Understanding (Un)Written Contracts of NVMe ZNS Devices with zns-tools

Tehrany, Nick; Doekemeijer, Krijn; Trivedi, Animesh

2023

Understanding (Un)Written Contracts of NVMe ZNS Devices with `zns-tools`

Nick Tehrany, Krijn Doekemeijer, and Animesh Trivedi
VU, Amsterdam

Abstract

Operational and performance characteristics of flash SSDs have long been associated with a set of Unwritten Contracts due to their hidden, complex internals and lack of control from the host software stack. These unwritten contracts govern how data should be stored, accessed, and garbage collected. The emergence of Zoned Namespace (ZNS) flash devices with their open and standardized interface allows us to write these unwritten contracts for the storage stack. However, even with a standardized storage-host interface, due to the lack of appropriate end-to-end operational data collection tools, the quantification and reasoning of such contracts remain a challenge. In this paper, we propose `zns.tools`, an open-source framework for end-to-end event and metadata collection, analysis, and visualization for the ZNS SSDs contract analysis. We showcase how `zns.tools` can be used to understand how the combination of RocksDB with the F2FS file system interacts with the underlying storage. Our tools are available openly at https://github.com/stonet-research/zns-tools.

1 Introduction

The emergence of flash solid state drives (SSDs) has resulted in the redesigning of the host interfaces [49, 53], the block layer [6,29], I/O schedulers [16,52], file systems [28], applications [11], and distributed systems [4]. Despite the aforementioned significant end-to-end changes to leverage the characteristics of flash storage, the reasoning and analysis of modern flash SSDs are governed by many *"unwritten contracts"* [15] regarding how user data should be stored, grouped, accessed, and deleted. In parts such expectations are considered unwritten as SSDs themselves are complex with many internal hidden details (flash chip sizes, layout, garbage collection (GC) units, buffer sizes) [30]. Furthermore, the complex architecture of the layered modern storage stack makes the end-to-end reasoning about the unwritten contracts challenging.

NVMe Zone Namespace Devices and ~~Un~~Written Contracts: To address the challenges of complexity, researchers have advocated for more open interfaces [7, 37, 55], which has culminated in the Zoned Namespace (ZNS) specification [5, 44, 50]. Briefly, a ZNS device divides its storage capacity into multiple *zones* that can only be sequentially written (or appended to), thus closely mimicking how flash chips internally work. Data can be read from anywhere either sequentially or randomly. I/O on different zones is isolated and done in parallel, thus a zone represents a unit of parallelism. Beyond these basic read/write operations, ZNS devices have a number of unique flash/ZNS management-related commands. Of particular interest is the *reset* command with zones, which explicitly resets a zone to be written again. In case there was any live data, it is the responsibility of the host software (file system, application) to ensure that the data is copied to a new zone before resetting the old zone. The significant advantage of such a design is that it clearly identifies the unit of garbage collection (zone) and the timing when to reset (under the control of the host software), both of these details are intricately complex to reverse engineer [10, 30]. Hence, with ZNS devices, previously "unwritten contracts" are now standardized *"written contracts"*, which forces us to re-think the operational design of our storage stacks [44].

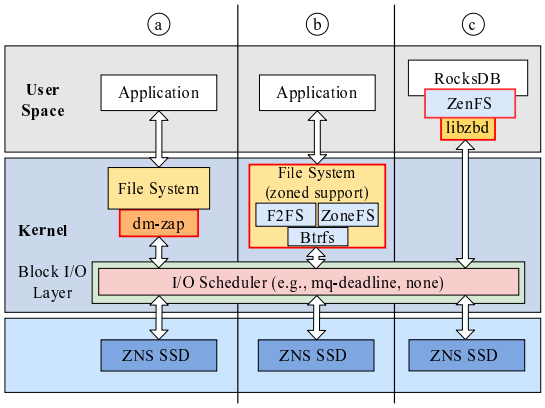


Figure 1: ZNS integration at the (a) block layer; (b) file system level; and (c) application-level (example RocksDB). In all three cases, an end-to-end written contract analysis is missing.

