Is Garbage Collection Overhead Gone? Case study of F2FS on ZNS SSDs

Dongjoo Seo*
Ping-Xiang Chen*
University of California, Irvine
{dseo3,p.x.chen}@uci.edu

Huaicheng Li
Virginia Tech
huaicheng@cs.vt.edu

Matias Bjørling
Western Digital
Matias.Bjorling@wdc.com

Nikil Dutt
University of California, Irvine
dutt@ics.uci.edu

ABSTRACT

The sequential write nature of ZNS SSDs makes them very well-suited for log-structured file systems. The Flash-Friendly File System (F2FS), is one such log-structured file system and has recently gained support for use with ZNS SSDs. The large F2FS over-provisioning space for ZNS SSDs greatly reduces the garbage collection (GC) overhead in the log-structured file systems. Motivated by this observation, we explore the trade-off between disk utilization and over-provisioning space, which affects the garbage collection process, as well as the user application performance. To address the performance degradation in write-intensive workloads caused by GC overhead, we propose a modified free segment-finding policy and a Parallel Garbage Collection (P-GC) scheme for F2FS that efficiently reduces GC overhead. Our evaluation results demonstrate that our P-GC scheme can achieve up to 42% performance enhancement with various workloads.

CCS CONCEPTS

• **Software and its engineering → File systems management**.

KEYWORDS

F2FS, Garbage Collection, Zoned Namespace SSDs

*Both authors contributed equally to the paper.

HotStorage '23, July 9, 2023, Boston, MA, USA

ACM ISBN 979-8-4007-0224-2/23/07.
https://doi.org/10.1145/3599691.3603409

ACM Reference Format:
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 *15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23), July 9, 2023, Boston, MA, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3599691.3603409

1 INTRODUCTION

The introduction of SSDs with Zoned Namespace (ZNS) support is a new type of storage device to avoid the performance overhead associated with the traditional block interface for flash-based SSDs [4, 9, 21]. Instead of providing an interface with a single array of logical blocks, a ZNS SSD groups logical blocks into zones. This allows the SSD to align the zones to its media characteristics. For flash-based SSDs specifically, logical blocks in a zone can be read randomly, but they must be written sequentially, and the entire zone must be reset before it can be rewritten. ZNS SSDs delegate the responsibility of fine-grained data management to the host system, which means that the SSD's Flash Translation Layer (FTL) collaborates with the host software to place data onto the logical blocks within a zone, while the SSDs continue to manage the coarse-grained activities, such as wear-leveling and media reliability. The collaboration significantly reduces the device-side garbage collection (GC), which leads the higher performance, improved latencies as well as longer device lifetime. Having the fine-grained data placement responsibility in the host presents an opportunity for the design of the host-side GC scheme to leverage the unique characteristics of ZNS SSDs. The Flash Friendly File System (F2FS) [14] was chosen to utilize the performance benefits of a ZNS SSD through a file interface at the user level. By forcing pure Log-File Structure (LFS) mode [20] to eliminate random writes to be written sequentially, F2FS has been proven to be efficient, in compliance with the sequential-write constraint of ZNS SSDs [4].

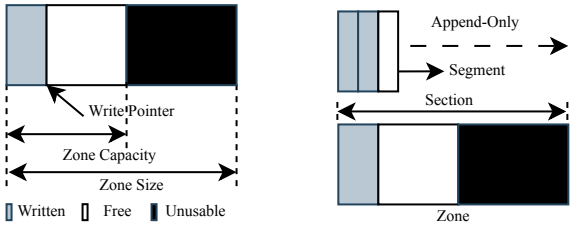
However, based on our observation, GC overhead [11, 13, 16, 23] does not appear when we use F2FS with ZNS SSDs even with high-utilization. This motivates us to revisit the layout of F2FS on ZNS SSDs compared with block-based SSDs. We found that F2FS relieves GC overhead by over-provisioning storage spaces for ZNS SSD, similar to how conventional SSDs over-provision its media which typically accounts for 7-28% of the total media. We explore the trade-off between a number of over-provisioning spaces and user application performance affected by GC, and observe that the size of over-provisioning space is an important factor for performance. Also, we observed that configurations with low over-provisioning space resulted in crashes for applications with high utilization. To enable exploration of these low over-provisioning space configurations, we added additional free segment searches in LFS mode. Since the low over-provisioning space configurations incur GC overhead, we propose a Parallel Garbage Collection (P-GC) scheme that accelerates the GC process by parallelizing GC processing of victim segments. The main contributions of this study can be highlighted as follows:

- This paper demonstrates the on-disk layout of the LFS mode of F2FS on ZNS SSDs design compared to F2FS for conventional SSDs.
- We explore the trade-off between over-provisioning space and file system performance with a new free segment search scheme and propose a parallel GC system design that accelerates the GC process.
- We evaluate our schemes based on F2FS with various application sets that include write-intensive workloads. Our P-GC scheme achieves an average 24.5% performance improvement in both micro- and macro-benchmarks.

2 BACKGROUND AND MOTIVATION

2.1 Zoned Namespace (ZNS) SSD

The ZNS interface is a new feature that is introduced to overcome the performance overhead associated with the existing block interface of flash-based SSDs. Unlike traditional SSDs, ZNS SSDs group logical blocks into zones that can only be written sequentially. As shown in Figure 1a, the zone size is defined as the total number of logical blocks within the zone. Each zone has a write pointer that designates the next writeable LBA within a writeable zone. Zoned storage SSDs also define a zone capacity attribute for each zone, which indicates the number of usable logical blocks within the zone, starting from the first logical block of the zone. It is either equal or smaller than the zone size. The use of zone capacity can be aligned to the size of flash erase blocks for a flash-based device, while maintaining a specific zone size characteristic, such as maintaining a power of two zone size.



(a) Zone Size / Capacity **(b) F2FS on ZNS SSDs**

Figure 1: ZNS SSDs and F2FS main area on-disk layout

2.2 F2FS with LFS mode

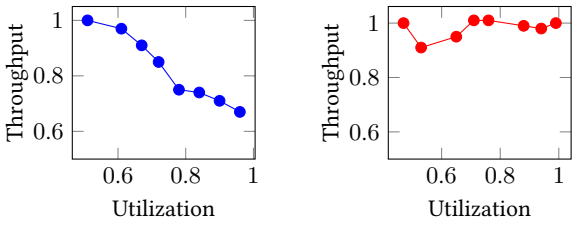
F2FS is a popular file system for flash-based disks that divides the data region into random writes and multi-stream sequential write regions. Using ZNS SSDs with F2FS requires a multi-device setup to place metadata blocks area (Superblock, Checkpoint, Segment Information Table, Node Address Table, and Segment Summary Area) on randomly writable storage and main blocks area (Node, Data) on ZNS SSD. Figure 1b shows the data structures layout of the main area that F2FS managed for the file system at zone on ZNS SSDs. F2FS divides the whole volume into fixed-size 2MB segments. The segment is a basic unit of management in F2FS and is used to determine the initial file system metadata layout. When a ZNS SSD is formatted for F2FS, a section is a group of fixed-size segments (2 MB), and the number of segments in a section is determined to match the zoned device zone size in ZNS SSD. The unwriteable logical blocks that exist between a zone's zone capacity and its zone size are explicitly marked reserved and not utilized for writes.

Furthermore, F2FS utilizes a pure Log-Structured File System (LFS) mode; the LFS forces sequential writes to segments and forces the sequential use of segments within sections, which results in full compliance with the zoned block device's write constraint. However, in the LFS mode of F2FS, we could not directly reuse the invalid block until it becomes free by GC. This limitation causes GC to trigger more frequently than in the normal mode of F2FS, resulting in high GC overhead when disk utilization gets higher, i.e, when we explore low over-provisioning configurations.

