workaround to hosts' limited control over data placement in conventional SSDs; the high hardware costs of conventional devices remains.

[1] A few DRAM-less conventional SSDs exist, which store the mapping data in host DRAM or on-board flash. However, they have not gained momentum in datacenters, as they lack the performance and functionality of ZNS SSDs.

[2] Using end-user prices as a proxy, we find that a 1GB DIMM costs more than twice as much per GB as 16-32GB DIMMs.

2.4 ZNS May Get Better Performance

The key performance strategies used by FTLs in conventional SSDs—buffering writes and placing writes strategically to leverage parallelism—are equally available to a ZNS device. While it requires further research to validate, ZNS SSDs should be able to accomplish these tasks as well as and often better than an on-board FTL (§4.1). It is clear, though, that information about applications is the key bottleneck for near-optimal garbage collection, and conventional FTLs do not have this information [43]. The host, on the other hand, is empowered to make application-aware data placement decisions. Because the host can control what data will be erased together by writing it to the same zone, it has fine-grained control over write amplification. The host may be able to significantly reduce write amplification by grouping data into zones based on when it expects the data will expire.

The host can also leverage application knowledge to make better scheduling decisions. In conventional SSDs, garbage collection increases tail latency by interfering with I/O. Garbage collection also makes performance unpredictable because the FTL uses opaque internal algorithms to schedule it [19, 55]. Hosts explicitly reclaim space on ZNS SSDs, increasing performance predictability and reducing read tail latency by allowing hosts to schedule garbage collection around I/O.

Early ZNS SSD experiments highlight their performance. Western Digital reports 60% lower average read latency and $3\times$ higher throughput in benchmarks [51]. Western Digital also reports $2-4\times$ lower read tail latency and $2\times$ higher write throughput for RocksDB over ZNS [10]. CMU researchers showed that RocksDB's write amplification drops from $5\times$ to $1.2\times$ on ZNS SSDs [3]. IBM reports $22\times$ lower tail latencies and 65% higher application throughput [39].

2.5 ZNS SSD Adoption

ZNS devices are already on the market from some vendors [38, 39, 51]. All major SSD manufacturers have working ZNS prototypes, with most being tested at hyperscalers for months.

One hyperscaler shared with us that ZNS SSDs are a crucial building block for deploying QLC flash and realizing significant cost savings in the future. In preparing for this deployment, significant investments are being made to make backend software compatible with ZNS. A second hyperscaler publicly shared performance results from porting their storage software stack to ZNS SSDs [39].

In the open-source world, Linux supports two native zoned filesystems (ZoneFS and F2FS) [53]. Additionally, mainstream filesystems Btrfs and ext4 are close to running natively on ZNS SSDs. Until then (and for XFS in the near future), as of version 4.14, Linux provides the `dm-zoned` device mapper which emulates a standard block device on top of any ZNS SSD [53]. Additionally, large applications like Ceph [4] and RocksDB [3, 10] have been ported to natively use ZNS SSDs, while realizing significant performance gains. The Storage Performance Development Kit (SPDK), a framework for developing kernel-bypass storage applications, supports ZNS as of version 20.10 [50].

3 REEVALUATING THE FOCUS OF SSD R&D

To understand the impact that this shift in technology will have on future systems research, we survey recent work on flash and SSDs published at OS and storage conferences. Specifically, we are interested in whether we would still carry out this research in a future where ZNS devices are dominant. To be clear, we do not question the papers' value or impact. These works solve real problems with conventional SSDs, and until now there was no alternative, similarly-priced technology that avoided those problems. However, ZNS changes that, and we must reevaluate our research agenda accordingly or we needlessly delay using this new technology to its fullest potential. Our survey thus aims to quantify the effort that *would* be misdirected if the systems community continued working on conventional SSDs.

We survey papers published in the last five years at FAST, OSDI, SOSP, and MSST. We collected 465 papers in total and narrowed them down to 104 papers where flash-based SSDs are a prominent part of the research or the system implementation; we classified these manually. This is not an exhaustive survey; our goal is to get a broad-strokes sense of the targets of recent systems research. We omit papers on ultra low-latency SSDs (e.g., Samsung Z-NAND [40]) and non-volatile memory (e.g., 3D XPoint [33]) since they have different systems implications than traditional flash SSDs.

