ZoneTrace: Zone Monitoring Tool for F2FS on ZNS SSDs

PING-XIANG CHEN, University of California Irvine, Irvine, United States
DONGJOO SEO, University of California Irvine, Irvine, USA
CHANGHOON SUNG, Kookmin University, Seoul, Republic of Korea
JONGHEUM PARK, Kookmin University, Seoul, Republic of Korea
MINCHUL LEE, Kookmin University, Seoul, Republic of Korea
HUAICHENG LI, Virginia Tech, Blacksburg, United States
MATIAS BJØRLING, Western Digital Corporation Irvine, Irvine, United States
NIKIL DUTT, University of California, Irvine, Irvine, United States

We present ZoneTrace, a runtime monitoring tool for the Flash-friendly File System (F2FS) on Zoned Namespace (ZNS) Solid-state Drives (SSDs). ZNS SSD organizes its storage into zones of sequential write access. Due to ZNS SSD's sequential write nature, F2FS is a log-structured file system that has recently been adopted to support ZNS SSDs. To present the space management with the zone concept between F2FS and the underlying ZNS SSD, we developed ZoneTrace, a tool that enables users to visualize and analyze the space management of F2FS on ZNS SSDs. ZoneTrace utilizes the extended Berkeley Packet Filter (eBPF) to trace the updated segment bitmap in F2FS and visualize each zone space usage accordingly. Furthermore, ZoneTrace is able to analyze on file fragmentation in F2FS and provides users with informative fragmentation histogram to serve as an indicator of file fragmentation. Using ZoneTrace's visualization, we are able to identify the current F2FS space management scheme's inability to fully optimize space for streaming data recording in autonomous systems, which leads to serious file fragmentation on ZNS SSDs. Our evaluations show that ZoneTrace is lightweight and assists users in getting useful insights for effortless monitoring on F2FS with ZNS SSD with both synthetic and realistic workloads. We believe ZoneTrace can help users analyze F2FS with ease and open up space management research topics with F2FS on ZNS SSDs.

CCS Concepts: • **Computer systems organization → Embedded software;**

Additional Key Words and Phrases: F2FS, zoned namespace SSDs, visualization

ACM Reference Format:
Ping-Xiang Chen, Dongjoo Seo, Changhoon Sung, Jongheum Park, Minchul Lee, Huaicheng Li, Matias Bjørling, and Nikil Dutt. 2024. ZoneTrace: Zone Monitoring Tool for F2FS on ZNS SSDs. *ACM Trans. Des. Autom. Electron. Syst.* 29, 5, Article 85 (September 2024), 15 pages. https://doi.org/10.1145/3656172

This work is supported by J. Yang & Family Foundation Fellowship.
Authors. addresses: P.-X. Chen, 4208 Palo Verde Rd, Irvine, CA 92617; e-mail: p.x.chen@uci.edu; D. Seo, Computer Science, University of California, Irvine, California, United States; e-mail: dseo3@uci.edu; C. Sung, J. Park, and M. Lee, Kookmin University, Seoul, Republic of Korea; e-mails: whtiesilver@kookmin.ac.kr, parkjh0821@kookmin.ac.kr, drminchul@kookmin.ac.kr; H. Li, Virginia Tech, Blacksburg, Virginia, United States; e-mail: huaicheng@cs.vt.edu; M. Bjørling, Western Digital Corporation Irvine, Irvine, California, United States; e-mail: Matias.Bjorling@wdc.com; N. Dutt, Computer Science, University of California, Irvine, Irvine, California, United States; e-mail: dutt@ics.uci.edu.

This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).
ACM 1084-4309/2024/09-ART85
https://doi.org/10.1145/3656172

1 INTRODUCTION

Zoned Namespace (ZNS) Solid-state Drive (SSD) was introduced for mitigating the performance drawbacks associated with the conventional block interface for flash-based SSDs [4]. ZNS SSDs divide the storage into fixed-size zones, which enables the user to align the zones to its own media characteristics. Each zone can be read randomly but need to be written sequentially. Moreover, a zone has to be reset to reuse it. ZNS SSDs are considered new innovations due to their ability to maintain a low **Write Amplification Factor (WAF)**, meaning that the amount requested by the system and the amount of data written on the actual device is similar, which provides users with predictive product life and performance consistency. The sequential-write constraint of ZNS SSDs makes log-structured file systems such as the **Flash-friendly File System (F2FS)** [21] suitable for ZNS SSDs. To be in compliance with the sequential-write constraint of ZNS SSDs, F2FS enables pure **Log-File Structure (LFS)** [30, 33] mode for forcing sequential writes to segments and the sequential usage of segments within sections. Also, for supporting ZNS SSD, F2FS aligns a section that is a management unit to match the zone size in the device. While several efforts have addressed space management of F2FS for SSDs [6, 28, 35, 37], their works lack the ability to visualize zone information when mounting F2FS on ZNS SSDs. This lack of information on F2FS with ZNS SSDs is an obstacle for researchers to understand research topics related to file system space management. Therefore, researchers critically need a tool to help them gain insight about zone-based space usage for ZNS SSDs.

We present `ZoneTrace`, a monitoring tool that enables visualization of the space management on F2FS with the zone concept at runtime. `ZoneTrace` utilizes eBPF [13, 23, 34] to track the updated metadata in F2FS and uses the system call to retrieve the file mapping information to the user space. The file mapping information includes the logical to physical block address translation of a file when using F2FS. ZoneTrace utilizes these information for the F2FS file system layout and displays the space usage onto each zone in ZNS SSDs through our visualization module. Last but not least, to help users analyze file fragmentation in F2FS using ZNS SSDs, ZoneTrace offers a range of fragmentation scenarios that enables users to engage in file layout manipulation, considering different fragment sizes and distances. `ZoneTrace` allows users to analyze F2FS at the zone level for further optimization of the space management policy with F2FS on ZNS SSDs in various research topics, such as file system aging [5, 8, 9, 16, 17] and host-managed garbage collection [7, 33].