F2FS employs foreground and background GC schemes with different priorities and challenges to reclaim space from invalidated data pages and maintain performance. Foreground GC is triggered in real-time and can potentially impact the user experience by blocking other requests, including concurrent free page requests. Background GC, on the other hand, is a periodic process that aims to reduce data fragmentation, increase space utilization, and minimize performance overhead. To enhance user experience, it is critical to decrease the overhead of foreground GC. In this study, we revisit these issues in the context of the ZNS SSDs.

2.3 What we observed

Figure 2 depicts the write-intensive YCSB-A workload throughput between F2FS on normal SSD and F2FS on ZNS SSD under different disk utilization. The normal SSD size is 45GB, and the ZNS SSDs is with 45 zones with 1077MiB (≈1GB) zone capacity and 2GB zone size. Surprisingly, we did not observe any performance decrease due to GC for high disk utilization on the tested ZNS SSD, which motivates us to answer why GC overhead does not happen. Figures 3a and 3b show the over-provisioning space by F2FS on both same usable size SSDs. F2FS on ZNS SSD over-provisioning 36.9x times the number of segments. Therefore, in this work, we want to answer several research questions: (1) How does F2FS decide the over-provisioning space for ZNS SSDs? (2) Can we over-provision less and observe its impact on performance? (3) How to reduce GC overhead if indeed it manifests in F2FS?



(a) Normal SSD **(b) ZNS SSD**

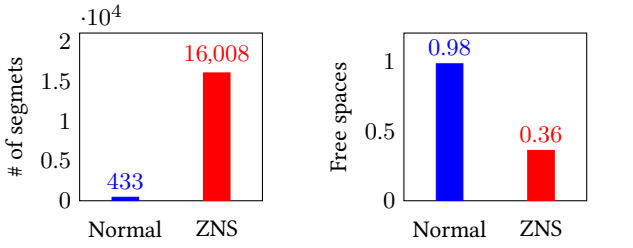
Figure 2: Performance v.s F2FS Utilization

3 EXPLORATION AND OPTIMIZATION

3.1 Over-provisioning in F2FS

To focus on file system level over-provisioning ratio, we intentionally ensure that block-based SSD has ample free space [14]. ZNS SSD has the advantage of not having internal over-provisioning space for GC. However, since the host system is responsible for GC, LFS still needs to reserve space for cleaning invalid blocks within each segment. F2FS reserves an over-provision area when the file system is initialized to address this. Figure 3 shows the number of over-provisioning segments when we formatted the same disk space size between conventional block-based SSDs and ZNS SSDs, both with 45 GB of usable space in total. As shown in Figure 3a, ZNS SSDs reserved 36.9× times additional number of segments than conventional SSDs, which led to only 36% of usable space in ZNS SSDs in Figure 3b. The reason for such difference is that the over-provisioning segments are based on average usable segments per section when formatting the device based on Formula 1 with $x\%$ and a $margin$[1] value for over-provisioning.

$$ovp\_segs = (100/x + margin) * avg\_seg\_per\_sec \quad (1)$$



(a) Reserved segments **(b) Free space per disk space**

Figure 3: Impact of reserved segments on F2FS

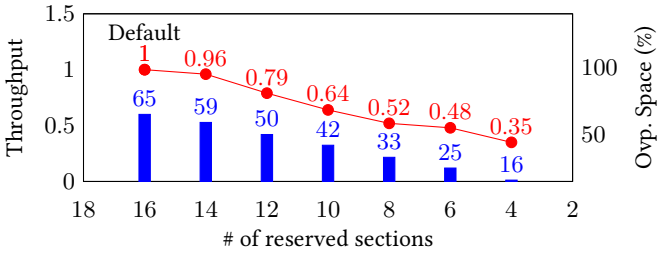


Figure 4: Exploration for reserved sections with F2FS on 45 zones ZNS SSD with 95% file system utilization

