ZNS-TOOLS: An eBPF-powered, Cross-Layer Storage Profiling Tool for NVMe ZNS SSDs

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

Krijn Doekemeijer
Vrije Universiteit Amsterdam
The Netherlands

Animesh Trivedi
Vrije Universiteit Amsterdam
The Netherlands

Abstract

Understanding operational characteristics of flash SSDs has been a challenging task due to their complex and closed internals. The recent emergence of Zone Namespace (ZNS) SSDs with their open interface allows the host software stack to explicitly control elements of this complexity, specifically around data placement, grouping, and garbage collection operations. Despite offering more control to applications, due to the opaque and layered nature of the software storage stack, it remains an open challenge to understand, profile, and reason about the data storage and placement decisions on ZNS devices in an end-to-end manner. In this paper, we present ZNS-TOOLS, an eBPF-powered *end-to-end* data storage events profiler (https://github.com/stonet-research/zns-tools) for the whole ZNS-enabled storage stack, including the NVMe/ZNS device driver, Linux block layer, file system, and application. Using ZNS-TOOLS we uncover diverse utilization profiles of a ZNS device for the same workload (YCSB-A), thus demonstrating the practical utility of ZNS-TOOLS.

CCS Concepts: • **General and reference** → *Performance*; • **Software and its engineering** → **Operating systems**.

Keywords: ZNS SSDs, eBPF, Visualization, Tracing, Profiling

ACM Reference Format:
Nick Tehrany, Krijn Doekemeijer, and Animesh Trivedi. 2024. ZNS-TOOLS: An eBPF-powered, Cross-Layer Storage Profiling Tool for NVMe ZNS SSDs. In *4th Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3642963.3652205

1 Introduction

Flash solid state drives (SSDs) have fundamentally changed the way we store and process data in computing. Their emergence in the mainstream commodity computing has resulted in the redesigning of almost all aspects of the host storage stack including (not limited to) the host interface [58, 63], block layer [7, 34], I/O schedulers [21, 62], file systems [33], applications [14] and distributed systems [4]. Despite the aforementioned end-to-end changes to leverage the performance characteristics of flash SSDs, reasoning about their operational characteristics remains a challenge. A key part of this challenge is the complex internal structure in which flash devices are packed to hide the nature of underlying flash chips (append-only writes, non-overwritable, requires explicit chip erase operation) [35]. These chips are actively managed by a piece of software known as the *Flash Translation Layer* (FTL) that runs inside an SSD, thus influencing its performance [38]. The FTL design and SSD's internal structure are typically proprietary information, thus forcing researchers into synthesizing *unwritten contracts* or guidelines about how to manage and extract the best performance from SSDs by collecting and analyzing detailed performance and operational trace events [20].

To address these issues, researchers have made a case for open SSD interfaces that allow more explicit control over SSD's internal operations by the host software [8]. Examples of such open interfaces are Open channel SSDs [9, 59], stream SSDs [65], software-defined flash (SDF) [43], and more recently standardized NVMe SSDs with Zone Namespace Storage (ZNS) [6, 52, 60] and emerging NVMe Flexible Data Plane (TP-4641) [10]. The NVMe ZNS interface is unique in this context as it is the only standardized and commercially available open standard (FDP is still being ratified). The new ZNS interface more accurately reflects flash chip properties (captured as *zones* that support append-only, sequential writes). The ZNS interface provides explicit control to the host software on the data placement, parallelism control, and flash chip erase operations before over-writing [13]. ZNS devices also have a set of new NVMe commands (RESET, FINISH, OPEN, CLOSE) to manage zones. Table 1 provides a high-level overview of the differences between an NVMe SSD *with* and *without* ZNS capabilities. We will provide a more detailed introduction to the ZNS devices in Section 2.

With this open interface, there is an opportunity to design and implement an end-to-end, cross-layer profiling and tracing framework. This framework can offer developers clear visibility into the newly available data placement decisions, scheduling events, and usage patterns, collectively referred to as *data-lifecycle events* in this paper. Thus, developers can

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

CHEOPS '24, April 22, 2024, Athens, Greece

ACM ISBN 979-8-4007-0538-0/24/04
https://doi.org/10.1145/3642963.3652205

	NVMe SSD *without* ZNS	NVMe SSD *with* ZNS
Addressing	Logical Block Address, or LBA	zones and LBAs within a zone
Data reads	from any LBA in sequential and random manner	
Data writes	sequential, random from any LBA	*only* sequential LBA writes within a zone
Data placement	*implicit*, SSD/FTL-managed	*explicit*, host controls placement by writing data in separate zones
Garbage collection	*implicit*, SSD/FTL-managed	*explicit*, host managed via a NVMe/ZNS zone RESET command
Parallelism	*implicit*, SSD/FTL-managed	*implicit*, but zone-level controls available to the host

Table 1. High-level comparison of NVMe flash SSDs without and with Zoned Namespaces (ZNS) [13].

leverage this information for better performance and device management [52]. For example, by following the "Grouping by death time" unwritten contract [20], data that is deleted together (i.e., has the same death time) should be stored together in the same zone so that software can garbage collect the whole zone when the data is deleted. In the past, such efforts were often hampered by the lack of access to the SSD internal state [18, 19].

However, building a detailed data-lifecycle event profiler can be a challenging task for multiple reasons. First, due to the complexity of and interactions among the modern storage stack and applications, it is not immediately clear at what level or granularity one should profile an application. System call-level tracing is one of the most popular ways to build an I/O profile of an application [2]. However, such profiling excludes any application-level data management events such as a B+-tree node splitting, or SSTables compaction in an LSM tree. Beyond that, the dynamic tracing of I/O calls can have a high overhead, whereas static tracing may require source code modifications or re-compilation. The selection and complexity of using the right tool from the available options also makes the decision non-trivial [5].

Secondly, the opaque and layered architecture of the modern storage stack makes it challenging to trace events across different layers that use different I/O abstractions. For example, an application interacts with file names or file descriptors, a file system manipulates inode structures, and the block layer only processes block addresses. Thus, identifying which file name or inode triggered an I/O event to which block address requires visibility and complex translations across different layers and layer-specific abstractions. This complexity is also apparent in the number of different ways an NVMe ZNS device can be integrated into a system (at the block level, file system, or application-level).

Lastly, the choice of a trace format and the lack of standard visualization tools also make building a profiler a challenging task. Many past tools often use tool-specific format options that can be outdated or worse, lack any documentation [2]. The close coupling between a visualization framework and the tool-format also makes it difficult to build new visualizations for the collected traces. To summarize, there is a need for a structured approach for an end-to-end data-lifecycle event tracing and profile building in order to assist application developers to best leverage modern, open SSD interfaces like ZNS.