We define four broad categories for the papers (bold indicates the column header in Table 1):

- **Simpl**ified/Solved: the paper's main problem is solved or simplified with ZNS SSDs. E.g., building the FTL for a flash simulator [26] or improving garbage collection [60].
- **Appr**oach: the paper's approach to solving the problem may change with ZNS. E.g., the system implementation may change [15].
- **Res**ults: the results of the research or evaluation may change. E.g., the system performance may change [18], or the findings of a measurement study may be different [17].
- **Orth**ogonal: the problem addressed in the paper is orthogonal to ZNS. E.g., it proposes a low-level security technique for flash [61].

The taxonomy is shown in Table 1. We find that 23% of SSD papers focus on problems ZNS either simplifies or solves. Papers in this category focus on mitigating the negative performance effects of garbage collection [19, 21, 55]

Venue	# Pubs.	Simpl	Appr	Res	Orth
FAST	126	9	8	23	8
OSDI	164	3	0	4	0
SOSP	77	2	2	2	0
MSST	98	10	7	16	10
Total	465	24	17	45	18

Table 1: Impact of ZNS adoption on existing work on flash-based SSDs. Columns are counts. *# Pubs.* indicates the total publications in the venue over the last 5 years.

or on managing write amplification to preserve device endurance [13, 23, 29]. Other papers reverse-engineer or redesign the FTL to reduce performance variability [24, 58]; this is much simpler in ZNS SSDs because the FTL is simpler.

An additional 59% of SSD papers would need to change their approach or revisit results, as they will change with ZNS SSDs. Many systems with a significant flash component would need to be redesigned for ZNS [6, 18, 45]. Systems that rely heavily on flash may show different endurance or performance characteristics after porting to ZNS and would need to be re-evaluated. Performance and reliability measurement studies that characterize device behavior in the field or under extreme workloads would need to be re-run to properly characterize systems that include ZNS SSDs [17, 31, 42].

4 RESEARCH AGENDA

Now that ZNS has been standardized and devices are commercially available, it is time to revisit many of the questions we asked about conventional SSDs, this time with ZNS, and ask new questions about how best to integrate this technology into our systems.

This section first describes several opportunities for improving performance that arise from co-locating garbage collection with applications on the host. Then it describes several challenges introduced by ZNS's new interface.

4.1 Improving Performance

How can application-level information improve zone management? Information about applications and their access patterns is precious when allocating writes to erasure blocks and zones. Garbage collection overheads are minimal if most of the data that is written to an erasure block expires at the same time. Software can often make educated guesses about data expiration, e.g., pages in the same file are more likely to get deleted or overwritten together than pages in different files. Files created at similar times are also more likely to expire together (e.g., intermediate files in analytics workloads). At an even higher level, sets of files created by the same application, container, or virtual machine are more likely to expire at the same time, compared to those with different creators and owners (e.g., when an application exits or when a virtual machine migrates or shuts down).

This crucial information has never been available to conventional SSDs. This information barrier has historically capped FTLs to suboptimal performance, even with near-optimal garbage collection algorithms [43]. The filesystem has this information readily available and can use it with ZNS SSDs; however, current Linux kernel filesystems for ZNS SSDs (e.g., F2FS) do not yet use this information. Consequently, this is a rich topic for exploration. How much can filesystem knowledge (owners, creators, timestamps) reduce write amplification? Beyond the filesystem, how much does application-specific information further reduce overheads? Given the additional information, how does the theoretically optimal garbage collection algorithm change?

How should applications interact with zones? Section 2.3 outlines the interfaces through which applications can access ZNS SSDs. Each of these interfaces makes performance and usability tradeoffs: raw zoned storage access offers the most control over I/O and data placement; filesystems and key-value stores offer less control but are easy to use (e.g., many filesystems handle metadata management and data integrity). There is variety even within a class of interfaces. For instance, F2FS [22] is a fully-featured, POSIX-compliant filesystem, while ZoneFS [2] treats zones as files with the same restrictions as zones themselves. Raw zoned storage may go through the kernel [52] or not [57]. Finally, individual applications have configurations that can be adjusted for performance (e.g., the active zone count in ZNS-enabled RocksDB [10]).