Reasoning about the Written ZNS Contracts with Applications in a Layered Storage Stack: ZNS software support in the storage stack is still under development. There are multiple ways through which ZNS contracts can be extended to an application in an end-to-end manner. Figure 1 shows such possibilities where ZNS software support is needed at the (a) block-level, I/O scheduler support with mq-deadline [9]; or (b) file system level, currently F2FS and Btrfs support ZNS [33]; or (c) application-level direct support, e.g., ZenFS customized file system backend for RocksDB [5]. In such a layered architecture, it is challenging to ensure and cross-check if the ZNS contracts are extended and respected by all layers due to the *semantic gap* between the layers. For example, the POSIX `fadvise` and `fctnl` calls are *hints* to a file system that the file system is free to ignore. Similarly, ZNS/flash file systems themselves have complex bookkeeping operations (garbage collection, log writing, fragmentation) where previously respected hints can be transparently overwritten by a file system without the application's knowledge (see below two examples in contract violation examples). Hence, in this work, we argue, *there is a need to systematically investigate if and how the new written ZNS contracts are extended to applications*.

`zns-tools`: A Framework to Study Written ZNS Contracts: In this work, we propose a set of open-sourced tools called `zns-tools` for the ZNS software stack. Put together, these tools allow us to collect operational data and events across the stack in a programmed manner. The collected data is then analyzed and visualized to identify various contract violations. More specifically, we are interested in understanding the following previously "unwritten contracts" and how they are (not) followed across the layers: (1) "Request Scale" with "Locality": how data placement decisions are made (that result in large I/O, and parallel requests with minimal FTL overheads in ZNS SSDs) and how such allocations look on ZNS SSDs; (2) "Grouping by death time": how data grouping decisions are made to ensure minimal overheads from GC-related data movements and overheads; and (3) "Uniform Data Lifetime": how a ZNS device's lifetime is managed by generating and deleting data with a uniform lifetime, and subsequently explicitly resetting a zone in a ZNS SSD that (indirectly) controls wear-leveling on the device.

Example: ZNS Contract violation with RocksDB and F2FS: Using `zns-tools`, we identify two such violations about data grouping in the state-of-the-practice combination of RocksDB on F2FS with a ZNS SSD. The first issue is F2FS-specific. F2FS reclassifies the hotness (hot, warm, cold) of a data block after a round of GC cycle, where the GC always moves data blocks to cold data. This reclassification allows previously segregated data blocks under different hotness classes to be co-located in a new segment together, thus violating hotness hints from RocksDB. The second issue is RocksDB and F2FS specific. RocksDB writes SSTable to a file system in two passes, raw data (the table) and a small footer (less than a page). After writing the raw SSTable data to hot or cold data, depending on the level of the SSTable , RocksDB synchronizes the file causing F2FS to flush the data to the ZNS device. Upon completion of the flush, RocksDB writes the footer of the SSTable, containing a checksum, which is synchronized to the storage. Due to only a single page for the footer being dirty in the page cache, which is below the minimum threshold for F2FS of 16 pages, F2FS sets an inode flag to indicate the data as being hot. As a result, F2FS writes the footer page to the hot data segment, violating the contract for the SSTables by ignoring any hints for the SSTable writes. Both of these incidents are examples of violating the "Grouping by death time" contract.

In this work we propose a collection of `zns-tools` that combined constitute a framework that allows parts-by-parts building an end-to-end understanding of how data is stored and managed in a layered storage stack on top of ZNS devices. The primary goal of the zns-tools framework is to allow building a complete picture of events across the layers that influence data placement and movement decisions. Figure 4 shows on such end-to-end example generated from our framework. Our key contributions in this work include the following:

- Making a case for building end-to-end operational data analysis tools to reason about previously unwritten, but now written ZNS SSDs storage contracts for applications.
- `zns-tools`, an end-to-end framework that provides support for collecting, analyzing, and visualizing ZNS contracts across the layered storage stack.
- Specific demonstration of the framework's capability on F2FS with RocksDB applications for an end-to-end visualization of operational analytics. Figure 4 shows one such end-to-end example generated from our framework.
- `zns-tools` are open-sourced and available here: https://github.com/stonet-research/zns-tools. This paper is available under CC-BY 4.0 License.

2 Design of `zns-tools` Framework

`zns-tools` is a new framework for ZNS SSDs that is designed to collect operational data about the ZNS storage stack to identify contract violations. It comes with tools that aid in extracting and visualizing: (**C1**) where is user data stored on ZNS SSDs; (**C2**) how much I/O is triggered to each zone; (**C3**) which zones are subjected to GC resets from the software stack and how data migrates across zones because of GC. Currently, there are four tools in the framework, `zns.fiemap`, `zns.segmap`, `zns.imap`, and `zns.trace`. They are visualized in Figure 2, and their exact bash command and outputs are shown in our GitHub repository.