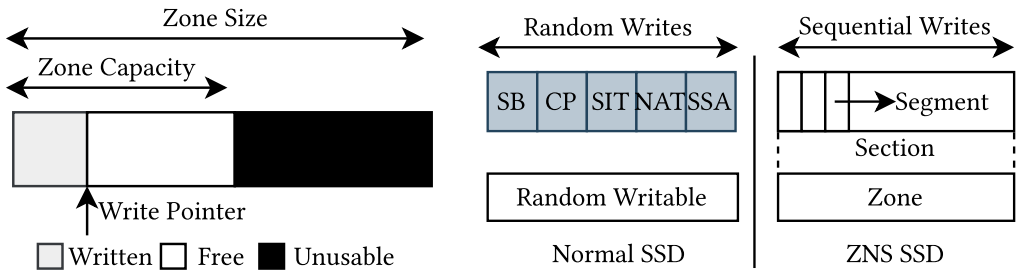
Our contributions are as follows:

- We present `ZoneTrace`, the first software tool capable of monitoring and visualizing space usage and mapping file information within each zone at runtime for F2FS on ZNS SSDs.
- We present a new scheme that assists users to manage file layout and present a fragment size histogram to succinctly capture the fragmentation issue in F2FS with ZNS SSDs.
- We present novel insights into how current scheme of F2FS could incur serious file fragmentation for storing data streams generated from autonomous systems using `ZoneTrace`.
- We evaluate `ZoneTrace` based on F2FS on a real ZNS SSD. `ZoneTrace` incurs low overhead and therefore enables users to trace and visualize the space usage with the zone concept, allowing users to effectively analyze F2FS on ZNS SSDs with ease.

2 BACKGROUND AND MOTIVATION

2.1 Zoned Namespace (ZNS) SSD

The introduction of the ZNS interface avoids the performance limitations associated with the existing block interface in traditional flash-based SSDs. Compared to traditional block-based SSDs, ZNS SSDs organize logical blocks into zones, requiring each zone to be written sequentially. As shown in Figure 1(a), the size of a zone is defined as the total number of logical blocks within the zone.



(a) Zone Size and Zone Capacity

(b) F2FS with ZNS SSD

Fig. 1. ZNS SSD and flash friendly file system (F2FS).

Each zone in the device address space has a write pointer that tracks the next writable LBA within each zone. ZNS SSDs also define a zone capacity attribute for each zone, indicating the number of usable logical blocks in the zone, starting from the first logical block of each zone. The zone capacity attribute is either equal to or smaller than the zone size. Employing zone capacities that differ from the zone size allows for the zone size to remain constant for all zones while optimizing the mapping of a zone capacity to the underlying media characteristics. F2FS and BTRFS [29] are currently the two POSIX compliant file systems that support ZNS SSDs. Although BTRFS currently supports zoned block devices, the support is still experimental [27], while F2FS is relatively mature. Therfore in this study, we choose F2FS as the target file system with ZNS SSDs.

2.2 F2FS with ZNS SSDs

F2FS is a widely adopted file system designed for flash-based SSDs, partitioning the storage space into regions dedicated to random writes for metadata-block and multi-stream sequential writes main data area. As shown in Figure 1(b), a multi-device setup is required for using ZNS SSDs with F2FS to place the metadata blocks on random writable storage. The metadata blocks area contains, **Superblock (SP)**, **CheckPoint (CP)**, **Segment Information Table (SIT)**, **Node Address Table (NAT)**, and **Segment Summary Area (SSA)**, containing necessary information on the file system data layout. F2FS divides the whole volume into fixed-size 2 MB segments. A group of segments is called a section. The number of segments in a section is determined to match the zoned device zone size. For example, with a 128 MB zone size, a section contains 64 segments. By default, F2FS tries to optimize block allocation by allowing some random writes within segments. Nevertheless, to be in full compliance with the zoned block device's sequential write requirement, F2FS needs to run in **Log-structured File System (LFS)** mode, which forces only sequential writes to segments and the sequential use of segments within sections.

2.3 Related Works

Several works have been proposed to trace the space management of F2FS for flash-based SSDs. They aim to understand the space management and trace related issues such as file fragmentation within F2FS. File fragmentation refers to the situation where a file consists of several blocks that are not located continuously. File fragmentation has a negative impact on I/O performance, since fragmented files require more management overhead to keep track of the scattered data blocks. Park et al. [28] manipulated F2FS with various predefined fragmentation sizes and fragmentation distances using direct I/O with block-based SSDs. Nevertheless, with F2FS on ZNS SSDs, to be in full compliance with sequential-write constraints of ZNS SSDs, F2FS with zoned storage is forced