Flash's limited endurance means there is a cost to using the wrong interface beyond just performance, as we see with conventional SSDs and the block interface (§2). Zoned storage devices (e.g., SMR HDDs) existed before ZNS, but they do not suffer the same endurance limitation as flash devices. Interacting with zoned flash devices in a way that maximizes I/O performance, usability, *and* device endurance has thus not been fully explored. In general, will applications prefer to use the zoned interface, a filesystem, or some other API? What is the best way to make the endurance, performance, and usability tradeoffs explicit for different applications and system settings? How can the process of experimenting with different interfaces (and interface configurations) be made as pain-free as possible?

What is the best approach to I/O scheduling with host-driven device management? In conventional SSDs, the timing of garbage collection was known neither to the OS nor applications. This led to complex workarounds, e.g., predicting the latency of an I/O request to a conventional SSD and using multiple SSDs to hedge against slow requests [19]. With ZNS SSDs, the performance of individual zones is independent of one another, beyond sharing bandwidth. (Bandwidth is

already not a limitation with current second-generation PCIe4 SSDs and will be even less of a limitation with PCIe5 products available in the near future.) Thus, the host is in full control and can precisely schedule zone erasures and maintenance operations. This flexibility enables new policies to prioritize one goal over the other, e.g., read latency over write latency and write amplification. Additionally, these policies can differ across sets of zones, enabling efficient co-location of multiple applications or tenants. What are the interfaces for application developers to communicate these goals and preferred trade-offs to applications or the OS? How do developers choose these trade-offs in a principled fashion?

How can we best exploit transparent data placement? In conventional SSDs, applications have to rely on tricks to force the FTL to place data as intended. For example, large-scale flash caching applications [6, 49] maintain several buckets of objects, where each bucket should be written to the same erasure block. In conventional SSDs, concurrent write operations are interleaved on an erasure block even if they are far apart in the logical address space. Applications have evolved to use DRAM as a buffer to coalesce many writes into one very large write. With ZNS SSDs, these buffers are no longer necessary. How can we identify and modify these applications at scale to reclaim the wasted DRAM? Is there a common abstraction that works for such applications?

4.2 Managing Limitations

How should hosts manage active zone limits? Current ZNS SSDs limit the number of zones that can be writable at the same time (see Section 2). This limitation is not a problem for existing kernel filesystem implementations, which already achieve enough parallelism to fully use available flash bandwidth. However, it becomes a problem when a ZNS SSD is divided among multiple applications that bypass the kernel. A simple strategy is to assign a fixed number of zones to each application together with a fixed active zone budget. However, this approach does not scale for typical bursty workloads as it does not allow multiplexing of this scarce resource. Is there a good strategy for dynamically assigning zones on demand? Assuming manufacturers will make devices with many more zones given the modest on-device resource requirements, what is a sufficient number of zones?

Are there workloads that perform worse on ZNS SSDs than on conventional SSDs? One well-known workload challenge with ZNS SSDs, which was discovered in prototypes after the specification was finalized, is that a zone's write pointer can suffer from lock contention. It is a problem for multi-writer workloads where writes are concentrated in a single zone, such as persistent queues and append-only data structures. This happens because the specification assigns responsibility to move the write pointer to host-side software. An addition to the ZNS specification resolves this problem: the `append` command, which does not specify an offset and allows the device to serialize concurrent writes to the same zone [8, 35]. Given the novelty of the ZNS interface and limited hardware deployments so far, it remains an open question if there are other workloads that will perform worse over ZNS than over conventional SSDs. Can we systematically test representative and synthetic workloads to discover if any perform worse over ZNS? Can their performance be rectified with extensions to the ZNS specification?

How do ZNS SSDs fit with recent trends to offload I/O tasks from the host to dedicated hardware? At the same time that hyperscalers are embracing ZNS SSDs, which shift responsibilities to the host, they are also offloading I/O processing from host CPUs to dedicated hardware: to ASICs at AWS [5], to ARM SoCs at Microsoft [27], and to FPGAs at Alibaba [59]. This apparent contradiction in system design philosophies calls for academic scrutiny. Similar to Facebook's Accelerometer for computation offloading [48], we envision research on how to decide which parts of the hardware stack should be responsible for which functionality on the I/O control and datapaths.