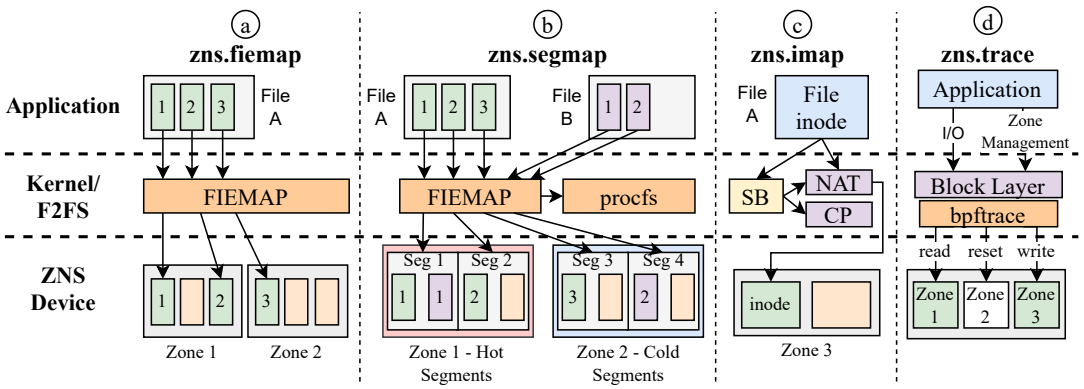


Figure 2: Visual representation of `zns-tools` with (a) `zns.fiemap` mapping individual files to zones; (b) `zns.segmap` mapping multiple files to zones , segments and their lifetime classifications; (c) the `zns.imap` tool mapping the inode of a file to its ZNS zones (d) `zns.trace` tracing I/O and zone management activity to the ZNS zones.

2.1 zns.fiemap

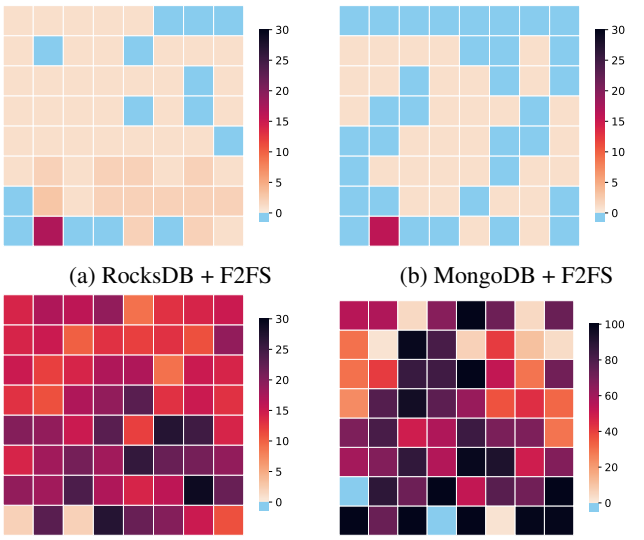
`zns.fiemap` (ZNS file-mapping) is a tool that is designed to extract placement information from the stack to identify how the "Locality" contract is followed. The placement of files and data within modern SSDs is not static, and constantly changes in response to application or file system level events. Applications can typically provide allocation/location hints (by means of data temperature), but it is ultimately the file system that decides about the final data storage location. ZNS SSDs require any live data to be copied during GC, hence, the data location changes. Furthermore, any log-structured file system has its own out-of-place update mechanisms. With ZNS SSDs and log-structured file system, the location of the data is thus dynamic and changes constantly. The repeated execution of `zns.fiemap` traces the file data movement in this dynamic environment to identify how data from different files are grouped in zones. Furthermore, while moving file extents, if the "Locality" rule is not followed, that leads to file fragmentation with holes that are known to cause severe performance degradation [21, 23, 24, 38, 56]. Holes violate the "Request Scale" contract that recommends large sequential I/O requests. In ZNS, how a file is stored among zones also determines the amount of chip-level parallelism, a large file I/O can extract. Hence, `zns.fiemap` provides detailed information about the on-device zone-level placement of files within various file systems (addressing `C1`,`C3`).