Table 1. Limitations of Related Works

	File Layout Manipulation	Zone Aware	Visualization	Fragmentation Metrics
FragPicker [28]	✓	✗	✗	✓
FragTracer [6]	✗	✗	✓	✗
zns-tools [35]	✗	✓	✗	✗
ZoneTrace	✓	✓	✓	✓

```
shawn@drg-storage:~/Workspace$ sudo cat /proc/fs/f2fs/nvme4n1p1/segment_bits
format: segment_type|valid_blocks|bitmaps
segment_type(0:HD, 1:WD, 2:CD, 3:HN, 4:WN, 5:CN)
0         3|1  | 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
2         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
4         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
5         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
7         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9         0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10        0|0  | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fig. 2. Output of /proc interface of space management of F2FS with a 1 TB ZNS SSD. The command took 21 s to dump the information.

to use buffered I/O [36] for writes. As a result, when users need to reevaluate how fragmentation size and distances are affected with F2FS on ZNS SSDs, their file manipulation scheme must consider the F2FS buffered I/O scheme with ZNS SSDs. Cho and Kang [6] propose a real-time fragmentation monitoring tool that helps users analyze the ratio of F2FS fragmentation in real-time with visualization. However, their tool is unable to visualize each fragment with the zone concept from ZNS SSDs. Tehrany et al. [35] introduced a tool aimed at aiding users in examining the utilization of F2FS space usage on ZNS SSDs. However, their tool lacks runtime monitoring and visualization capabilities, making it less user-friendly. ZoneTrace overcomes these limitations of related work as summarized in Table 1.

2.4 Motivation

Currently F2FS provides users with a very rudimentary interface that dumps space management information through the `/proc` file system interface, as shown in Figure 2. This interface is not easily readable by a human, and moreover is lacking the zone concept. Indeed, existing works are unable to visualize each zone's space usage with F2FS on ZNS SSDs, which is crucial information to understand the data management. Developers need a tool to help them visualize the data that is more intuitive than text-based raw data and monitor zone-based space usage for ZNS SSDs at runtime. Furthermore, existing works are unable to manipulate the storage space due to the buffered I/O scheme for writes when using F2FS on ZNS SSDs. To address these issues, we present `ZoneTrace`, a runtime monitoring and visualization tool for F2FS on ZNS SSD. `ZoneTrace` can assist users in understanding the space management in F2FS for each zone of ZNS SSDs. `ZoneTrace` also provides users with a new scheme for manipulating the fragmentation in F2FS zoned mode and proposes a fragmentation histogram as a fragmentation metric for each file on F2FS with ZNS SSDs.

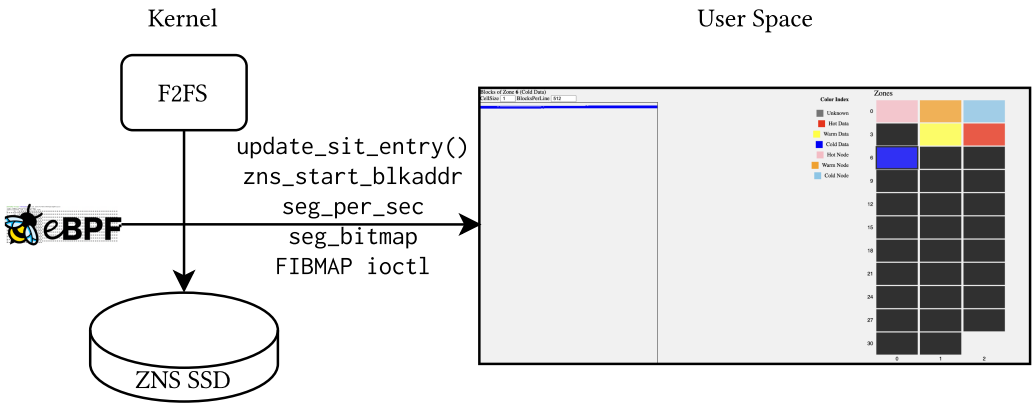


Fig. 3. Overview of ZoneTrace

3 ZONETRACE

3.1 Overview

Figure 3 presents an overview of `ZoneTrace`. As shown in Figure 3, `ZoneTrace` is able to trace specific functions and data structures in F2FS and visualize the zone usage in ZNS SSDs accordingly at runtime under the server-client architecture. We first modified `f2fs-tools`, tools allow users to format, check, repair, and manage the F2FS file system, for retrieving the ZNS device information and the file system layout when mounting F2FS on ZNS SSDs. Then, we implemented an eBPF program that traces `update_sit_entry` in F2FS and exports the F2FS segment kernel data structure to the user space. eBPF is a powerful and flexible framework in the Linux kernel that allows for the dynamic tracing and manipulation on various kernel events. Our eBPF program traces the updated events and transfers the updated valid bitmap in binary format to the standard output. To enhance the portability our eBPF program, we enable BPF **Compile Once—Run Everywhere (CO-RE)** that enables the eBPF program to run across different kernel versions and architectures. Third, we also make use of the `FIBMAP` ioctl system call to query the file mapping of each file and calculate the mapping to zone on ZNS SSDs. A server is implemented to capture the standard output from the ePBF program and the `FIBMAP` ioctl system call via a pipe command. Based on the information that we capture during file system mounting time, we can calculate and visualize the file mapping. Then, the server transfers the decoded data structure for rendering onto the web browser client via http. To help users evaluate metrics for file fragmentation, we further propose a fragmentation histogram accordingly during runtime. Finally, the client visualizer renders the space management of F2FS in each zone on the ZNS SSD.

ZoneTrace is publicly available via this link: https://github.com/pingxiang-chen/bpf-f2fs-zonetrace.git. The comments in each source file of ZoneTrace contains detailed data structure implementation information. For example, the implementation eBPF program that traces the updated event from F2FS is located at src/bpf/f2fszonetracer.bpf.c and the main function of the eBPF program is at src/bpf/f2fszonetracer.c. The server client visualization architecture is implemented within the folder viewer/

3.2 Obtain F2FS File Layout with Zone Information

To obtain the corresponding F2FS file system layout on ZNS SSDs, we modify f2fs-tools to obtain structure `device_info`, which contains the logical data layout of ZNS SSD with F2FS. The

```
----------------bdev info-------------------
devname = 0000:00
start_lba = 0

----------------file info-------------------
/mnt/f2fs/target_file.png :
--------------------------------------------
dev       [259:2]
ino       [0x       4 : 4]
mode      [0x    81a4 : 33188]
nlink     [0x       1 : 1]
uid       [0x       0 : 0]
gid       [0x       0 : 0]
size      [0x   fe000 : 1040384]
blksize   [0x    1000 : 4096]
blocks    [0x     7f0 : 2032]
--------------------------------------------

file_pos   start_blk     end_blk        blks
       0     4194304     4194430         127
  520192     4194687     4194813         127
```

Fig. 4. Example output of the `fibmap.f2fs` applications from f2fs-tools. `file_pos` means the logical file offset in the created file. `start_blk` means the physical block number from logical to the physical block address. `blks` stands for the number of contiguous blocks starting from the physical block number.

`device_info` includes the start block address, the number of zones, the number of blocks within a zone, and the number of unusable blocks within each zone of the mounted ZNS SSD. With the information above, we can determine the start segment number in F2FS mounted on ZNS SSDs based on Equation (1). Since F2FS aligns the number of segments in a section to match the zoned device zone size, we can determine the number of segments in each section by Equation (2). With the starting segment number of F2FS and the number of segments per zone, we can trace the updated segments in F2FS of each zone, visualizing each zone's space usage in F2FS on ZNS SSD:

$$start_seg_num = zns_start_blkaddr/blks_per_seg, \tag{1}$$

$$seg_per_sec = zone_blocks/blks_per_seg. \tag{2}$$

To further visualize how F2FS files data layout on ZNS SSDs, we additionally make use of the `FIBMAP` [20] (stands for "Filesystem Index Block Map") ioctl system call from Linux. `FIBMAP` system call is primarily used to query the file system for details about the physical location of data blocks associated with a specific file or range of a file. Figure 4 shows the sample output of the implemented `fibmap.f2fs` from the f2fs-tools [18] application queries the extent mapping of the file `/mnt/f2fs/target_file.png`. As shown in Figure 4, the `FIBMAP` system call returns a list of extents, where each extent represents a contiguous block of data on the storage device. For example, the logical file offset 0 in file `/mnt/f2fs/target_file.png` is mapped to physical block number 4194304 with a contiguous length of 127 4 KB blocks. Then, we can determine the first extent start block number of the file `/mnt/f2fs/target_file.png` into zone number on ZNS SSD based on Equation (3), and the block offset in zone based on Fomula 4 and visualize each extent from file mapping with the length information:

$$zone_num = (strat_blk - zns_start_blkaddr)/blks_per_sec, \tag{3}$$

$$blknum_in_zone = (strat_blk - zns_start_blkaddr)\%blks_per_sec. \tag{4}$$

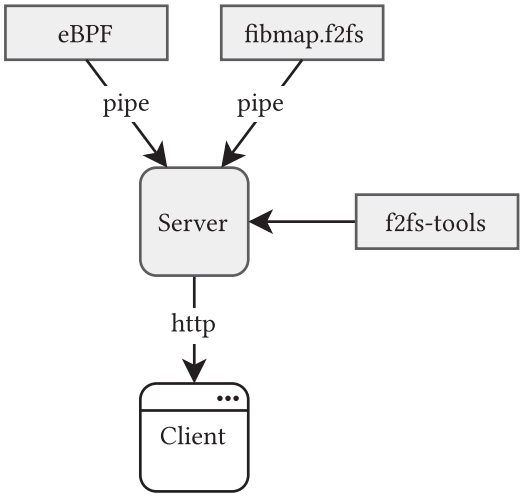


Fig. 5. Server client architecture of ZoneTrace

ALGORITHM 1: Event Transmission