To address the aforementioned challenges in this paper, we present ZNS-TOOLS, an eBPF-powered, cross-layer storage profiling tool. ZNS-TOOLS uses the Linux eBPF framework that has evolved into a versatile framework that allows users an unprecedented amount of online in-place trace collection and visibility into various kernel subsystems using simple C-like pseudo-code. All modern profiling tools internally leverage eBPF due to its light-weight JIT-based architecture, support for various dynamic/static profiling with kernel or application-level probes, expressive data structures (shared kernel-user maps, arrays, counters), and extensive documentation with an active community [15]. We use an eBPF-supported nanosecond-resolution timestamping mechanism to build a cross-layer timeline for data-lifecycle events. ZNS-TOOLS follows the Model-View-Controller design pattern where event collection (using eBPF probes), processing (building end-to-end timelines, offline) and visualization processes are decoupled.

ZNS-TOOLS uses the standard JSON format for the event traces (can also be extended to other timeseries formats like Panda Timeseries) that can then be visualized using dedicated frameworks like Perfetto or Chrome (Section 3). ZNS-TOOLS consists of three levels of subtools: ZNS-TOOLS.NVME for the Linux block layer and NVMe device driver-level tracing, ZNS-TOOLS.FS for file system-level tracing (F2FS and Btrfs supported with ZNS) and ZNS-TOOLS.APP for application-level end-to-end tracing (currently done for RocksDB).

Our primary contributions in this work include:

- Making a case for building an end-to-end, cross-layer storage profiler and analysis tool to reason about the NVMe ZNS SSDs utilization with the host software stack and applications (Section 2).
- ZNS-TOOLS, an end-to-end eBPF-powered framework that collects, analyzes and visualizes data-lifecycle events across the layered storage stack including the block layer and device driver (ZNS-TOOLS.NVME), file systems (ZNS-TOOLS.FS) and applications (ZNS-TOOLS.APP) (Section 3).

- We demonstrate the utility of ZNS-TOOLS by identifying the vastly uneven use of ZNS SSDs for the same user-level workload, YCSB-A (Figure 2). We further illustrate its visualization capabilities by showing an end-to-end application-level data-lifecycle event trace visualization for RocksDB with F2FS in Figure 3.
- ZNS-TOOLS are open-sourced and currently available at https://github.com/stonet-research/zns-tools.

2 Background and Motivation

In this section, we provide the necessary background on ZNS SSDs (Section 2.1), the complexity within the layered and opaque storage stack (Section 2.2), and various ZNS integration options in the storage stack that make end-to-end tracing challengeing (Section 2.3).

2.1 NVMe with Zone Namespace (ZNS) SSDs

NVMe SSDs with *Zoned Namespaces* (ZNS) offer a fundamentally new way in which applications interact with the underlying storage device [6, 60]. Table 1 shows the bird's eye view of high-level differences between NVMe SSDs *without* and *with* zone namespaces. A ZNS-capable SSD exposes its storage capacity as a set of fixed-size zones (each made up of multiple blocks) instead of the traditionally used block-oriented design. The concept of zones closely maps to the concept of flash erasure units. Like with any NVMe SSD, reads can be issued to any Logical Block Address (LBA) with ZNS SSDs. However, unlike an NVMe SSD, ZNS SSDs only allow zones to be sequentially written in an increasing LBA addresses within a zone. Before a zone can be rewritten, it must be explicitly `reset`, like the erase operation on flash chips. Resetting a zone (*Garbage Collection*, GC) is the responsibility of the host software (the kernel, or the application). To initiate a zone `reset`, there is a new NVMe command called RESET on ZNS devices. While device internal parallelism and data placement are still managed on the device, the host has some control over parallelism and placement with zone-level controls by grouping and writing data in different zones [13].

ZNS SSDs make it easier to capture previously hidden events like GC and communicate their characteristics to the SSD designers about how devices are used. For example, since applications now explicitly issue zone RESETS, GC events can be accurately measured. This control on GC makes it possible to understand its effects on wear-leveling and/or performance isolation. However, this does require tooling to become available. Currently, no such tooling exists. We argue that there should be a framework that can show how data is stored and managed in a layered storage stack on top of ZNS devices. To create such a framework, it should show: (1) where data is stored on ZNS; (2) how much I/O is issued to each zone; (3) which zones trigger the most GC and how data moves between zones because of GC.

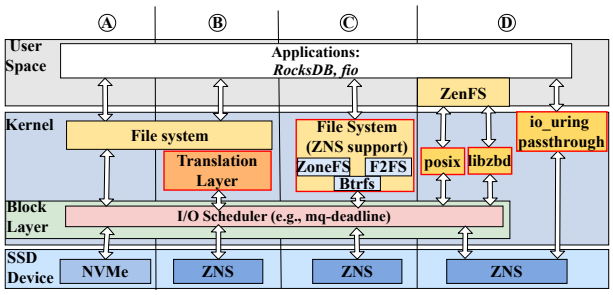


Figure 1. The current integration options for ZNS in the Linux storage stack (2024). Configuration (a) shows the integration of a conventional SSDs; (b) shows the changes at the block layer level to accommodate ZNS devices; (c) shows file system adaptations (F2FS, Btrfs) for ZNS; (d) shows application-specific integration (e.g., Zenfs for RocksDB) and `io_uring` passthrough (`ioctl()`-ish interface) to ZNS.

2.2 Layered Storage Stack

Applications typically do not directly interface with the underlying SSDs, but through a layered storage stack with multiple abstractions in-between. Layered designs are beneficial as they lead to a modular application design, but they result in *semantic gaps* between the layers. There is such a gap because each layer only exposes part of its API to the next layer, leading to a narrow communication window. Such strict layering can result in an intention mismatch between the application (passed with the POSIX `fadvise` and `fcntl` calls), file system, and underlying block storage device.