5 CONCLUSION

While much SSD research has focused on the problems with conventional SSDs, industry has moved ahead, standardizing and beginning to adopt Zoned Namespaces. This inflection point opens up exciting and impactful research directions around how to use these emerging devices to their full potential. It also points to a problem facing the systems community: a lot of research is intended for settings such as large cloud providers, but often only those in industry or with close industry connections have privileged access to emerging technologies and problems. This disadvantages researchers without such connections, but it also harms the field as a whole. Progress is stymied when access to relevant problems is restricted and when the diversity of thinkers attacking the most pressing problems is curtailed. This era of online workshops and conferences lowers barriers to participation; we hope organizers will take advantage by encouraging participation from researchers from a range of institutions worldwide and by organizing industry panels as an informal avenue for discussion of emerging trends and challenges.

6 ACKNOWLEDGEMENTS

We thank the anonymous reviewers, Matias Bjørling, and Kai Li for their helpful comments, which substantially improved the paper. Theano Stavrinos is supported by the National Science Foundation grant CNS-1910390.

REFERENCES

[1] RocksDB. https://rocksdb.org/, 2020. Accessed: 2021-1-17.

[2] ZoneFS - Zone filesystem for Zoned block devices. https://www.kernel.org/doc/html/latest/filesystems/zonefs.html, 2020. Accessed: 2021-1-17.

[3] A. Aghayev. *Adopting Zoned Storage in Distributed Storage Systems*. PhD thesis, Carnegie Mellon University, 2020.

[4] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis. The Case for Custom Storage Backends in Distributed Storage Systems. *ACM Transactions on Storage (TOS)*, 16(2):1–31, 2020.

[5] Amazon Web Services. AWS Nitro System. https://aws.amazon.com/ec2/nitro/, 2020. Accessed: 2021-2-3.

[6] B. Berg, D. S. Berger, S. McAllister, I. Grosof, S. Gunasekar, J. Lu, M. Uhlar, J. Carrig, N. Beckmann, M. Harchol-Balter, and G. R. Ganger. The CacheLib Caching Engine: Design and Experiences at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, 2020. USENIX Association.

[7] M. Bjørling. From Open-Channel SSDs to Zoned Namespaces. In *Proceedings of the 2019 Linux Storage and Filesystems Conference (Vault'19)*, 2019. USENIX Association.

[8] M. Bjørling. Zone Append: A New Way of Writing to Zoned Storage. In *Proceedings of the 2020 Linux Storage and Filesystems Conference (Vault'20)*, 2020. USENIX Association.

[9] M. Bjørling, J. González, and P. Bonnet. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*, 2017. USENIX Association.

[10] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. L. Moal, G. Ganger, and G. Amvrosiadis. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC'21)*, 2021. USENIX Association.

[11] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani. TAO: Facebook's Distributed Data Store for the Social Graph. In *Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC'13)*, 2013. USENIX Association.

[12] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai. Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation. In *IEEE International Conference on Computer Design*, pages 123–130, 2013.

[13] Z. Chen and K. Shen. OrderMergeDedup: Efficient, Failure-Consistent Deduplication on Flash. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*, 2016. USENIX Association.

[14] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. A survey of flash translation layer. *Journal of Systems Architecture*, 55 (5-6):332–343, 2009.

[15] J. Cui, W. Wu, X. Zhang, J. Huang, and Y. Wang. Exploiting latency variation for access conflict reduction of NAND flash memory. In *Proceedings of the 32nd International Conference on Massive Storage Systems and Technology (MSST'16)*, 2016. IEEE Computer Society.

[16] A. Eisenman, A. Cidon, E. Pergament, O. Haimovich, R. Stutsman, M. Alizadeh, and S. Katti. Flashield: a Hybrid Key-value Cache that Controls Flash Write Amplification. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI'19)*, 2019. USENIX Association.

[17] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, et al. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*, 2018. USENIX Association.

[18] S. Han, D. Jiang, and J. Xiong. LightKV: A Cross Media Key Value Store with Persistent Memory to Cut Long Tail Latency. In *Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST'20)*, 2020. IEEE Computer Society.

[19] M. Hao, L. Toksoz, N. Li, E. E. Halim, H. Hoffmann, and H. S. Gunawi. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, 2020. USENIX Association.

[20] J. Kim, P. Park, J. Ahn, J. Kim, J. Kim, and J. Kim. SSDcheck: Timely and accurate prediction of irregular behaviors in black-box SSDs. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*, 2018. Association for Computing Machinery (ACM).