`zns.fiemap` retrieves file location mappings from the Linux kernel using the `ioctl()` syscall with the `FIEMAP` flag. Support for `FIEMAP` is not necessary for file systems for POSIX compliance, however, all currently available file systems with ZNS support (F2FS and Btrfs) support this flag. With `FIEMAP`, file systems implement the tracking of extents, representing ranges of physically contiguous data for a file, which are returned to the `ioctl()` caller. By iterating over the logical range of a file, `zns.fiemap` retrieves data mappings of all the extents for the particular file. The collected extents for the targeted file are mapped to their respective zone(s) containing the file's data using their logical 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.fiemap` reports a detailed profile of the extent distribution (min, max, percentiles), hole statistics, and zone-level placement information. Figure 2 (a) illustrates the operational aspect of `zns.fiemap`, retrieving the extent mappings of File `A` using `ioctl()` with the `FIEMAP` flag, followed by mapping the three file extents to zones 1 and 2.

2.2 `zns.segmap` and `zns.imap`

`zns.segmap` and `zns.imap` are tools that collect and quantify metadata around the "Grouping by death time" contract. This contract is implemented within a file system that uses application-level lifetime hints (`RWH_WRITE_LIFE_*` flags with the `fcntl` call), or its own data segregation policies based on data access heatmaps. It recommends that any data that is about to get deleted, or over-written, should be stored within a single GC unit. With this design, once data is invalidated, the GC unit can be simply reset without having to copy any live data, thus reducing GC overheads. Naturally, an accurate grouping requires coordinated efforts from the application, as well as the file system, over a lifetime of the data/file(s). Furthermore, such groupings should be constantly evaluated due to the dynamic nature of data placement with GC events within the file system.

`zns.segmap` and `zns.imap` are tools to extract grouping information for file data and metadata for F2FS. F2FS does heat-based data grouping in *segments*. A file can have its data stored in multiple segments, and a single segment can contain data from multiple files. By default, F2FS uses 3 classes of



(a) RocksDB + F2FS (b) MongoDB + F2FS

(c) PostgreSQL + F2FS (d) RocksDB + (aged) F2FS

Figure 3: Reset visualization using zns.trace for an identical YCSB workload-A for multiple databases on F2FS. Figure (d) has a different heatmap scale (0-100).

classification (hot, warm, cold) on two types of data, file data, and file metadata (inodes). `zns.segmap` extracts the file-to-segment mappings using `zns.fiemap`, and reads the segment hotness classification from `procfs`[1]. To locate the inode in F2FS, `zns.imap` firstly reads the F2FS superblock, which is written at a particular offset within the storage device, followed by parsing the superblock to identify the location of the node address table (NAT), where F2FS stores the block addresses of inodes, and the last checkpoint. Traversing the NAT, `zns.imap` retrieves the block address for the file inode to look up, followed by issuing a single read request to retrieve the inode. The retrieved inode is mapped to the zone in which it is contained. With these tools, we report for each F2FS segment, its hotness classification, number of file extents, the segment-to-zone mapping, and the inode-to-zone mapping. With this information, we can stitch a complete timeline of user data as it ages in the storage system (Figure 4). Though these tools are F2FS-specific, they can be extended to other file systems that do active data segregation. With F2FS, this information is available in the `/proc` file system[2]. The file system-related metadata is also available as a part of low-level libraries (such as libext2fs or libf2fs) or can be generated [45]. Figure 2 (b) and (c) illustrate the design of `zns.segmap` and `zns.imap` tools. They retrieve extents for various files (`ioctl()` call) and inodes (FS metadata walks) from F2FS, and retrieve F2FS segment lifetime classifications and the inode location.

2.3 `zns.trace`

The `zns.trace` tool identifies performance-critical "Request Scale" and "Uniform Data Lifetime" contracts by tracking ZNS command calls. `zns.trace` is a combination of a BPFtrace [19] script and a plotting tool that traces zone-level ZNS write, append, read, and reset operations (addressing `C2`). Tracing such data is useful for investigating the access patterns to the underlying storage device independently of file system or application workloads.

`zns.trace` utilizes BPFtrace [19], 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 I/O event with the command I/O sizes (their histogram) and zone reset calls, and maps the events to their corresponding zone(s). By analyzing the function arguments, the script identifies the type of operation, and further extracts data fields based on the request size. Importantly, the tracing supports ZNS devices in VMs (apart from just the host), where the zone reset command sets the function argument for the type of command to `REQ_OP_DRV_OUT`, indicating the host driver (e.g., vfio-pci when using NVMe passthrough to the VM) is responsible for the request. This case requires to furthermore analyze the NVMe command and its zone management command field on the type of zone management action (e.g., close, finish, reset, open, offline). `zns.trace` relies on inserting kernel probes to parse I/O functions, and is therefore currently limited to kernel-based ZNS tracing. However, ongoing work is extending the tracing framework for user-level (e.g., SPDK) I/O and broader zone management request tracing.