We report on two issues related to the mis-grouping of data due to the semantic gap in the state-of-the-practice combination of RocksDB and F2FS for ZNS. Data grouping based on access patterns is a commonly used technique through which log-based file systems (FS) like F2FS group data together in a single erase unit to reduce the GC overheads. The first issue happens for all applications using F2FS/ZNS where F2FS reclassifies data after doing FS-level GC for its own log-structured segments[1]. Typically, a log-structured file system like F2FS segregates data based on its access/update frequency (known as the temperature) and store data with the same/similar temperature together in a single erasure unit like F2FS segments. When an application hints that a file is hot (i.e., it may be frequently accessed), new writes to this hot file are correctly stored to a hot segment and zone. However, later when it does segment cleaning with GC, F2FS needs to clean zones and re-group valid data around. During this process, F2FS re-groups the data by changing the temperature of the data (hot to cold) and storing them in

[1]Confusingly enough, F2FS also uses the terms like *zone* and *garbage collection*. The discussion in this section pertains only to the file system-level terms, not the ZNS SSD.

a single cold segment[2]. Such an application-transparent reclassification violates the application's expectation that may have hinted that the file is hot.

The second issue is RocksDB- and F2FS-specific. RocksDB writes Sorted Strings Table (SSTable) files to a file system in two passes, raw data (the table) and a small footer (less than a page), both stored in the same file. After writing the raw data RocksDB flushes the file system, thus leaving only the footer in the page cache. If the number of dirty pages in the page cache is below a threshold of 16 pages (DEF_MIN_HOT_BLOCKS) with a few more conditions[3], F2FS classifies the footer data as hot, thus overriding any previous RocksDB hint on the SSTable (where SSTables are considered as cold data).

The aforementioned simplified but real examples illustrate the complexity associated with data classification and placement-related decisions made within the layered and opaque storage stack with semantic gaps.

2.3 ZNS Integration Options

Having discussed the complexity of the layered storage stack, the availability of ZNS-related events (zone commands) further complicates the end-to-end control reasoning. A ZNS SSD can be integrated into the host storage stack in multiple manners. Figure 1 shows multiple possible options for the ZNS integration. Configuration-A shows the standard NVMe stack with a file system, block layer, and SSD device (without the ZNS interface). In configuration-B, a ZNS SSD device can be integrated above the block layer, but below the file system so that file systems do not require any modification to work with the ZNS SSD [12]. This design choice also implies that file systems do not have visibility into the ZNS utilization and operations, thus defeating the purpose of the ZNS interface. For example, here both file systems (e.g., F2FS) and ZNS SSDs will perform independent GC operations without any coordination. Configuration-C is a native file system-level integration of ZNS SSDs where file systems are modified to become ZNS-aware, thus further unifying file system level and ZNS level operations [47, 54]. Currently, within the Linux kernel, F2FS and Btrfs file systems are ZNS-aware. Lastly, a direct application-level integration is also possible for maximum semantic integration of ZNS and application data storage semantics. An example of this approach is RocksDB's domain-specific ZenFS storage backend for ZNS SSDs [6, 61].

With these ZNS integration options, how user data is stored, which layer made the data placement and grouping decisions, and in which file, or data segment, or zone data is stored become important questions to answer. The availability of ZNS makes some of these decision-making more explicit, but these decisions must be analyzed in an end-to-end manner to reason about the performance and operational characteristics of the ZNS-supporting storage stack.

3 ZNS-TOOLS: Design and Implementation

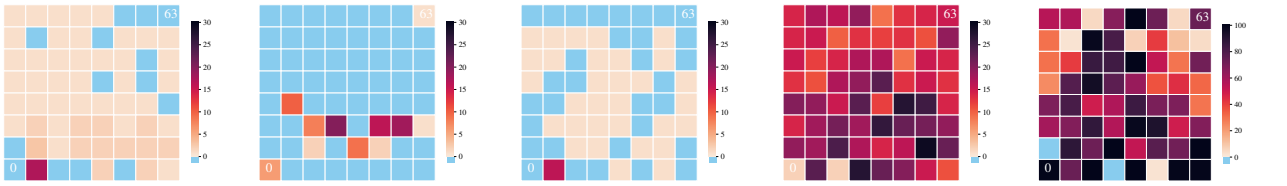
Having established our motivation for an end-to-end, cross-layer data-lifecycle event collection and profiling tool, in this section, we present the design and implementation of one such tool: ZNS-TOOLS. ZNS-TOOLS is a collection of three tools (so far) that collect, process, and visualize data-lifecycle management-related events in the Linux storage stack (originally developed for the `v5.19` version). We made the following design choices with ZNS-TOOLS:

- **Do one thing well:** Instead of building a single, über-all framework for I/O profiling, we follow the UNIX (pipe) philosophy[4] and aim to design multiple, single-purpose tools that can trace and profile data-lifecycle events from multiple layers. The scope of these tools is restricted to a single layer of storage (application, file system, and block with device driver). Ideally, these tools can be combined, like the UNIX pipe abstraction, to build a complex tracing DAG. However, in the current implementation they work independently. We are working on improving the design.
- **Keep it modular and standardized:** We follow the Model-View-Controller (MVC) paradigm in the design of ZNS-TOOLS where data collection (the controller), processing (the model), and visualization (the view), are decoupled from each other, thus offering flexibility in optimizing and upgrading the individual component. For example, the output format can be chosen to any of the multiple available formats (currently supported JSON) and can be visualized using any of the multiple frameworks possible (Perfetto, Google Chrome).
- **Keep it lightweight:** Lastly, we leverage eBPF, which is shown to be a versatile, comprehensive, but lightweight data event collection and processing framework. ZNS-TOOLS are written as a set of Python/C utilities that use the Linux/eBPF-based filtering and event collection framework. eBPF probes can be put in the Linux kernel as well as in userspace application code dynamically to construct an end-to-end cross-layer timeline using eBPF-supported timestamps. Today, eBPF has a vibrant software ecosystem around it, https://ebpf.io/.

ZNS-TOOLS works by running the command as root while the under-benchmarking application is running. When the

[2] https://github.com/torvalds/linux/blob/v6.8-rc7/fs/f2fs/gc.c#L1285.
[3] https://github.com/torvalds/linux/blob/v6.8-rc7/fs/f2fs/data.c#L3063.

[4] Basics of the Unix Philosophy, *"Make each program do one thing well"*, http://www.catb.org/~esr/writings/taoup/html/ch01s06.html. Originally taken from the Bell System Technical Journal, 57, Forward on UNIX Time-Sharing System by M.D. McIlroy, E.N. Pinson and B.A. Tague (July-August 1978) at https://archive.org/details/bstj57-6-1899/page/n3/mode/2up.