[21] J. Kim, K. Lim, Y. Jung, S. Lee, C. Min, and S. H. Noh. Alleviating Garbage Collection Interference Through Spatial Separation in All Flash Arrays. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC'19)*, 2019. USENIX Association.

[22] C. Lee, D. Sim, J. Hwang, and S. Cho. F2FS: A new file system for flash storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*, 2015. USENIX Association.

[23] E. Lee, J. Kim, H. Bahn, and S. H. Noh. Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM. In *Proceedings of the 32nd International Conference on Massive Storage Systems and Technology (MSST'16)*, 2016. IEEE Computer Society.

[24] B. Li, W. Tong, J. Liu, D. Feng, Y. Feng, J. Qin, P. Li, and B. Liu. Maximizing Bandwidth Management FTL Based on Read and Write Asymmetry of Flash Memory. In *Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST'20)*, 2020. IEEE Computer Society.

[25] C. Li, P. Shilane, F. Douglis, and G. Wallace. Pannier: Design and Analysis of a Container-Based Flash Cache for Compound Objects. *ACM Transactions on Storage (TOS)*, 13(3):1–34, 2017.

[26] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi. 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)*, 2018. USENIX Association.

[27] H. Li, M. Hao, S. Novakovic, V. Gogte, S. Govindan, D. R. Ports, I. Zhang, R. Bianchini, H. S. Gunawi, and A. Badam. LeapIO: Efficient and Portable Virtual NVMe storage on ARM SoCs. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020. Association for Computing Machinery (ACM).

[28] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky. SILT: A memory-efficient, high-performance key-value store. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*, 2011. Association for Computing Machinery (ACM).

[29] C.-y. Liu, J. Kotra, M. Jung, and M. Kandemir. PEN: Design and Evaluation of Partial-Erase for 3D NAND-Based High Density SSDs. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*, 2018. USENIX Association.

[30] L. Lu, T. S. Pillai, H. Gopalakrishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. WiscKey: Separating Keys from Values in SSD-Conscious Storage. *ACM Transactions on Storage (TOS)*, 13(1):1–28, 2017.

[31] S. Maneas, K. Mahdaviani, T. Emami, and B. Schroeder. A Study of SSD Reliability in Large Scale Enterprise Storage Deployments. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*, 2020. USENIX Association.

[32] S. Maneas, K. Mahdaviani, T. Emami, and B. Schroeder. Reliability of SSDs in Enterprise Storage Systems: A Large-Scale Field Study. *ACM Transactions on Storage (TOS)*, 17(1):1–27, 2021.

[33] Micron Technology, Inc. 3D XPoint Technology. https://www.micron.com/products/advanced-solutions/3d-xpoint-technology, 2020. Accessed: 2021-2-3.

[34] NVM Express Workgroup. NVM Express Specification. https://nvmexpress.org/developers/nvme-specification/, 2020. Accessed: 2021-4-19.

[35] NVM Express Workgroup. NVM Express Technical Proposal for New Feature: 4053 - Zoned Namespaces. https://nvmexpress.org/, 2020. Accessed: 2021-1-17.

[36] NVM Express Workgroup. NVM Express Technical Proposal for New Feature: 4065a - Simple Copy Command. https://nvmexpress.org/, 2020. Accessed: 2021-1-17.

[37] I. L. Picoli, N. Hedam, P. Bonnet, and P. Tözün. Open-Channel SSD (What is it Good For). In *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR'20)*, 2020. www.cidrdb.org.

[38] Radian Memory Systems, Inc. RMS-325. https://www.radianmemory.com/edge-card-ssd-rms-325/, 2020. Accessed: 2021-1-17.

[39] Radian Memory Systems, Inc. Case Study: Integration of SALSA, a unified storage software stack developed by IBM Research, with Radian's Zoned Flash SSD. https://www.radianmemory.com/wp-content/uploads/2020/07/RMS-Salsa_Case_Study_Final_Approved.pdf, 2020. Accessed: 2021-1-17.

[40] Samsung. Ultra-Low Latency with Samsung Z-NAND SSD. https://www.samsung.com/semiconductor/newsroom/tech-leadership/ultra-low-latency-with-samsung-z-nand-ssd/, 2020. Accessed: 2021-2-3.