To illustrate the utility of `zns.trace` we run an identical workload on a number of database/file system and quantify the workload reset profiles. The number of resets is directly linked to the number of reset cycles that an SSD can undergo before exhausting flash chips. The "Uniform Data Lifetime" contract recommends generating data with a uniform lifetime to evenly spread the device wear. 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 a YCSB workload-A (50% update, 50% read) [8] on RocksDB [13](7.4.3), MongoDB [35](6.06), and PostgreSQL [32](9.6.24) as KV backend targets with F2FS as file system. Figure 3 shows our results. Here each cell represents a zone, and the color indicates the total number of resets issued to the zone (since startup). Blue squares indicate a zone to which no reset commands were issued at all. There are two interesting observations. Firstly, for an identical workload, the three KV backends show vastly different profiles. We can see that for this test configuration PostgreSQL leads to significantly more resets. Additionally, we can identify that one of the bottom left zones (zone 2) is in all three cases heavily utilized. This zone corre-

[1] `/proc/fs/f2fs/nvme0n1/segment_info`

[2] We also tried Brtfs with ZNS, but it was unstable.

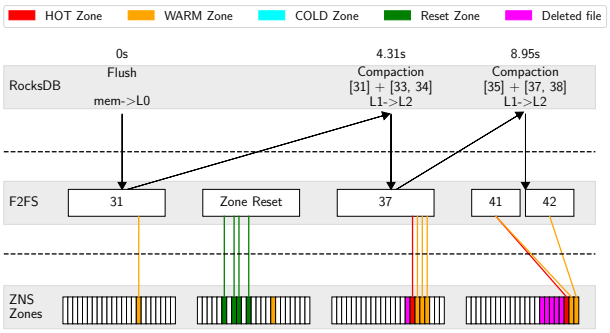


Figure 4: (Abridged) End-to-end event timeline visualized using `zns-tools`.

sponds 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 3 (d) which has an aged F2FS (10× iterations). Both of these results make a case for a systematic study of how zone resets are called or consequently, how data grouping by uniform lifetime or death time is done.

2.4 RocksDB on F2FS with ZNS Timeline

To demonstrate an end-to-end utility of `zns-tools`, we build an end-to-end data placement visualization using RocksDB on F2FS. Using the RocksDB's db_bench benchmark with workloads *fillrandom* and *overwrite* fills the F2FS file system on the ZNS device. Minor modifications (less than 100 LOCs) to the RocksDB source code are made to trace the activity of operations, and retrieve the data mapping of the generated files. On each compaction or flush operation, RocksDB calls the `zns-tools` to map all its files. Figure 4 illustrates this end-to-end timeline (automatically generated from traces) with a few salient events across the layers (applications, file systems, and ZNS device) over the lifetime of an SSTable. The timeline starts at time 0, where data is flushed from memory to level-0 (abbreviated L0) as SSTable 31. The file is stored in the orange zone classified as WARM. Subsequently, F2FS issues a series of reset calls on several zones. The next logical event is the compaction of file 31 with 33 and 34 to generate file 37 on L2[3]. While file 37 is written as WARM file, old files in prior zones are deleted (pink lines). In the next round, file 37 is picked up for compaction with 35 and 38 files to generate two L2 files (41, and 42). Such a timeline visualization gives an understanding of how the different RocksDB operation interaction with F2FS affects the utilization of the ZNS storage space, file classification, and data movement over time, thus making it easy to spot contract violations visually.

[3]internal trivial promotion events such as moving 31 from L0 to L1 are skipped in the visualization for the sake of clarity.