```
set 0.2 sec timer
while server is running do
    if update_event occurs then
        if update_zone != select_zone then
            sends update_zone to client
        else
            store valid bitmap
            mark updated segments
        end if
    end if
    if timer then
        retrieves the bitmap
        sends the bitmap to the client
    end if
end while
```

3.3 Portability

Although we use the eBPF to trace the update segment function in F2FS, the limitation of using BPF programs is a lack of portability, since the data structures and types are not guaranteed to be the same between kernel versions. To address the issue, we enable BPF CO-RE [25] for better portability of our program. The Linux kernel should be compiled with both the `CONFIG_DEBUG_INFO_BTF` and `CONFIG_DEBUG_INFO_BTF_MODULES` enabled for generating deduplicated **BPF-type Format (BTF)** [3] type information from the kernel. BTF is a file format that provides better introspection into the program by encoding the type information of a BPF program. By default, F2FS is compiled as a kernel module. We use `bpftool` to dump required BTF information to a header file with F2FS BTF structures using the following command:

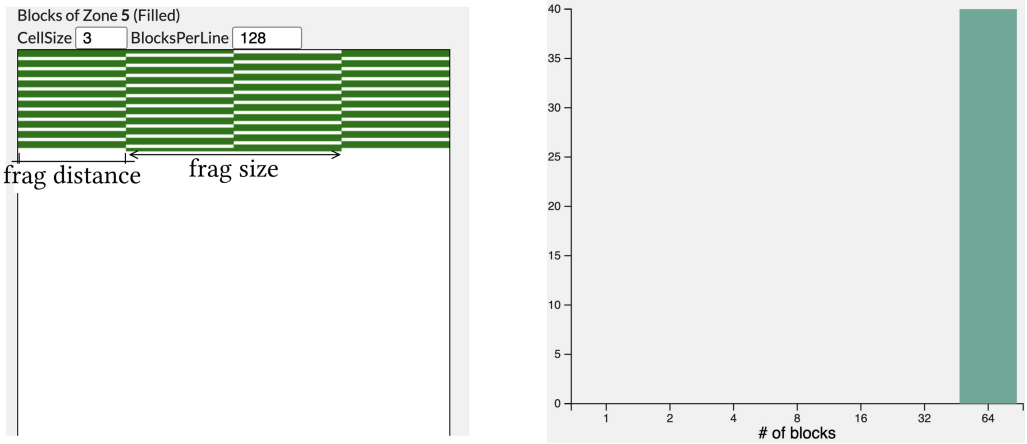
```
bpftool btf dump file /sys/kernel/btf/f2fs format c > f2fs.h
```

With the generated header and BTF information from the compiled kernel, our BPF program could trace specific F2FS functions and structures with better portability across various platforms.

3.4 Visualization

Figure 5 shows the server client architectures of the visualization module of `ZoneTrace`. The modified `f2fs-tools` provides the layout of F2FS on ZNS SSDs with device information. Our eBPF program tracing and `fibmap.f2fs` from f2fs-tools transfers the updated valid bitmap and the file extent layout in binary format to the standard output. A server intercepts the standard output of the eBPF program, as well as the `FIBMAP` ioctl system call through the pipe interface. Then, the server decodes the standard output with rendering information to the client web browser, and visualizes the space management of F2FS on ZNS SSDs accordingly.

Due to limited memory size of the client web browser, we are not able to visualize each zone's bitmap usage simultaneously. Instead, at the client side, we let the user choose the zone to monitor and visualize the selected zone's segment block bitmap. We display all other zones not chosen by the user at the zone level, indicating which zones are active with colors representing segment type, which enables users to focus on zone monitoring. A server was created to collect update events from the eBPF program in real time and sends the corresponding update that bitmap of the



(a) Visualization of F2FS file mapping to zone

(b) Frag. histogram of the selected file

Fig. 6. Visualization of a 10 MB file with frag size (green) 256 KB and 128 KB frag distance (white gap).

user-selected zone to the client. As shown in Algorithm 1, we used batch processing of multiple events for relieving the runtime overhead and set the time interval to 200 ms. On each update event from eBPF, we check if the number of the updated zone matches the user-selected zone. If it does not match, then we send the updated zone number to the client. If it matches, then we notate the updated segment number and store the updated valid bitmap. In each batch, we dispatch the bitmap for notated segments to the client and clear these notations. This mechanism allows us to provide efficient updates from server to client with light-weight overhead.

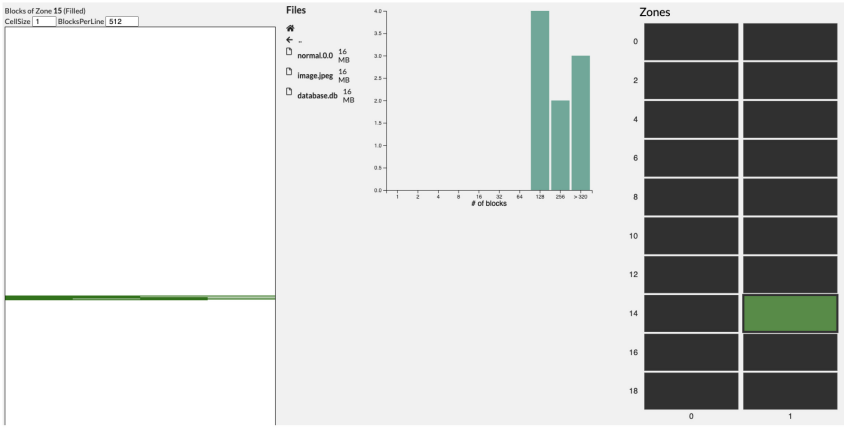
3.5 File Layout Manipulation with F2FS Zoned Mode

Previous work [28] uses direct I/O on F2FS with block-based SSD to synthetically generate file fragmentation. However, in F2FS with ZNS SSDs, F2FS is forced to use buffered I/O [36] for writes. Researchers can not directly apply previous schemes for file system manipulation. To adress this need, we provide new ways to manipulate the file layout with F2FS on ZNS SSDs within the context of the F2FS buffered I/O scheme. First, motivated by previous work [28], we define two metrics *frag size* and *frag distance*, which can be visualized with *ZoneTrace* as shown in Figure 6(a). *frag size* refers to the size of each fragment, while *frag distance* denotes the distance