With 2MB segments, for conventional SSDs, the default segment per section is 1. However, in our ZNS SSDs setting with 1077MiB zone capacity, the usable segments per section are 539 by zone capacity. Thus, suppose we would like to reserve 10% in the file system. Then, the total of reserved segments for normal SSDs will be 16, but for ZNS SSDs will be 8624, which could be 539× times larger than the default F2FS setting. We conduct the following experiment to explore the trade-off between over-provisioning space and write-intensive multi-threaded applications. We first modify f2fs-tools [10] for customizing over-provisioning space in F2FS for 45 zones ZNS SSDs. Then, we use fio [3], which fills 95% of the available space in the file system with 4KB random buffered writes with multiple small file access per job for 10 minutes to compare the write throughput under the different size of over-provisioning space. Intuitively, the more over-provisioning space, the more latency reduction we can obtain, which can be shown in Figure 4.

Finding a balance between disk utilization and GC overhead is essential. Having too little free space can result in significant GC overhead, even though sacrificing storage space can improve performance. As a result, considering over-provisioning space is also an important host-management factor for users wanting to use F2FS with ZNS SSDs. The

[1]This margin value equals to the number of active logging in F2FS, which in default is 6.

current approach for making over-provisioning decisions in F2FS is tilted toward avoiding GC overhead for ZNS SSDs with large zone sizes. Based on our exploration in Figure 4, we choose to reserve 10 sections for F2FS with ZNS SSDs since this configuration can strike a balance between over-provisioning and the performance of the application.

3.2 Free segment search in LFS mode

Figure 5 shows that the system crashes for low over-provisioning space configurations for I/O intensive applications. We investigated the cause of this problem and propose a new free segment search policy that enables exploration of these configurations.

The LFS mode performs a forward search (right direction) for a free segment starting at the current section. When new write requests are received, F2FS checks whether segments can handle the writes and if not, it attempts to find a new free section in the right direction using a bitmap. When the forward search cannot find an available section, an alert is sent to the user side, and the application crashes. Note that we do not observe this situation with conventional SSDs since one-to-one segment-per-section mapping results in a high likelihood of finding a free section in the right direction, even under high utilization.

However, ZNS SSDs only have a number of sections that match the number of zones, making it difficult to find a free section under high utilization with multiple writers on the file system. To address this issue, we added a retry behavior that checks for free sections from the start of the main data part to the end of the device via a bitmap in the new segment-finding routine. This approach significantly reduces the risk of crashes in scenarios where multiple write applications are running on a high-utilization system. Figure 5 shows that the performance clearly reduced due to low over-provisioning space for GC. **Vanilla** can maintain the performance well but can not utilize the disk space we partitioned. **ZNS-OP**, configured with a low over-provisioning space, shows that performance drops by an increase in utilization. However, an application has the possibility that it can crash when disk utilization goes high. Finally, **ZNS-OP-SS**, which adds a new segment search routine on **ZNS-OP**, uncovers the entire design space, including the high utilization region.

3.3 Garbage collection optimization

F2FS has two garbage collection (GC) modes, namely foreground (FG) and background (BG). The FG mode is triggered when there are insufficient free sections available. On the other hand, the BG mode runs periodically in the background when the file system is idle, either with or without I/O operations. We focus on the FG path because BG mode will skip if FG is triggered. Since ZNS SSDs have a limitation of

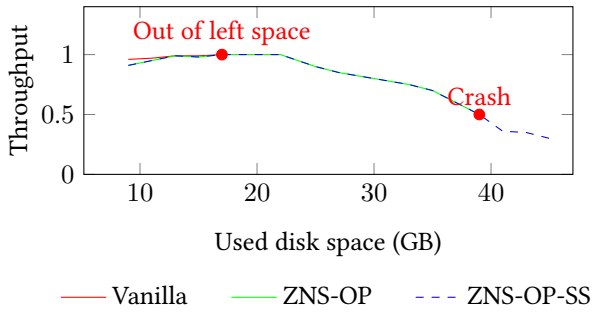


Figure 5: Workload performance graphs by utilization

Algorithm 1 Parallel GC

```
stop fs operations
find victim section
Iterate each segment
if validblks ≥ half then
    GC with parallelize
else
    GC with single
end if
resume fs operations
```