(a) RocksDB + F2FS **(b)** RocksDB + ZenFS **(c)** MongoDB + F2FS **(d)** PostgreSQL + F2FS **(e)** RocksDB + aged F2FS

Figure 2. RESET visualization using ZNS-TOOLS.NVME for the identical YCSB workload-A (update heavy workload, 50% read, 50% write) running on multiple ways ZNS SSDs can be leveraged by a storage stack. The zones are enumerated with their zone numbers from 0 (bottom, left) to 63 (top, right). Figure 2e has a different heatmap scale (0-100). These heatmap visualizations are done using Seaborn: statistical data visualization framework.

tool runs, it collects both application-specific and system-wide events with timestamps (`bpf_ktime_get_ns`) to build an end-to-end timeline. The tools currently hold all tracing data in memory while tracing (100s–1,000 of MiBs, depending on the tracing frequency and details) and eventually write out to JSON files when stopped. We have also tested an alternate design where trace data is written periodically or after a certain number of events, thus relaxing the requirements for the memory capacity needed. This design is tested in small examples, but not implemented for all tools. In the coming subsection, we provide a more detailed description of three specific tools that constitute ZNS-TOOLS right now.

3.1 ZNS-TOOLS.NVME

ZNS-TOOLS.NVME collects and visualizes events collected from the Linux block and NVMe device driver layer. Specifically, the tool traces individual I/O and zone management operations over a user-defined time period and generates visualizations. Tracing such data is useful for investigating the access patterns (data grouping, placement) at the device-level independently of file system and application-level I/O patterns. For example, the tool can be used to study the number of RESETS per zone to study the impact of data writing and wear-leveling. ZNS-TOOLS.NVME utilizes BPFtrace [24], which inserts probes into the Linux kernel functions, that upon being triggered (i.e., the function being called), initiate data collection. The tracing script captures the NVMe commands events and based on the type of command (I/O or management) extracts further details (size of payload, time, number of zones resets, etc.). It then maps each command to its corresponding zone to generate zone-specific traces. Currently, it traces ZNS WRITE, APPEND, READ, and RESET operations, but it can be extended to more operations (e.g., FINISH operations) and triggering conditions by writing a few lines of eBPF filters.

Uniquely, ZNS-TOOLS.NVME also supports tracing in an NVMe virtualized environment within the QEMU/VM framework. Here, the zone RESET command sets the *function* argument for the type of command to `REQ_OP_DRV_OUT`, indicating that the host driver (e.g., `vfio-pci` when using the NVMe passthrough to the VM) is responsible for the request. The user can decide when to stop the script by hand by sending the SIGINT signal. Upon termination of the tracing scripts, all collected data is written to a file, followed by post-processing to generate a visualization of the various collected information. A unique benefit of ZNS devices is the representation of zones, thus allowing collected data to be grouped and represented on the basis of a zone.

To illustrate the utility of ZNS-TOOLS.NVME, we run an *identical* workload on a number of ZNS-supporting database/file systems and quantify the workload's RESET profile. The number of RESETS is directly linked to the number of Program/Erase (P/E) cycles that an SSD can undergo before exhausting flash chips. Visualizing this information gives a direct approximation of the wear-level capabilities of the ZNS-aware software stack. We run experiments on an emulated NVMe ZNS device (QEMU `v6.0.0`) with 64 zones of size 64MiB (4GiB in total). Our workload is the YCSB workload-A (update heavy workload, 50% read, 50% write) [11] on RocksDB (`v7.4.3`) [17], MongoDB (`v6.06`) [41], and PostgreSQL (`v9.6.24`) [39] as storage backend targets. With RocksDB, we have three configurations of a storage backend: (i) the default POSIX backend with ZNS/F2FS (configuration (c) in Figure 1); (ii) a ZNS-specific domain-specalized file backend, called ZenFS [6, 61] (configuration (d) in Figure 1); and (iii) a POSIX backend with an *aged* F2FS [28].

Figure 2 shows a heatmap visualization of trace data collected (about 10–100s of KiB). Here, each cell represents a zone, enumerated from the bottom left (zone 0) to the top right (zone 63). The color of the cell indicates the total number of RESETS issued to the zone (since startup). Blue squares indicate zones to which no reset commands were issued at all while the darker colours represents more RESETS issued (captured by the scale of the heatmap). There are three interesting observations here. Firstly, for an identical workload, these ZNS-enabled storage stacks show vastly different RESET profiles. We report that the PostgreSQL ZNS stack issues a large number of RESETS, as shown in Figure 2-(d). Additionally, we can identify that in the case of F2FS, one of the

bottom left zones (zone 2) is heavily utilized. This zone corresponds to a warm node zone initialized by F2FS, where the inodes of files are written. Secondly, file system aging has a significant impact on the reset profile as shown by Figure 2e. Aging of F2FS is done by executing the same YCSB-A workload 10× times. Lastly, the domain-specific RocksDB ZNS-file system, ZenFS (Figure 2b), does a limited amount of wear-leveling across the device and re-uses the same zones over and over again, thus leading to a few zones which are frequently reset. In comparison, F2FS has a more even distribution of RESETs (Figure 2a). All these results demonstrate a need and the utility of such trace collection and visualization tools to study the impact of various integration levels of ZNS devices as shown in Figure 1.

3.2 ZNS-TOOLS.FS

ZNS-TOOLS.FS has two specific tracing responsibilities: (1) capture file I/O events to their block-level storage locations by extracting file placement information from the file system; (2) capture file system-specific *semantic* information such as data grouping or heat-based file extent classification. Both of these pieces of information are critical for understanding the data placement and grouping decisions made by a file system. ZNS SSDs require an explicit file system level garbage collection procedure [51]. With ZNS SSDs and a log-structured file system, the location of the file data is thus dynamic and changes constantly based on (i) the user-initiated events like file read/writes; and (ii) the file system-initiated events such as garbage collection with heat segregation. ZNS-TOOLS.FS aims to capture and report traces of both of these kinds of events.

ZNS-TOOLS.FS retrieves file location mappings (extent-oriented) from the Linux kernel using the `ioctl()` syscall with the `FIEMAP` flag[5]. Both ZNS-compliant file systems (F2FS and Btrfs) support this call. A file location is represented by contiguous storage extents (LBA address-length pair), and a file can have multiple non-contiguous extents. With `FIEMAP`, file systems that implement the tracking of extents return the extent information to the `ioctl()` caller. By iterating over the logical range of a file, ZNS-TOOLS.FS retrieves data mappings of all the extents for a particular file. The collected file extents are then mapped to their respective zone(s) containing the file's data using their logical LBA address ranges. For example, on a ZNS SSD with a zone size of 1MiB, logical addresses between $[0, 1MiB)$ fall within the first zone, $[1MiB, 2MiB)$ on the second zone, and so on. The zone size and zone size ranges can be queried with the ZNS device using a zone management command. With all information about file extents, their addresses, address-to-zone mappings, ZNS-TOOLS.FS reports a detailed profile of the extent distribution (min, max, percentiles), zone-level placement information, and hole or fragmentation statistics. Holes (of fragmentation levels) are an important performance-related property that are known to cause severe performance degradation [26, 28, 29, 44, 66] because they violate the unwritten "Request Scale" contract that recommends large sequential requests [20].