between two consecutive fragments. Then, to manipulate the created file from a single zone with F2FS on ZNS SSDs, we create a file with predefined multimedia file extensions such as png, which will be stored in the F2FS cold log. Next, we use sync which flushes file system buffers when certain *frag size* is copied. For the purpose of generating the *frag distance*, we create another file that has the size of *frag distance*, then delete it later to generate the empty distance of the between files. Since ZNS SSDs do not have magnetic heads like traditional hard disk drives, the *frag distance* between each fragment does not affect the read performance. As a result, we further propose a novel fragment histogram that further helps visualize the distribution of fragment size of each file as the fragmentation metric, as shown in Figure 6(b) for user selected files.

4 EVALUATION

We demonstrate and evaluate the utility of `ZoneTrace` on a 1TB Western Digital Ultrastar DC ZN540 drive [11]. This device has a 2 GiB zone size and a 1,077 MiB zone capacity with 905 zones in total. Due to limited space of the article, we make use of 20 zones of ZNS SSDs in total. For primary

(a) F2FS segments usage view



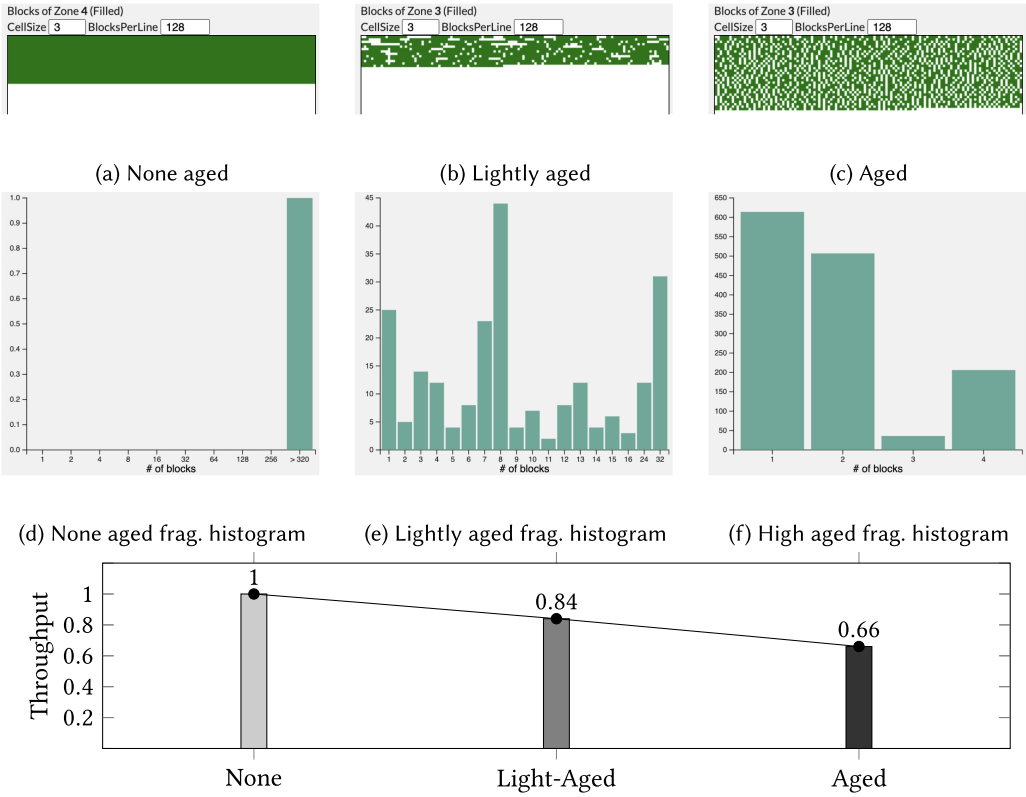
(b) F2FS file mapping with fragmentation histogram

Fig. 7. Monitoring interface of ZoneTrace.

use of the ZNS SSD with 20 zones, we partitioned a 500 GB WD Blue SN570 NVMe SSD [12] into 4 GB partitions for random writable sections in F2FS. Our experimental setup involved a machine equipped with an Intel Core i7-10700K (8 cores @ 3.80 GHz) and 32 GB of DDR4 memory (2,666 MHz) running Ubuntu 22.04 (kernel 6.2.10, f2fs-tools 1.16.0 [10]) For overhead analysis, we use fio [1] with random write-heavy workloads, the worst cases for log-structure file systems. We further evaluate the overhead with filebench [24], and **Yahoo! Cloud Serving Benchmark (YCSB)** workloads [10] with RocksDB [14], which are used for evaluation in existing works. For users without ZNS SSDs, we also includes a quick start guide of using `ZoneTrace` with FEMU [22], a QEMU-based flash emulator.

4.1 User Interface

Figure 7 shows a typical snapshot of the `ZoneTrace` user interface for visualizing the space management of F2FS on ZNS SSDs. `ZoneTrace` provides users with two separate user interfaces for visualization: the F2FS segment usage, and the F2FS file mapping with fragment histogram. As shown in Figure 7(a), for the F2FS segment usage interface, users can select a zone to monitor, and



(g) Normalized sequential read performance of the synthetic file

Fig. 8. Fragment distribution of the synthetic 10 MB file.

ZoneTrace shows the corresponding F2FS segments' valid bitmap within the zone on the left side. Furthermore, in F2FS there are six types of segments for hot and cold data separation by default. As a result, we further label the type of updated segments in the zone with different colors. For example, blue represents cold data segments and red represents hot data segments. Figure 7(b) shows the file mapping visualization interface of ZoneTrace. Users can select a file to inspect, ZoneTrace will highlight the zones containing the selected files. Then, the user can select the highlighted zones of the file, and the left-hand side will show the file mapping layout on the selected zone for further analysis.

4.2 File Layout Manipulation

To demonstrate the use cases of ZoneTrace, we created a 10 MB file using fio [1] with ranges of issues request block sizes and sync option enable for flushing the file system page cache for generating various level of fragmentation. Users can easily observe and identify each file's fragmentation in each zones of ZNS SSDs easily with ZoneTrace. Moreover, to model the fragmentation level of each file, Figures 8(d), 8(e), and 8(f) shows the fragmentation histogram of the 10 MB file from ZoneTrace. As shown in Figure 8(f), files with higher-levels of fragmentation will contain more fragments having smaller number of continuous blocks. Figures 8(e) and 8(f) show the fragmentation histogram from ZoneTrace, which contains more contiguous data blocks distribution. To evaluate the effect of different levels of fragmentation F2FS has on ZNS SSDs, we measured the

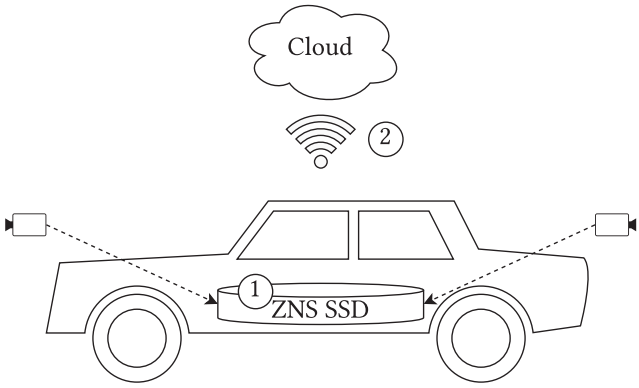
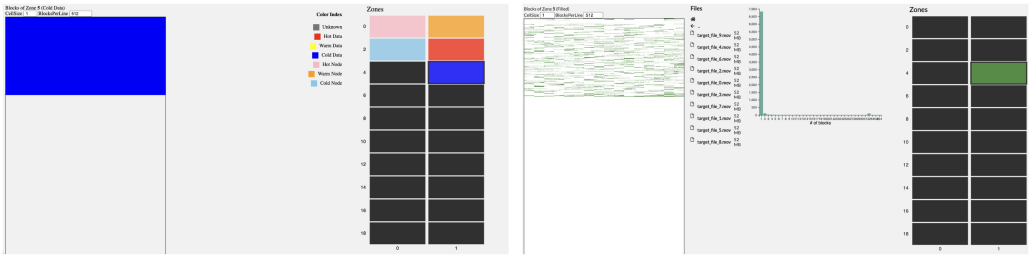