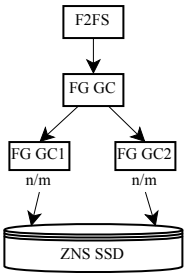


Figure 6: Parallel GC Scheme

sequential write, LFS cannot reuse invalid blocks and must clean an entire section to reclaim a dirty one. This limitation accelerates the use of free segments and trigger GC.

When FG occurs, victim selection policy finds the section that includes multiple segments composed of multiple blocks. The selected section includes multiple segments, and we currently need to check each segment sequentially to determine if we need to move the data inside of the segment to others for the free segment. When revisiting the F2FS GC process, we found that most of the total time consumed by GC stems from valid data moving within large zones which involves locating valid blocks and moving them to other sections.

From this observation, we propose Algorithm 1, namely P-GC, a method to improve GC performance, which consists of three phases: (1) Finding the victim sections. (2) Iterating each segment in the sections to acquire information about the target victim blocks. (3) Performing P-GC per segment if the number of valid blocks is larger than half of the total blocks in the segment. Another restriction from F2FS with zoned storage is that F2FS forces to convert direct write I/O to buffered I/O [22]. In cases where there is a burst of buffered read/write usage, the buffered I/O inheritance problem exacerbates. Such a burst results in increased memory usage, which may trigger the page cache drop and cause data written to the disk to be lost. As a result, the data-moving process during GC, which entails large sequential write and

multiple random read operations, is affected. To address this issue, as shown in Algorithm 1, P-GC breaks down the GC per segment process into multiple threads if the number of valid blocks exceeds half of the blocks per segment. Our approach involves parallelizing m threads to split n victim blocks in each segment as well as enhancing GC's random read performance, as depicted in Figure 6.

4 EVALUATION

4.1 Experimental setup

All the experiments were conducted using 1TB Western Digital Ultrastar DC ZN540 [18]. The device comes with 2GiB zone size and 1077MiB zone capacity. In order to achieve high utilization in F2FS more quickly, we first use *nvme-cli* [1] to format the ZNS device and conduct an evaluation with 45 zones in total with our fio [3] micro-benchmark. Similarly, we further format the device using 301 zones and fill the file system to 95% utilization for our filebench [17] and YCSB workloads [7] on RocksDB [8] macro benchmarks evaluation. For using ZNS SSD mainly, we partitioned a 500GB WD Blue SN570 NVMe SSD [19] with 4GB partition in total for random writable sections in F2FS and mounted both of the devices with F2FS using LFS mode. Based on our exploration from the microbenchmark, we empirically over-provisioning 10 sections for 301 zones of the tested ZNS SSD. We set up a machine equipped with Intel(R) Core(TM) i7-10700K (8 cores @ 3.80GHz) and 32 GB of memory (DDR4, 2666 MHz), running Ubuntu 20.04 (kernel 6.0.7, f2fs-tools 1.15.0 [10]).

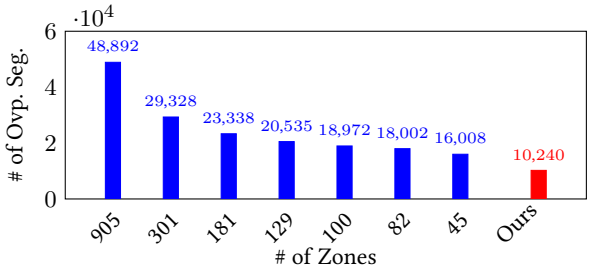


Figure 7: Exploration for reserved segments with F2FS with different ZNS SSDs configuration

Figure 7 shows the difference between vanilla F2FS and our modified over-provisioning space for ZNS SSDs. Based on our exploration in Section 3.1, we over-provisioning 10 sections for every configuration of the ZNS SSD on F2FS. As shown in Figure 7, vanilla F2FS reserved an average of 8.28x times more than our 10-section setting. This over-provisioning difference can be larger when we format our device with more space. While we use an empirical 10-section setting, there is still room for determining the appropriate over-provisioning space, which we will discuss in Section 7.