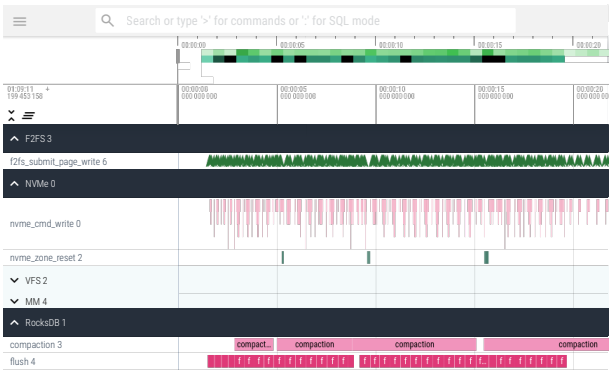


Figure 3. An end-to-end, cross-layer data-lifecycle event visualization example using ZNS-TOOLS.APP. This timeline is generated using an input JSON trace file with Perfetto.

Though placement-level information can be extracted for any `FIEMAP`-supporting file system, the extraction of semantic information is file system specific. Here, ZNS-TOOLS.FS only supports F2FS which stores file data in multiple segments, and a single F2FS segment can contain data from multiple files. By default, F2FS uses three classes of segment temperature classification (hot, warm, cold) for two types of data: file data, and file metadata (inodes). ZNS-TOOLS.FS supports classifying various file segments (in the F2FS parlance) by reading the segment hotness classification from the Linux `procfs`[6]. Segments can also store information about directory data (the files stored within). Hence, ZNS-TOOLS.FS is also capable of reading the F2FS superblock, checkpoints, and the NAT table. Put together, ZNS-TOOLS.FS reports for any file or directory all of its F2FS segments, their hotness classifications, the number of file extents contained within each segment, the inode-to-zone mappings, and the location of the segments on a ZNS SSD.

3.3 ZNS-TOOLS.APP

ZNS-TOOLS.APP does a collaborative userspace and kernel-based tracing for various data-lifecycle related events to build an end-to-end, time-based (nanoseconds-resolution), cross-layer event profile. It has eBPF probes for two parts, the kernel and a userspace application, to collect trace events. We have a pre-defined (but extensible) number of eBPF probes to collect trace events with timestamps (in nanoseconds) inside the kernel on the following particular function call paths: (i) the VFS - mostly file I/O and hint syscalls such as `fcntl_set_rw_hint`, `vfs_create`, `vfs_fsync`; (ii) the F2FS file system and memory management related

[5] https://www.kernel.org/doc/Documentation/filesystems/fiemap.txt

[6] /proc/fs/f2fs/nvme0n1/segment_info

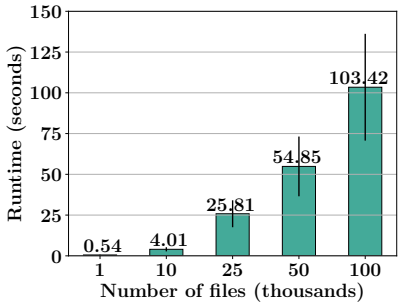


Figure 4. ZNS-TOOLS.FS runtime (y-axis, lower is better) on F2FS with the number of files (the x-axis).

calls like `f2fs_submit_page_write`, `move_data_block`, `mm_do_writepages`; (iii) the NVMe device command tracing (uses ZNS-TOOLS.NVME). For the userspace application, we rely on the application developer to identify such a function or call path of importance. We have done so for RocksDB, where we trace `NotifyOnCompactionBegin` and `NotifyOnCompactionCompleted` (among others) for LSM-tree compaction events. The idea is that by collecting events across the kernel and userspace, we can attribute events across the stack, thus building an end-to-end profile. Figure 3 illustrates this end-to-end timeline generated from traces using Perfetto. The figure shows a few selected events (for brevity) with their timelines from five layers (bottom to top): RocksDB, MM, VFS, NVMe, and F2FS. Such a timeline visualization gives an understanding of how the different RocksDB operations interacting with F2FS affect the utilization of the ZNS storage space, file classification, and data movement over time, thus making it easy to reason about the decision making process by following the timeline.

4 Overheads of ZNS-TOOLS

In this section, we briefly report on the tracing overheads of ZNS-TOOLS. Figure 4 reports the overheads associated with the repeated calling of `ioctl()` call with `FIEMAP` to resolve file extents and F2FS segment information. We report the runtime (in seconds) of ZNS-TOOLS.FS from an F2FS file system mount point containing 1,000 to 100,000 files in a single directory. We observe that the runtime initially grows slowly (below the linear cost) up to 25K files and then follows a linear overhead growth pattern.

We also report the runtime overheads associated with doing a full application tracing with the ZNS-TOOLS.APP (not shown). In our experiments, ZNS-TOOLS.APP incurs a small overhead of less than 10%. With `fillrandom` and `overwrite` workloads in db_bench of RocksDB, the overheads are 7.44% (164.85K IOPS without vs. 152.58K IOPS with tracing) and 3.15% (120.65K IOPS without vs. 116.85K IOPS with tracing), respectively. Furthermore, the size of the trace file for the visualization in Figure 3 is around 150MiB.

5 Related Work

There is a large body of work on studying the interaction of a file system with storage devices by collecting, analyzing, and visualizing operational traces. Flash SSDs, with their complex internal logic and *unwritten contracts*, have also been studied in detail for performance and operation characterizations [23, 30, 31, 36] with the impact of GC operations [22, 32, 45, 56, 57]. Jung and Kandemir provide a thorough and detailed empirical evaluation of six SSDs for their read, write, TRIM (similar to ZNS RESET command) interference from background activities (GC and buffer flush) performances [27]. In their seminal work, Traeger et al. identify various pitfalls in file system benchmarking [55]. CodeMRI is a framework to capture traces from a workload to build a microprofile that can synthetically be scaled up and down to study the impact of a workload on a storage system [1]. Lu et al. report on an eight year file system evolution study, however, their study is done manually by classifying and studying various development patches [37]. In a similar spirit to ZNS-TOOLS, Prabhakaran et al. introduce techniques to study file system behavior with semantic knowledge of events and on-disk data structure layouts [46]. ZNS-TOOLS extends such motivation to include workloads with the new ZNS management operations as well.