3 Related Work

Flash SSDs with their complex internal logic and *"unwritten contracts"* [15] have also been studied in detail for performance and operation characterizations [18, 22, 25, 26, 30, 31] and with the impact of GC operations [17, 27, 39, 47, 48]. There is a rich history of collecting file system traces, analyzing, and replaying them for understanding the impact of optimizations [3, 12, 20, 36, 42, 46]. Much of these works only focus on basic read/write interfaces that are sufficient for HDDs, but not SSDs. None of these tools track device management-related operations (reset, open, finish, close), which are now a critical part of ZNS devices. The libzbd ZNS library also support a basic GUI visualization tool for zone states [2], however, it lacks any application or file system level information. Beyond these tools, the eBPF-based BCC framework has emerged as the de-facto API for writing complex, end-to-end tracking frameworks [1], which we also leverage. Similar to `zns-tools`, IOScope uses eBPF-assisted file offset-based I/O tracing [43]. 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. Several block-level tracking tools exist (BCC's biotop and biosnoop, DTraces's iosnoop), however, they do not link block-level I/O back to the file system or applications. MapFS is a file system interface that presents low-level details from file to storage mappings for applications to manipulate data-heavy operations via light-weight metadata operations [51]. Prabhakaran et al. [40] introduce techniques to study file system behavior with semantic knowledge of events and on-disk data structure layouts. `zns-tools` extends such motivation to include workloads with the new ZNS management operations as well. HintStore is a flexible framework that is designed to explore the effectiveness of hints with heterogeneous storage [14]. `zns-tools` analysis captures the after-effect of the hints, and its collected data can be used to verify if hints are implemented in an end-to-end manner. In a distributed setting, systems like Wintermute [34], Apollo [41], and Beacon [54] provide a distributed framework for operational data collection and analysis. In comparison to these works, the focus of `zns-tools` is on collecting, analyzing, and visualizing written ZNS contracts across multiple storage stack layers to reason about previously unwritten SSD storage contracts.

4 Conclusion and Ongoing Work

In this work, we have presented a case to systematically reason about previously "*unwritten contracts*" for flash SSDs on recently emerged ZNS flash SSD devices. Due to the unique I/O and management interface of ZNS SSDs, they make such unwritten contracts, written by putting them under the control of the host storage stack. To investigate the effectiveness of ZNS contracts, we have developed and presented a set of `zns-tools` to collect, analyze, and visualize ZNS operational

data. We are working on tracing events from PostgreSQL, Aerospike, and scientific workloads on top of BrtFS also, to develop a broader framework in which such end-to-end contract analysis can be done.

Acknowledgment: This work in-parts is supported by the donations from Western Digital.

Availability

https://github.com/stonet-research/zns-tools.

References

[1] BPF Compiler Collection (BCC). https://github.com/iovisor/bcc, 2023. Accessed: 2023-29-03.

[2] libzbd User Library. https://zonedstorage.io/docs/tools/libzbd, 2023. Accessed: 2023-29-03.

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

[4] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobbler, Michael Wei, and John D. Davis. CORFU: A shared log design for flash clusters. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 1–14, San Jose, CA, April 2012. USENIX Association.

[5] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. ZNS: Avoiding the block interface tax for flash-based SSDs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 689–703. USENIX Association, July 2021.

[6] Matias Bjørling, Jens Axboe, David Nellans, and Philippe Bonnet. Linux block io: Introducing multi-queue ssd access on multi-core systems. In *6th International Systems and Storage Conference*, SYSTOR 13. ACM, 2013.

[7] Matias Bjørling, Javier González, and Philippe Bonnet. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies*, FAST'17, page 359–373, Santa clara, CA, USA, 2017.

[8] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *ACM Symposium on Cloud Computing (SoCC)*, page 143–154, 2010.

[9] Western Digital. Zoned Storage - Write Ordering Control. https://zonedstorage.io/docs/linux/sched, 2023. Accessed: 2023-29-03.

[10] Krijn Doekemeijer, Nick Tehrany, Balakrishnan Chandrasekaran, Matias Bjørling, and Animesh Trivedi. Performance characterization of nvme flash devices with zoned namespaces (zns). In *(to appear) IEEE International Conference on Cluster Computing, CLUSTER 2023, October 31-November 3, 2023, Santa Fe, New Mexico, USA*. IEEE, 2023.

[11] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. Evolution of development priorities in key-value stores serving large-scale applications: The rocksdb experience. In *FAST*, pages 33–49, 2021.

[12] Daniel Ellard and Margo Seltzer. New NFS tracing tools and techniques for system analysis. In *17th Large Installation Systems Administration Conference (LISA 03)*, San Diego, CA, October 2003. USENIX Association.