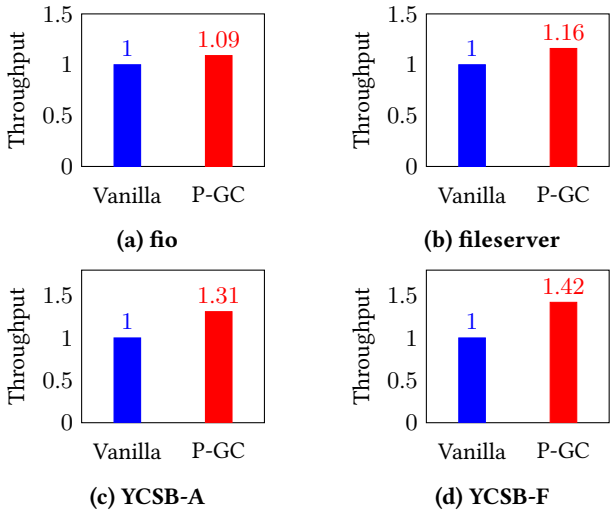


Figure 8: Performance comparison

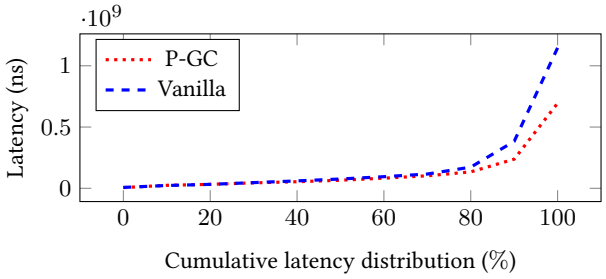
4.2 Performance evaluation

We evaluated the effectiveness of our proposed Parallel GC (P-GC) scheme on a micro- and macro-benchmark set. The microbenchmark we used is fio, a flexible I/O testing application, which issues 4KB random writes for 10 minutes under 80% file system utilization. For the macro-benchmarks, we used filebench and YCSB-A and YCSB-F workloads on RocksDB over 95% file system utilization.

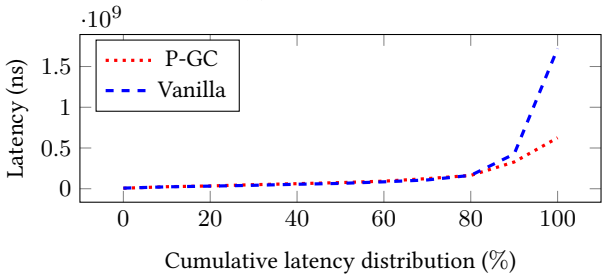
4.2.1 Micro benchmark. Figure 8a shows that fio's performance increased by 9% even with 80% file system utilization. Such performance increase reflects that GC in LFS mode still occurs even under 90% utilization. We also measured how much data was read from the device during buffered write operations. Vanilla F2FS read *58GB*, and our P-GC scheme read *51GB* during the 10 minutes operations. With a similar total number of GC between vanilla and P-GC, P-GC is able to accelerate the random read part of data movement during GC. This result shows the correlation between performance increase and data movement speed of P-GC.