The collection of traces to study storage systems has a long history. Ousterhout, et al. present one of the early results from trace collection, operational data analytics, and simulator-based trace replay to study the impact of caches on the file system performance [42]. Ellard and Seltzer make a case for decoupling the NFS trace collection from the analysis [16]. Several block-level tracking tools exists (BCC's biotop, BCC's bIOsnoop, DTraces's IOsnoop), however, they do not link the block-level I/O commands back to the file system. IOScope uses eBPF assisted file offset-based I/O tracing [50]. However, its tracing is limited to the files (at the VFS level) and does not connect the file to its location, which can change based on the file system and application level operations. Re-Animator [2] does system call level tracing using Linux tracepoints that can include data payloads. AndroStep with MobiBench is an I/O traces collection, reply, and analysis framework in Android mobile devices [25]. Broadly there is a rich history of collecting file system traces, analyzing them, and replaying them to understand the impact of optimizations [3, 16, 42, 49, 53]. Much of these works only focus on basic read/write interfaces that are sufficient for HDDs, but not SSDs with their expressive interface and active flash management via an FTL. Our work focuses on developing extensible and flexible mechanisms to collect the relevant operational data for SSDs that can be used for modeling and the visualization of the full stack storage operations (device, file systems, and applications).

In a distributed setting, Wintermute [40] is a distributed data analytics system that collects operational data and traces

across multiple machines and software components to stitch a single timeline for analysis. Apollo is a distributed telemetry data collection and storage framework that leverages ML to identify the right data to collect [48]. Beacon is an I/O trace collection framework for the Sunway TaihuLight supercomputer that collects various I/O events in a single system for performance analysis and diagnosis [64]. In comparison to these works, the focus of zns-tools is on collecting, analyzing, and visualizing operational trace data across multiple storage stack layers (vertical integration) to reason about data-lifecycle events with visualization.

6 Conclusion and On-Going Work

In this paper we have presented the design and implementation of the open-sourced zns-tools to collect, process, and visualize data-lifecycle related events on NVMe ZNS storage SSDs. The availability of an open SSD interface such as ZNS, where the host software (block layer, file system, applications) controls various data management related events motivates us to build such an end-to-end visualization tool. As a next step, we aim to scale zns-tools on multiple high-capacity devices (TBs SSDs), generalize the design to non-ZNS ecosystems with the emerging NVMe FDP support, and extend the zns-tools.app to other storage-heavy applications such as databases and HPC workloads (beyond RocksDB). zns-tools is open-sourced and currently available on GitHub at https://github.com/stonet-research/zns-tools.

Acknowledgments

This work is supported by generous hardware donations from Western Digital arranged by Matias Bjørling. This work is partially funded by Netherlands-funded projects from the Dutch Research Council (NWO) grants (OCENW.KLEIN.561 and OCENW.KLEIN.209) and GFP 6G FNS, and the EU-funded projects MCSA-RISE Cloudstars and Horizon Graph-Massivizer. Krijn Doekemeijer is funded by the VU PhD innovation program. We further thank the anonymous CHEOPS'24 reviewers and the AtLarge group at the Vrije Universiteit Amsterdam for their feedback.

References

[1] Nitin Agrawal, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2008. Towards Realistic File-System Benchmarks with CodeMRI. *SIGMETRICS Perform. Eval. Rev.* 36, 2 (aug 2008), 52–57. https://doi.org/10.1145/1453175.1453184

[2] Ibrahim Umit Akgun, Geoff Kuenning, and Erez Zadok. 2020. Re-Animator: Versatile High-Fidelity Storage-System Tracing and Replaying. In *Proceedings of the 13th ACM International Systems and Storage Conference* (Haifa, Israel) *(SYSTOR '20)*. Association for Computing Machinery, New York, NY, USA, 61–74. https://doi.org/10.1145/3383669.3398276

[3] Akshat Aranya, Charles P. Wright, and Erez Zadok. 2004. Tracefs: a file system to trace them all. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (San Francisco, CA) *(FAST'04)*. USENIX Association, USA, 10.

[4] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobbler, Michael Wei, and John D. Davis. 2012. CORFU: A Shared Log Design for Flash Clusters. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, San Jose, CA, 1–14. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/balakrishnan

[5] Jean Luca Bez, Suren Byna, and Shadi Ibrahim. 2023. I/O Access Patterns in HPC Applications: A 360-Degree Survey. *ACM Comput. Surv.* 56, 2, Article 46 (sep 2023), 41 pages. https://doi.org/10.1145/3611007

[6] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. 2021. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Virtual Event, 689–703. https://www.usenix.org/conference/atc21/presentation/bjorling

[7] Matias Bjørling, Jens Axboe, David Nellans, and Philippe Bonnet. 2013. Linux block IO: introducing multi-queue SSD access on multi-core systems. In *Proceedings of the 6th International Systems and Storage Conference* (Haifa, Israel) *(SYSTOR '13)*. Association for Computing Machinery, New York, NY, USA, Article 22, 10 pages. https://doi.org/10.1145/2485732.2485740

[8] Matias Bjørling, Philippe Bonnet, Luc Bouganim, and Niv Dayan. 2013. The Necessary Death of the Block Device Interface. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, Asilomar, CA, USA. http://cidrdb.org/cidr2013/Papers/CIDR13_Paper89.pdf

[9] 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* (Santa clara, CA, USA) *(FAST'17)*. USENIX Association, USA, 359–373.

[10] Mike Allison Chris Sabol, Ross Stenfort. August 2023. Flexible Data Placement: State of the Union. https://nvmexpress.org/wp-content/uploads/FMS-2023-Flexible-Data-Placement-FDP-Overview.pdf. Accessed: 2024-20-01.

[11] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, Indiana, USA) *(SoCC '10)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/1807128.1807152

[12] Western Digital Corporation. Accessed: 2024-Feb-01. dm-zap, Device mapper for converting sequential write required zones into conventional zones. https://github.com/westerndigitalcorporation/dm-zap.

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

[14] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Evolution of Development Priorities in Key-value Stores Serving Large-scale Applications: The RocksDB Experience. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, Virtual Event, 33–49. https://www.usenix.org/conference/fast21/presentation/dong

[15] eBPF - Introduction, Tutorials & Community Resources. 2024. https://ebpf.io/. Accessed: 2024-Feb-02.