Fig. 9. Autonomous systems data collection process.



(a) F2FS segments usage view (b) F2FS file mapping view

Fig. 10. Visualization results of ZoneTrace with emulated 10 video streams saving.

sequential read on the synthetic 10 MB file. As shown in Figure 8(g), in highly fragmented situations, there can be up to a 34% degradation in sequential read throughput. `ZoneTrace` is able to visualize and assist users with fragmentation histogram to identify issues with the space management of F2FS with ZNS SSDs.

4.3 `ZoneTrace` Use Case: Evaluating Streams Saving in Autonomous Systems

As shown in Figure 9, **autonomous vehicles (AVs)** require in-vehicle data recording [15], e.g., for future investigation when accidents occur. Furthermore, with the advancement of cloud technology, to improve the **Advanced Driver-assistance Systems (ADAS)** in AVs, the sensor data generated from AVs must be stored locally (① in Figure 9) and then sent to the cloud (② in Figure 9) later for future reuse in training and refining the ADAS [2, 26, 32].

To emulate the streaming saving scenarios of AVs, we make use of the `dd` [31] command to sequentially write 10 video streams simultaneously to ZNS SSD formatted with F2FS with `.mov` file extension. Through `ZoneTrace`, as shown in Figure 10, we observed that there is a series of fragmentation with multiple video streams when using F2FS with ZNS SSDs, which motivates us to investigate the reasons behind why F2FS stores each stream on ZNS SSDs with a higher level of fragmentation.

As shown in Table 2, F2FS maintains a predefined file extensions list to determine the data type of files. For example, for multimedia files such as videos and images, F2FS treats them as cold data by default, since the files are prone to show write-once and read-only patterns. However, as illustrated in Figure 11, saving multiple cold streams into the same cold zone with ZNS SSD incurs

Table 2. F2FS Predefined File Extension Lists for Hot and Cold Data [19]

File Type	File extensions	Segment Type
video	.avi, .m4v, .m4p, .mkv, .mov, .webm	Cold
audio	.wav, .m4a, .3gp, .opus, .flac	Cold
image	.gif, .png, .svg, .web	Cold
database	.db	Hot
virtual machines	.vmdk, .vdi, .qcow2	Hot

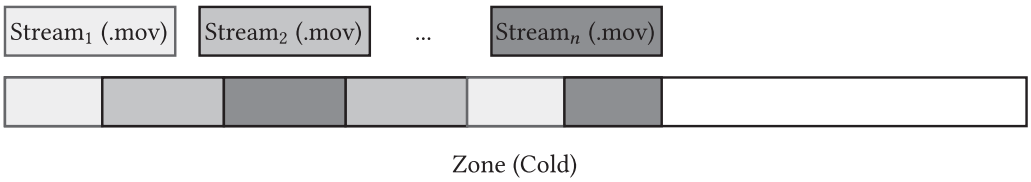


Fig. 11. F2FS store streams into same cold zones on ZNS SSDs based on file extensions.

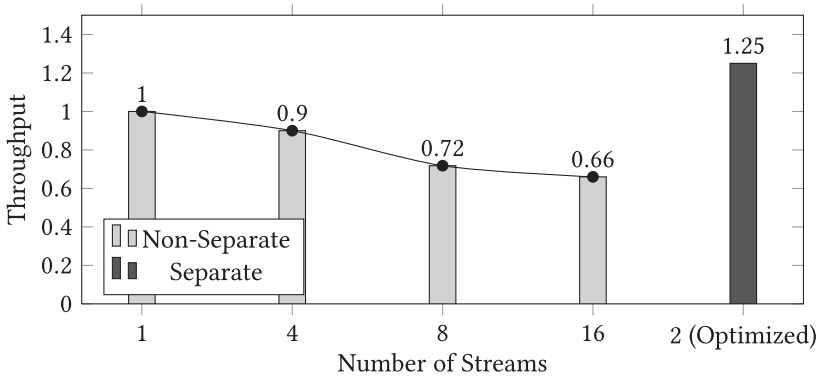


Fig. 12. Normalized average sequential stream read throughput with F2FS on ZNS SSD.

fragmentation. Such fragmentation could lead to read performance degradation when sending the streams back to the cloud. The visualization capability of `ZoneTrace` has helped us observe fragmentation issues intuitively on ZNS SSDs at runtime.

To demonstrate the read performance degradation due to fragmentation, we emulate the streaming saving with varying number of streams. Each stream is saved with cold data file extensions predefined with F2FS. After the emulation, we later measure the average sequential read throughput of each saved streams. As shown in Figure 12, the average read throughput decreases dramatically as the number of streams due to the increased level of file fragmentation. We further do initial exploration by storing two streams with two different file extensions: `.mov` and .db with F2FS on ZNS SSDs. By default, the files type such as the database files that tend to be updated frequently are stored as hot data in F2FS. As a result, the video streams with file extensions `.mov` will be stored within cold zone, the database streams will be stored within the hot zone, which separate the two emulated data streams. Therefore, as shown in Figure 12, separation of saving streams is able to reduce stream fragmentation with F2FS on ZNS SSD, improving the read throughput of the saving streams. For future optimization, a fine grained data placement strategy should be proposed to alleviate the fragmentation issues with F2FS zone mode for storing data streams.

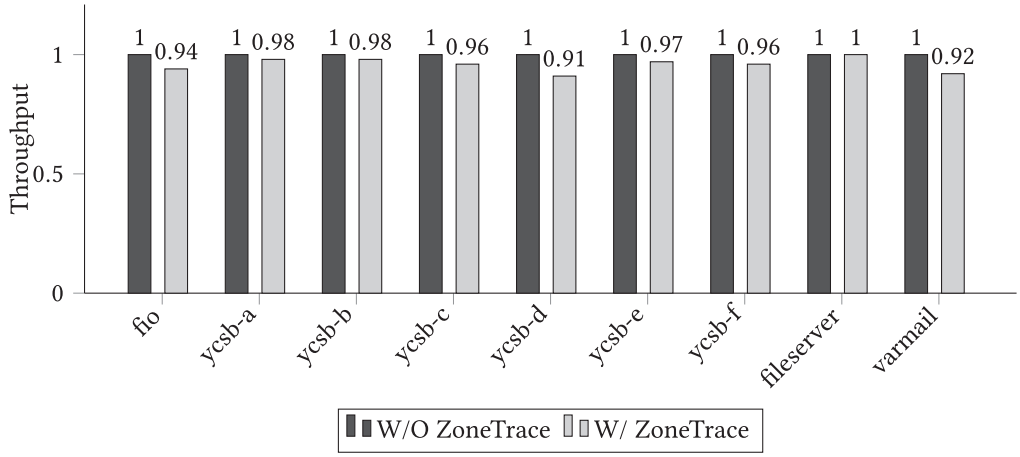


Fig. 13. Tracing overhead of ZoneTrace.

4.4 ZoneTrace Tracing Overhead