4.2.2 Macro benchmarks. For the macro benchmarks, we used a larger partition size composed of 301 zones. Figure 8b,8c and 8d shows a comparison between vanilla and P-GC with fileserver and YCSB-A, F workloads on RocksDB. P-GC outperforms up to 42% than vanilla. In YCSB-A, F, the average time of read operations (**47.19us** to **34.58us**, **49.75us** to **31.77us**) and write operations (**62.65us** to **48.23us**, **21.50us** to **20.93us**) decreased. In fileserver, both read operations (**350.8mb/s** to **410.4mb/s** and write operations (**346.4 mb/s** to **402.6 mb/s**) performance increased, which illustrates that application performance is less affected in P-GC than in vanilla GC.

During the FG GC, the application gets paused due to the GC thread until the section is entirely freed up. This highlights that FG GC's latency is crucial in determining the application performance. Therefore, in Figure 9, we measured and sorted the latency of FG GC during each YCSB experiment. And, we compared the latency distribution. In Figure 9a, the overall average speed of the P-GC exceeded the Vanilla by 29% and performed more than 2.03× times quicker in the slowest case. For Figure 9b, the average P-GC speed was 7% faster. In the slowest case, P-GC was 2.98× times faster. These experimental outcomes substantiate that P-GC efficiently manages slow GC requests.



(a) YCSB-A



(b) YCSB-F

Figure 9: Latency distribution comparison of FG GC

5 CONCLUSIONS

In this study, we investigate using F2FS LFS mode on ZNS SSDs for identifying the disk layout of F2FS on ZNS SSDs and exploring the trade-off between over-provisioning in F2FS and GC performance under high disk utilization. We set a low empirical over-provisioning space to utilize better the storage space with a new free segment-finding scheme. Our results show that over-provisioning space should be considered for the design of ZNS-based systems, particularly for write-intensive workloads.

Additionally, we propose a Parallel GC (P-GC) scheme to reduce the overhead of foreground GC in F2FS. Our proposed P-GC scheme reduces GC overhead by up to 42% in both micro- and macro-benchmarks.

6 RELATED WORKS

Optimizing file systems for zoned storage has been widely studied in recent years [2, 5, 12, 24]. ZNS SSDs are emerging storage devices that follow the zoned storage models and have gained tremendous research attention. Prior work has suggested several optimization schemes for reducing F2FS GC overhead for ZNS SSDs. Choi et al. [6] proposed a log-structured merge-style zone cleaning scheme that separates the hot and cold segments in a fine-grained manner. They also examined how segment searching improved by increasing the number of worker threads. Lee et al. [15] suggest preempting the ongoing GC to reduce the interference between the cleaning process and the foreground applications. Nevertheless, prior works are based on emulation. This work explores the trade-off of over-provisioning for using F2FS implemented on a ZNS SSD device and suggests a novel parallel FG GC optimization for LFS mode.

7 LIMITATIONS AND DISCUSSIONS

In the course of our research, we have identified multiple promising research directions. Firstly, our approach to determining over-provisioning space in ZNS SSDs was based on empirical values, and we aim to develop a systematic method for making this determination in the future. Secondly, GC overhead can be caused by two factors: GC trigger timing and GC throughput. While we improved GC throughput in this study, the effectiveness of this enhancement depends on the file system's application scenario and GC trigger timing, which is closely related to the over-provisioning space we explored. Therefore, to improve overall GC performance, GC trigger timing and GC throughput must be considered together in future research. Furthermore, we need to investigate the scalability of our P-GC scheme because we used a few threads for accelerating the GC process. We currently only use two threads for P-GC since the file system locking mechanisms make it hard to scale P-GC. We plan to proactively load the required information for GC to handle scenarios where page cache drops occur due to extensive buffered I/O. Lastly, for the proposed free segment search, how many times to search and when to stop searching should be explored based on the characteristics of running workloads. This fact motivated us to explore how we model problem solutions in the future.

ACKNOWLEDGMENTS

The authors would like to thank four anonymous reviewers, and our shepherd Dr. Zhichao Cao for their help in improving the presentation of this paper. We also thank the members of Western Digital for their support. This work is supported by the U.S. National Science Foundation under Grant No.: CCF-1704859 and J. Yang & Family Foundation Fellowship.