[16] Daniel Ellard and Margo Seltzer. 2003. New NFS Tracing Tools and Techniques for System Analysis. In *Proceedings of the 17th USENIX Conference on System Administration* (San Diego, CA) *(LISA '03)*. USENIX

Association, USA, 73–86.

[17] Facebook. 2024. RocksDB: A Persistent Key-Value Store for Flash and RAM Storage. https://github.com/facebook/rocksdb. Accessed: 2024-02-01.

[18] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. 2017. MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 168–183. https://doi.org/10.1145/3132747.3132774

[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 *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Virtual Event, 173–190. 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* (Belgrade, Serbia) *(EuroSys '17)*. Association for Computing Machinery, New York, NY, USA, 127–144. https://doi.org/10.1145/3064176.3064187

[21] Mohammad Hedayati, Kai Shen, Michael L. Scott, and Mike Marty. 2019. Multi-Queue Fair Queueing. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, USA) *(USENIX ATC '19)*. USENIX Association, USA, 301–314.

[22] Jian Hu, Hong Jiang, and Prakash Manden. 2012. Understanding Performance Anomalies of SSDs and Their Impact in Enterprise Application Environment. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (London, England, UK) *(SIGMETRICS '12)*. Association for Computing Machinery, New York, NY, USA, 415–416. https://doi.org/10.1145/2254756.2254820

[23] H. Howie Huang, Shan Li, Alex Szalay, and Andreas Terzis. 2011. Performance modeling and analysis of flash-based storage devices. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE Computer Society, Denver, Colorado, USA, 1–11. https://doi.org/10.1109/MSST.2011.5937213

[24] iovisor. 2024. BPFTrace. https://github.com/iovisor/bpftrace. Accessed: 2024-02-01.

[25] Sooman Jeong, Kisung Lee, Jungwoo Hwang, Seongjin Lee, and Youjip Won. 2013. AndroStep: Android storage performance analysis tool. In *Software Engineering 2013 - Workshopband*, Stefan Wagner and Horst Lichter (Eds.). Gesellschaft für Informatik e.V., Bonn, 327–340.

[26] Cheng Ji, Li-Pin Chang, Liang Shi, Chao Wu, Qiao Li, and Chun Jason Xue. 2016. An empirical study of file-system fragmentation in mobile storage systems. In *Proceedings of the 8th USENIX Conference on Hot Topics in Storage and File Systems* (Denver, CO) *(HotStorage'16)*. USENIX Association, USA, 76–80.

[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* (Pittsburgh, PA, USA) *(SIGMETRICS '13)*. Association for Computing Machinery, New York, NY, USA, 203–216. https://doi.org/10.1145/2465529.2465548

[28] Saurabh Kadekodi, Vaishnavh Nagarajan, and Gregory R. Ganger. 2018. Geriatrix: Aging what you see and what you don't see. A file system aging approach for modern storage systems. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 691–704. https://www.usenix.org/conference/atc18/presentation/kadekodi

[29] Jaegeuk Kim. 2021. DEFRAG.F2FS. https://manpages.debian.org/testing/f2fs-tools/defrag.f2fs.8.en.html.

[30] Joonsung Kim, Kanghyun Choi, Wonsik Lee, and Jangwoo Kim. 2021. Performance Modeling and Practical Use Cases for Black-Box SSDs. *ACM Trans. Storage* 17, 2, Article 14 (jun 2021), 38 pages. https://doi.org/10.1145/3440022

[31] Jihun Kim, Joonsung Kim, Pyeongsu Park, Jong Kim, and Jangwoo Kim. 2018. SSD Performance Modeling Using Bottleneck Analysis. *IEEE Computer Architecture Letters* 17, 1 (2018), 80–83. https://doi.org/10.1109/LCA.2017.2779122

[32] Tomer Lange, Joseph (Seffi) Naor, and Gala Yadgar. 2021. Offline and Online Algorithms for SSD Management. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3, Article 33 (dec 2021), 28 pages. https://doi.org/10.1145/3491045

[33] 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* (Santa Clara, CA) *(FAST'15)*. USENIX Association, USA, 273–286.

[34] Gyusun Lee, Seokha Shin, Wonsuk Song, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. 2019. Asynchronous I/O stack: a low-latency kernel I/O stack for ultra-low latency SSDs. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, USA) *(USENIX ATC '19)*. USENIX Association, USA, 603–616.

[35] 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* (Haifa, Israel) *(SYSTOR '22)*. Association for Computing Machinery, New York, NY, USA, 72–84. https://doi.org/10.1145/3534056.3534940

[36] Shan Li and H. Howie Huang. 2010. Black-Box Performance Modeling for Solid-State Drives. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society, Miami, Florida, USA, 391–393. https://doi.org/10.1109/MASCOTS.2010.48

[37] Lanyue Lu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Shan Lu. 2013. A Study of Linux File System Evolution. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*. USENIX Association, San Jose, CA, 31–44. https://www.usenix.org/conference/fast13/technical-sessions/presentation/lu

[38] Dongzhe Ma, Jianhua Feng, and Guoliang Li. 2014. A Survey of Address Translation Technologies for Flash Memories. *ACM Comput. Surv.* 46, 3, Article 36 (jan 2014), 39 pages. https://doi.org/10.1145/2512961

[39] Bruce Momjian. 2001. *PostgreSQL: introduction and concepts*. Addison-Wesley Longman Publishing Co., Inc., USA.

[40] Alessio Netti, Micha Müller, Carla Guillen, Michael Ott, Daniele Tafani, Gence Ozer, and Martin Schulz. 2020. DCDB Wintermute: Enabling Online and Holistic Operational Data Analytics on HPC Systems. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (Stockholm, Sweden) *(HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 101–112. https://doi.org/10.1145/3369583.3392674

[41] Trong-Dat Nguyen and Sang-Won Lee. 2018. Optimizing MongoDB Using Multi-streamed SSD. In *Proceedings of the 7th International Conference on Emerging Databases*, Wookey Lee, Wonik Choi, Sungwon Jung, and Min Song (Eds.). Springer Singapore, Singapore, 1–13.