[13] Facebook. RocksDB: A Persistent Key-Value Store for Flash and RAM Storage. https://github.com/facebook/rocksdb. Accessed: 2023-03-01.

[14] Xiongzi Ge, Zhichao Cao, David H. C. Du, Pradeep Ganesan, and Dennis Hahn. Hintstor: A framework to study I/O hints in heterogeneous storage. *ACM Trans. Storage*, 18(2):18:1–18:24, 2022.

[15] Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. The unwritten contract of solid state drives. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, page 127–144, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Mohammad Hedayati, Kai Shen, Michael L. Scott, and Mike Marty. Multi-queue fair queueing. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '19, page 301–314, USA, 2019. USENIX Association.

[17] Jian Hu, Hong Jiang, and Prakash Manden. 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*, SIGMETRICS '12, page 415–416, New York, NY, USA, 2012. Association for Computing Machinery.

[18] H. Howie Huang, Shan Li, Alex Szalay, and Andreas Terzis. Performance modeling and analysis of flash-based storage devices. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11, 2011.

[19] iovisor. Bpftrace. https://github.com/iovisor/bpftrace. Accessed: 2023-29-03.

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

[21] Cheng Ji, Li-Pin Chang, Liang Shi, Chao Wu, Qiao Li, and Chun Jason Xue. An empirical study of File-System fragmentation in mobile storage systems. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, Denver, CO, June 2016. USENIX Association.

[22] Myoungsoo Jung and Mahmut Kandemir. Revisiting widely held ssd expectations and rethinking system-level implications. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, page 203–216, New York, NY, USA, 2013. Association for Computing Machinery.

[23] Saurabh Kadekodi, Vaishnavh Nagarajan, and Gregory R. Ganger. 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)*, pages 691–704, Boston, MA, 2018. USENIX Association.

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

[25] Jihun Kim, Joonsung Kim, Pyeongsu Park, Jong Kim, and Jangwoo Kim. Ssd performance modeling using bottleneck analysis. *IEEE Computer Architecture Letters*, 17(1):80–83, 2018.

[26] Joonsung Kim, Kanghyun Choi, Wonsik Lee, and Jangwoo Kim. Performance modeling and practical use cases for black-box ssds. *ACM Trans. Storage*, 17(2), jun 2021.

[27] Tomer Lange, Joseph (Seffi) Naor, and Gala Yadgar. Offline and online algorithms for ssd management. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(3), dec 2021.

[28] Changman Lee, Dongho Sim, Joo-Young Hwang, and Sangyeun Cho. F2fs: A new file system for flash storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, page 273–286, USA, 2015. USENIX Association.

[29] Gyusun Lee, Seokha Shin, Wonsuk Song, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. Asynchronous i/o stack: A low-latency kernel i/o stack for ultra-low latency ssds. In *USENIX Annual Technical Conference*, USENIX ATC 19, page 603–616. USENIX Association, 2019.

[30] Nanqinqin Li, Mingzhe Hao, Huaicheng Li, Xing Lin, Tim Emami, and Haryadi S. Gunawi. Fantastic ssd internals and how to learn and use them. In *Proceedings of the 15th ACM International Conference on Systems and Storage*, SYSTOR '22, page 72–84, New York, NY, USA, 2022. Association for Computing Machinery.

[31] Shan Li and H. Howie Huang. Black-box performance modeling for solid-state drives. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 391–393, 2010.

[32] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[33] Aota Naohiro. Btrfs: Zoned block device support [lwn.net]. https://lwn.net/Articles/833260/, oct 2020. Accessed: 2023-29-03.

[34] Alessio Netti, Micha Müller, Carla Guillen, Michael Ott, Daniele Tafani, Gence Ozer, and Martin Schulz. 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*, HPDC '20, page 101–112, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Trong-Dat Nguyen and Sang-Won Lee. Optimizing mongodb using multi-streamed ssd. In *Proceedings of the 7th International Conference on Emerging Databases: Technologies, Applications, and Theory*, pages 1–13. Springer, 2018.

[36] John K. Ousterhout, Hervé Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson. A trace-driven analysis of the unix 4.2 bsd file system. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, SOSP '85, page 15–24, New York, NY, USA, 1985. Association for Computing Machinery.

[37] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. Sdf: Software-defined flash for web-scale internet storage systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, page 471–484, New York, NY, USA, 2014. Association for Computing Machinery.