To evaluate the tracing overhead of ZoneTrace, we first use fio with random-write heavy workloads with 16 number of jobs as micro-benchmarks. For macro-benchmarks evaluation, we use YCSB workloads a to f and fileserver and varmail workloads from filebench. Figure 13 shows the normalized overall throughput reported from each benchmarks with the comparison between running with and without ZoneTrace. As shown in Figure 13, although ZoneTrace causes extra CPU and memory usage for runtime tracing, ZoneTrace lead to only an average 4.1% throughput slowdown to the foreground workloads. Since the main goal of ZoneTrace is to assist researchers to analyze space management with F2FS on ZNS SSD with ease as illustrated in Section 4.3, we believe the overhead should be acceptable.

5 CONCLUSION

We introduced ZoneTrace, a runtime visualization and monitoring tool that effectively presents the space management between F2FS and ZNS SSDs. ZoneTrace facilitates the analysis of F2FS at the zone level, enabling further optimization of space management policies with F2FS on ZNS SSDs in various research areas, such as file system aging and host-managed garbage collection. ZoneTrace leverages eBPF to continuously monitor changes in the segment bitmap of F2FS during runtime and employs the syscall to retrieve F2FS file extent mappings from userspace. Additionally, ZoneTrace offers a new scheme to aid users in analyzing file fragmentation in F2FS when using ZNS SSDs. The enhanced functionality of our tool enable users to explore different design considerations, including various fragment sizes and distances. We demonstrated the usefulness of ZoneTrace on the practical use-case of optimizing space for streaming data recording in autonomous systems. With the help of ZoneTrace, we gained insight how F2FS leads to fragmentation in the current scheme that uses file extensions for hot/cold data identification. ZoneTrace facilitated an initial exploration of optimized data placement for F2FS on ZNS SSDs with streams saving. Our future work will investigate how ZoneTrace can assist with evaluating novel data placement scheme using F2FS on ZNS SSDs. The source code of ZoneTrace can be accessed with the following link: https://github.com/pingxiang-chen/bpf-f2fs-zonetrace.git

ACKNOWLEDGMENTS

We thank four anonymous reviewers for their help in improving the presentation of this article.

REFERENCES

[1] Jens Axboe. 2005. Fio-flexible i/o tester synthetic benchmark. Retrieved from https://github.com/axboe/fio

[2] Liam Benson, Kashish Jain, and Taehoon Kang. 2022. US11823505B1—Data manager—Google Patents. Retrieved from https://patents.google.com/patent/US11823505B1/en?oq=11823505

[3] Aniket Bhattacharyea. 2021. A practical guide to BTF (BPF Type Format). Retrieved from https://www.airplane.dev/blog/btf-bpf-type-format

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

[5] Ping-Xiang Chen, Shuo-Han Chen, Yuan-Hao Chang, Yu-Pei Liang, and Wei-Kuan Shih. 2021. Facilitating the efficiency of secure file data and metadata deletion on SMR-Based Ext4 file system. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC'21)*. ACM, New York, NY, 728–733. https://doi.org/10.1145/3394885.3431517

[6] Minseon Cho and Donghyun Kang. 2023. FragTracer: Real-time fragmentation monitoring tool for F2FS file system. *Sensors* 23, 9 (2023), 4488.

[7] Gunhee Choi, Kwanghee Lee, Myunghoon Oh, Jongmoo Choi, Jhuyeong Jhin, and Yongseok Oh. 2020. A new LSM-style garbage collection scheme for ZNS SSDs. In *Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'20)*.

[8] Shih-Chun Chou, Yi-Shen Chen, Ping-Xiang Chen, Yuan-Hao Chang, Ming-Chang Yang, Tei-Wei Kuo, Yu-Fang Chen, and Yu-Ming Chang. 2023. FSD: File-related secure deletion to prolong the lifetime of solid-state drives. In *Proceedings of the IEEE 12th Non-Volatile Memory Systems and Applications Symposium (NVMSA'23)*. 19–24. https://doi.org/10.1109/NVMSA58981.2023.00015

[9] Alex Conway, Eric Knorr, Yizheng Jiao, Michael A. Bender, William Jannen, Rob Johnson, Donald Porter, and Martin Farach-Colton. 2019. Filesystem aging: It's more usage than fullness. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'19)*.

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

[11] Western Digital Corporation. 2021. 2.5-inch U.2, 15mm, NVMe ZNS Solid State Drive (SSD). Retrieved from https://www.hdstorageworks.com/Ultrastar-DC-ZN540.asp

[12] Western Digital Corporation. 2022. WD Blue SN570 NVMe SSD. Retrieved from https://www.westerndigital.com/products/internal-drives/wd-blue-sn570-nvme-ssd#WDS500G3B0C

[13] Renzo Davoli and Michele Di Stefano. 2019. *Berkeley Packet Filter: Theory, practice, and perspectives*. Ph.D. Dissertation. Master's thesis, Universita di Bologna.