[42] John K. Ousterhout, Hervé Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson. 1985. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles* (Orcas Island, Washington, USA) *(SOSP '85)*. Association for Computing Machinery, New York, NY, USA, 15–24. https://doi.org/10.1145/323647.323631

[43] 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 (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 471–484. https://doi.org/10.1145/2541940.2541959

[44] Jonggyu Park and Young Ik Eom. 2021. FragPicker: A New Defragmentation Tool for Modern Storage Devices. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 280–294. https://doi.org/10.1145/3477132.3483593

[45] Roman Pletka, Ioannis Koltsidas, Nikolas Ioannou, Saša Tomić, Nikolaos Papandreou, Thomas Parnell, Haralampos Pozidis, Aaron Fry, and Tim Fisher. 2018. Management of Next-Generation NAND Flash to Achieve Enterprise-Level Endurance and Latency Targets. *ACM Trans. Storage* 14, 4, Article 33 (dec 2018), 25 pages. https://doi.org/10.1145/3241060

[46] Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2005. Analysis and Evolution of Journaling File Systems. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (Anaheim, CA) *(ATEC '05)*. USENIX Association, USA, 8.

[47] Devashish R. Purandare, Sam Schmidt, and Ethan L. Miller. 2023. Persimmon: an append-only ZNS-first filesystem. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, Washington, DC, USA, 308–315. https://doi.org/10.1109/ICCD58817.2023.00054

[48] Neeraj Rajesh, Hariharan Devarajan, Jaime Cernuda Garcia, Keith Bateman, Luke Logan, Jie Ye, Anthony Kougkas, and Xian-He Sun. 2021. Apollo: An ML-Assisted Real-Time Storage Resource Observer. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing* (Virtual Event, Sweden) *(HPDC '21)*. Association for Computing Machinery, New York, NY, USA, 147–159. https://doi.org/10.1145/3431379.3460640

[49] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson. 2000. A comparison of file system workloads. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (San Diego, California) *(ATEC '00)*. USENIX Association, USA, 4.

[50] Abdulqawi Saif, Lucas Nussbaum, and Ye-Qiong Song. 2018. IOscope: A Flexible I/O Tracer for Workloads' I/O Pattern Characterization. In *High Performance Computing*, Rio Yokota, Michèle Weiland, John Shalf, and Sadaf Alam (Eds.). Springer International Publishing, Cham, 103–116.

[51] 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* (Boston, MA, USA) *(HotStorage '23)*. Association for Computing Machinery, New York, NY, USA, 102–108. https://doi.org/10.1145/3599691.3603409

[52] Theano Stavrinos, Daniel S. Berger, Ethan Katz-Bassett, and Wyatt Lloyd. 2021. Don't Be a Blockhead: Zoned Namespaces Make Work on Conventional SSDs Obsolete. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Ann Arbor, Michigan) *(HotOS '21)*. Association for Computing Machinery, New York, NY, USA, 144–151. https://doi.org/10.1145/3458336.3465300

[53] V. Tarasov, S. Kumar, J. Ma, D. Hildebrand, A. Povzner, G. Kuenning, and E. Zadok. 2012. Extracting flexible, replayable models from large block traces. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (San Jose, CA) *(FAST'12)*. USENIX Association, USA, 22.

[54] Nick Tehrany. 2023. msF2FS: Design and Implementation of an NVMe ZNS SSD Optimized F2FS File System. http://resolver.tudelft.nl/uuid:3c2b3e73-6aff-45f3-af43-31a50314b547. Accessed: 2024-02-02.

[55] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. 2008. A Nine Year Study of File System and Storage Benchmarking. *ACM Trans. Storage* 4, 2, Article 5 (may 2008), 56 pages. https://doi.org/10.1145/1367829.1367831

[56] Benny Van Houdt. 2013. A Mean Field Model for a Class of Garbage Collection Algorithms in Flash-Based Solid State Drives. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems* (Pittsburgh, PA, USA) *(SIGMETRICS '13)*. Association for Computing Machinery, New York, NY, USA, 191–202. https://doi.org/10.1145/2465529.2465543

[57] Robin Verschoren and Benny Van Houdt. 2019. On the Endurance of the D-Choices Garbage Collection Algorithm for Flash-Based SSDs. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 4, 3, Article 13 (jul 2019), 23 pages. https://doi.org/10.1145/3326121

[58] Don H Walker. Accessed: 2024-02-02. A Comparison of NVMe and AHCI. https://sata-io.org/sites/default/files/documents/NVMe%20and%20AHCI_%20_long_.pdf.

[59] Peng Wang, Guangyu Sun, Song Jiang, Jian Ouyang, Shiding Lin, Chen Zhang, and Jason Cong. 2014. An efficient design and implementation of LSM-tree based key-value store on open-channel SSD. In *Proceedings of the Ninth European Conference on Computer Systems* (Amsterdam, The Netherlands) *(EuroSys '14)*. Association for Computing Machinery, New York, NY, USA, Article 16, 14 pages. https://doi.org/10.1145/2592798.2592804

[60] Western Digital. Accessed: 2024-02-02. Ultrastar DC ZN540. https://www.westerndigital.com/products/internal-drives/data-center-drives/ultrastar-dc-zn540-nvme-ssd.

[61] Western Digital. Accessed: 2024-02-02. ZenFS: RocksDB Storage Backend for ZNS SSDs and SMR HDDs. https://github.com/westerndigitalcorporation/zenfs.

[62] 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*, Marcos K. Aguilera and Gala Yadgar (Eds.). USENIX Association, Virtual Event, 403–415. https://www.usenix.org/conference/fast21/presentation/woo

[63] NVM Express Workgroup. 2022. *NVM Express NVM Command Set Specification 2.0*. Standard. Available from: https://nvmexpress.org/specifications.

[64] Bin Yang, Xu Ji, Xiaosong Ma, Xiyang Wang, Tianyu Zhang, Xiupeng Zhu, Nosayba El-Sayed, Haidong Lan, Yibo Yang, Jidong Zhai, Weiguo Liu, and Wei Xue. 2019. End-to-end I/O Monitoring on a Leading Supercomputer. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 379–394. https://www.usenix.org/conference/nsdi19/presentation/yang

[65] Jingpei Yang, Rajinikanth Pandurangan, Changho Choi, and Vijay Balakrishnan. 2017. AutoStream: Automatic Stream Management for Multi-Streamed SSDs. In *Proceedings of the 10th ACM International Systems and Storage Conference* (Haifa, Israel) *(SYSTOR '17)*. Association for Computing Machinery, New York, NY, USA, Article 3, 11 pages. https://doi.org/10.1145/3078468.3078469

[66] Lihua Yang, Fang Wang, Zhipeng Tan, Dan Feng, Jiaxing Qian, and Shiyun Tu. 2020. ARS: Reducing F2FS Fragmentation for Smartphones Using Decision Trees. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe* (Grenoble, France) *(DATE '20)*. EDA Consortium, San Jose, CA, USA, 1061–1066.