[41] M. Saxena, M. M. Swift, and Y. Zhang. FlashTier: a lightweight, consistent and durable storage cache. In *Proceedings of the 7th European Conference on Computer Systems (EuroSys'12)*, 2012. Association for Computing Machinery (ACM).

[42] B. Schroeder, R. Lagisetty, and A. Merchant. Flash Reliability in Production: The Expected and the Unexpected. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*, 2016. USENIX Association.

[43] M. Shafaei and P. Desnoyers. Near-Optimal Offline Cleaning for Flash-Based SSDs. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST'17)*, 2017. IEEE Computer Society.

[44] S. Shamasunder. Hybrid XFS—Using SSDs to Supercharge HDDs at Facebook. In *SREcon19 Asia/Pacific*, 2019. USENIX Association.

[45] Z. Shen, F. Chen, Y. Jia, and Z. Shao. DIDACache: A Deep Integration of Device and Application for Flash Based Key-Value Caching. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*, 2017. USENIX Association.

[46] R. Smith. SSD Cost/Pricing Calculator. https://www.soothsawyer.com/ssd-cost-pricing-calculator, 2019. Accessed: 2021-5-2.

[47] Z. Song, D. S. Berger, K. Li, and W. Lloyd. Learning Relaxed Belady for Content Distribution Network Caching. In *Proceedings of the 17th USENIX Conference on Networked Systems Design and Implementation (NSDI'20)*, 2020. USENIX Association.

[48] A. Sriraman and A. Dhanotia. Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020. Association for Computing Machinery (ACM).

[49] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li. RIPQ: Advanced Photo Caching on Flash for Facebook. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*, 2015. USENIX Association.

[50] Tomasz Zawadzki. SPDK v20.10: NVMe-oF multipath, NVMe ZNS, iSCSI login redirection. https://spdk.io/release/2020/10/30/20.10_release/, 2020. Accessed: 2021-4-16.

[51] Western Digital Corporation. Ultrastar DC ZN540 Now Sampling. https://blog.westerndigital.com/zns-ssd-ultrastar-dc-zn540-sampling/, 2020. Accessed: 2020-12-16.

[52] Western Digital Corporation. Zoned Storage. zonedstorage.io, 2020. Accessed: 2020-12-16.

[53] Western Digital Corporation. Zoned File Systems and Block Devices. https://zonedstorage.io/linux/fs/#file-systems-and-zoned-block-devices, 2021. Accessed: 2021-01-16.

[54] G. Wu and X. He. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, 2012. USENIX Association.

[55] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman, A. A. Chien, and H. S. Gunawi. Tiny-tail flash: near-perfect elimination of garbage collection tail latencies in NAND SSDs. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*, 2017. USENIX Association.

[56] P. Yang, N. Xue, Y. Zhang, Y. Zhou, L. Sun, W. Chen, Z. Chen, W. Xia, J. Li, and K. Kwon. Reducing garbage collection overhead in SSD based on workload prediction. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'19)*, 2019. USENIX Association.

[57] T. Zawadzki. SPDK v20.10: NVMe-oF multipath, NVMe ZNS, iSCSI login redirection. https://spdk.io/release/2020/10/30/20.10_release/, 2020. Accessed: 2021-1-7.

[58] J. Zhang, M. Kwon, M. Swift, and M. Jung. Scalable Parallel Flash Firmware for Many-core Architectures. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*, 2020. USENIX Association.

[59] T. Zhang, J. Wang, X. Cheng, H. Xu, N. Yu, G. Huang, T. Zhang, D. He, F. Li, W. Cao, et al. FPGA-Accelerated Compactions for LSM-based Key-Value Store. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*, 2020. USENIX Association.

[60] K. Zhou, S. Hu, P. Huang, and Y. Zhao. LX-SSD: Enhancing the Lifespan of NAND Flash-based Memory via Recycling Invalid Pages. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST'17)*, 2017. IEEE Computer Society.

[61] A. Zuck, Y. Li, J. Bruck, D. E. Porter, and D. Tsafrir. Stash in a Flash. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*, 2018. USENIX Association.

[62] A. Zuck, P. Gühring, T. Zhang, D. E. Porter, and D. Tsafrir. Why and How to Increase SSD Performance Transparency. In *Proceedings of the 17th Workshop on Hot Topics in Operating Systems (HotOS'19)*, 2019. Association for Computing Machinery (ACM).