REFERENCES

[1] 2015. nvme-cli. https://github.com/linux-nvme/nvme-cli.

[2] Abutalib Aghayev, Theodore Ts'o, Garth Gibson, and Peter Desnoyers. 2017. Evolving ext4 for shingled disks. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*. 105–120.

[3] Jens Axboe. 2005. Fio-flexible i/o tester synthetic benchmark. *URL https://github. com/axboe/fio (Accessed: 2023-01-13)* (2005).

[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 *2021 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*. 728–733.

[6] 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 *HotStorage*. 1–6.

[7] 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.

[8] Facebook. 2019. RocksDB. (2019).

[9] Hans Holmberg. 2020. ZenFS, Zones and RocksDB-Who Likes to Take out the Garbage Anyway.

[10] Jaegeuk Kim. 2022. f2fs-tools. https://kernel.googlesource.com/pub/scm/linux/kernel/git/jaegeuk/f2fs-tools/+/refs/tags/v1.15.0.

[11] Jaeho Kim, Kwanghyun Lim, Youngdon Jung, Sungjin Lee, Changwoo Min, and Sam H Noh. 2019. Alleviating Garbage Collection Interference Through Spatial Separation in All Flash Arrays.. In *USENIX Annual Technical Conference*. 799–812.

[12] Thomas Kim, Jekyeom Jeon, Nikhil Arora, Huaicheng Li, Michael Kaminsky, David G. Andersen, Gregory R. Ganger, George Amvrosiadis, and Matias Bjørling. 2023. RAIZN: Redundant Array of Independent Zoned Namespaces. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 660–673. https://doi.org/10.1145/3575693.3575746

[13] Ryusuke Konishi, Yoshiji Amagai, Koji Sato, Hisashi Hifumi, Seiji Kihara, and Satoshi Moriai. 2006. The Linux implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review* 40, 3 (2006), 102–107.

[14] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. 2015. {F2FS}: A New File System for Flash Storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. 273–286.

[15] Manjong Lee, Jonggyu Park, and Young Ik Eom. 2023. An Efficient F2FS GC Scheme for Improving I/O Latency of Foreground Applications. In *2023 IEEE International Conference on Consumer Electronics (ICCE)*. 1–3.

[16] Jeanna Neefe Matthews, Drew Roselli, Adam M. Costello, Randolph Y. Wang, and Thomas E. Anderson. 1997. Improving the Performance of Log-Structured File Systems with Adaptive Methods. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (Saint Malo, France) *(SOSP '97)*. Association for Computing Machinery, New York, NY, USA, 238–251. https://doi.org/10.1145/268998.266700

[17] Richard McDougall and Jim Mauro. 2005. FileBench. *URL: http://www. nfsv4bat. org/Documents/nasconf/2004/filebench.pdf* (2005).

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

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

[20] Mendel Rosenblum and John K Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)* 10, 1 (1992), 26–52.

[21] Nick Tehrany and Animesh Trivedi. 2022. Understanding NVMe Zoned Namespace (ZNS) Flash SSD Storage Devices. *arXiv preprint arXiv:2206.01547* (2022).

[22] Linux Torvalds. 2023. F2FS forces direct I/O to convert to buffered I/O for zoned device. https://github.com/torvalds/linux/blob/d3f704310cc7e04e89d178ea080a2e74dae9db67/fs/f2fs/file.c#L808.

[23] Qiuping Wang, Jinhong Li, Patrick PC Lee, Tao Ouyang, Chao Shi, and Lilong Huang. 2022. Separating Data via Block Invalidation Time Inference for Write Amplification Reduction in {Log-Structured} Storage. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. 429–444.

[24] Fenggang Wu, Ming-Chang Yang, Ziqi Fan, Baoquan Zhang, Xiongzi Ge, and David HC Du. 2016. Evaluating host aware {SMR} drives. In *8th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*.