[38] Jonggyu Park and Young Ik Eom. Fragpicker: A new defragmentation tool for modern storage devices. In Robbert van Renesse and Nickolai Zeldovich, editors, *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 280–294. ACM, 2021.

[39] Roman Pletka, Ioannis Koltsidas, Nikolas Ioannou, Saša Tomić, Nikolaos Papandreou, Thomas Parnell, Haralampos Pozidis, Aaron Fry, and Tim Fisher. Management of next-generation nand flash to achieve enterprise-level endurance and latency targets. *ACM Trans. Storage*, 14(4), dec 2018.

[40] Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Analysis and evolution of journaling file systems. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 8, USA, 2005. USENIX Association.

[41] Neeraj Rajesh, Hariharan Devarajan, Jaime Cernuda Garcia, Keith Bateman, Luke Logan, Jie Ye, Anthony Kougkas, and Xian-He Sun. Apollo: An ml-assisted real-time storage resource observer. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '21, page 147–159, New York, NY, USA, 2021. Association for Computing Machinery.

[42] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson. A comparison of file system workloads. In *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, San Diego, CA, June 2000. USENIX Association.

[43] Abdulqawi Saif, Lucas Nussbaum, and Ye-Qiong Song. Ioscope: A flexible i/o tracer for workloads' i/o pattern characterization. In Rio Yokota, Michèle Weiland, John Shalf, and Sadaf Alam, editors, *High Performance Computing*, pages 103–116, Cham, 2018. Springer International Publishing.

[44] Theano Stavrinos, Daniel S. Berger, Ethan Katz-Bassett, and Wyatt Lloyd. Don't be a blockhead: Zoned namespaces make work on conventional ssds obsolete. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '21, page 144–151, New York, NY, USA, 2021. Association for Computing Machinery.

[45] Kuei Sun, Daniel Fryer, Joseph Chu, Matthew Lakier, Angela Demke Brown, and Ashvin Goel. Spiffy: Enabling File-System aware storage applications. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 91–104, Oakland, CA, February 2018. USENIX Association.

[46] Vasily Tarasov, Santhosh Kumar, Jack Ma, Dean Hildebrand, Anna Povzner, Geoff Kuenning, and Erez Zadok. Extracting flexible, replayable models from large block traces. In William J. Bolosky and Jason Flinn, editors, *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*, page 22. USENIX Association, 2012.

[47] Benny Van Houdt. 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*, SIGMETRICS '13, page 191–202, New York, NY, USA, 2013. Association for Computing Machinery.

[48] Robin Verschoren and Benny Van Houdt. On the endurance of the d-choices garbage collection algorithm for flash-based ssds. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 4(3), jul 2019.

[49] Don H Walker. A comparison of nvme and ahci. https://sata-io.org/sites/default/files/documents/NVMe%20and%20AHCI_%20_long_.pdf, Accessed: 2022-05-02.

[50] Western Digital. Ultrastar dc zn540. https://www.westerndigital.com/products/internal-drives/data-center-drives/ultrastar-dc-zn540-nvme-ssd, Accessed: 2022-05-02.

[51] Jake Wires, Mark Spear, and Andrew Warfield. Exposing file system mappings with MapFS. In *3rd Workshop on Hot Topics in Storage and File Systems (HotStorage 11)*, Portland, OR, June 2011. USENIX Association.

[52] Jiwon Woo, Minwoo Ahn, Gyusun Lee, and Jinkyu Jeong. D2FQ: device-direct fair queueing for nvme ssds. In Marcos K. Aguilera and Gala Yadgar, editors, *19th USENIX Conference on File and Storage Technologies, FAST 2021, February 23-25, 2021*, pages 403–415. USENIX Association, 2021.

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

[54] 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. End-to-end I/O monitoring on a leading supercomputer. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 379–394, Boston, MA, February 2019. USENIX Association.

[55] Jingpei Yang, Rajinikanth Pandurangan, Changho Choi, and Vijay Balakrishnan. Autostream: Automatic stream management for multi-streamed ssds. In *Proceedings of the 10th ACM International Systems and Storage Conference*, SYSTOR '17, New York, NY, USA, 2017. Association for Computing Machinery.

[56] Lihua Yang, Fang Wang, Zhipeng Tan, Dan Feng, Jiaxing Qian, and Shiyun Tu. Ars: Reducing f2fs fragmentation for smartphones using decision trees. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, DATE '20, page 1061–1066, San Jose, CA, USA, 2020. EDA Consortium.