[14] Facebook. 2019. RocksDB. http://rocksdb.org

[15] Stephan Heinrich and Lucid Motors. 2017. Flash memory in the emerging age of autonomy. In *Proceedings of the Flash Memory Summit*. 1–10.

[16] Yuhun Jun, Shinhyun Park, Jeong-Uk Kang, Sang-Hoon Kim, and Euiseong Seo. 2024. We ain't afraid of no file fragmentation: Causes and prevention of its performance impact on modern flash SSDs. In *Proceedings of the 22nd USENIX Conference on File and Storage Technologies (FAST'24)*. 193–208.

[17] 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 *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'18)*. 691–704.

[18] Jaegeuk Kim. 2022. f2fs-tools. Retrieved from https://kernel.googlesource.com/pub/scm/linux/kernel/git/jaegeuk/f2fs-tools/+/refs/tags/v1.16.0

[19] Jaegeuk Kim. 2023. F2FS default hold and cold data file extentions list. Retrieved from https://kernel.googlesource.com/pub/scm/linux/kernel/git/jaegeuk/f2fs-tools/+/668e8154404b9577d95c641e8ff45ad19a6fb5b8%5E%21/mkfs/f2fs_format.c

[20] Colin Ian King. 2009. FIBMAP ioctl example—Get the file system block number of a file. Retrieved from http://smackerelofopinion.blogspot.com/2009/06/fibmap-ioctl-file-system-block-number.html

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

[22] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminatahan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. 2018. The case of FEMU: Cheap, accurate, scalable and extensible flash emulator. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*. USENIX Association, 83–90.

[23] Steven McCanne and Van Jacobson. 1993. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter Conference*, Vol. 46. 259–270.

[24] Richard McDougall and Jim Mauro. 2005. FileBench. Retrieved from http://www.nfsv4bat.org/Documents/nasconf/2004/filebench.pdf
[25] Andrii Nakryiko. 2020. BPF Portability and CO-RE. Retrieved from https://facebookmicrosites.github.io/bpf/blog/2020/02/19/bpf-portability-and-co-re.html
[26] Cristian Olariu, Juan Diego Ortega, and J Javier Yebes. 2018. The role of cloud-computing in the development and application of ADAS. In *Proceedings of the 26th European signal processing conference (EUSIPCO'18)*. IEEE, 1037–1041.
[27] Western Digital Corporation or its affiliates. 2023. Btrfs Zoned Mode. Retrieved from https://zonedstorage.io/docs/linux/fs#btrfs Accessed 30-01-2024.
[28] 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*. 280–294.
[29] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. BTRFS: The linux B-tree filesystem. *ACM Trans. Storage* 9, 3, Article 9 (Aug. 2013), 32 pages. https://doi.org/10.1145/2501620.2501623
[30] Mendel Rosenblum and John K. Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.* 10, 1 (1992), 26–52.
[31] Paul Rubin, David MacKenzie, and Stuart Kemp. 2023. dd(1)—Linux manual page—man7.org. Retrieved from https://man7.org/linux/man-pages/man1/dd.1.html. Accessed 27-01-2024.
[32] Tomas Saaristola et al. 2022. Data collection system for autonomous vehicles. https://aaltodoc.aalto.fi/items/676ec2f4-cf6f-4820-8300-df23fffe558a
[33] Dongjoo Seo, Ping-Xiang Chen, Huaicheng Li, Matias Bjørling, and Nikil Dutt. 2023. Is garbage collection overhead gone? Case study of F2FS on ZNS SSDs. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage'23)*. ACM, New York, NY, 102–108. https://doi.org/10.1145/3599691.3603409
[34] Dongjoo Seo, Biswadip Maity, Ping-Xiang Chen, Dukyoung Yun, Bryan Donyanavard, and Nikil Dutt. 2022. ProSwap: Period-aware proactive swapping to maximize embedded application performance. In *Proceedings of the IEEE International Conference on Networking, Architecture and Storage (NAS'22)*. 1–4. https://doi.org/10.1109/NAS55553.2022.9925330
[35] Nick Tehrany, Krijn Doekemeijer, and Animesh Trivedi. 2023. Understanding (Un)written contracts of NVMe ZNS devices with zns-tools. Retrieved from https://arXiv:2307.11860
[36] Linux Torvalds. 2023. F2FS forces direct I/O to convert to buffered I/O for zoned device. Retrieved from https://github.com/torvalds/linux/blob/d3f704310cc7e04e89d178ea080a2e74dae9db67/fs/f2fs/file.c#L808
[37] 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 Design, Automation, and Test in Europe Conference and Exhibition (DATE'20)*. IEEE, 1061–1066.

Received 9 November 2023; revised 13 March 2024; accepted 22 March 2024