Exploring Performance Characteristics of ZNS SSDs: Observation and Implication

Hojin Shin
Dept. of Computer Science
Dankook University
Youngin, Korea
ghwls03s@dankook.ac.kr

Myounghoon Oh
Dept. of Computer Science
Dankook University
Youngin, Korea
snt2426@dankook.ac.kr

Gunhee Choi
Dept. of Computer Science
Dankook University
Youngin, Korea
choi_gunhee@dankook.ac.kr

Jongmoo Choi
Dept. of Computer Science
Dankook University
Youngin, Korea
choijm@dankook.ac.kr

***Abstract*—ZNS SSDs (Zoned NameSpace Solid State Drives) are a new type of SSD that provides various features such as zone concept and host-level flash management. In this paper, we explore how these features affect performance characteristics of ZNS SSDs. To this end, we design an analysis tool based on a real ZNS SSD prototype that allows to evaluate performance under different workloads. Then, we investigate diverse characteristics in terms of parallelism, isolation and predictability. Our observations reveal that 1) requesting I/Os in a large unit is indispensable in ZNS SSDs to obtain parallelism, 2) workloads can be isolated more effectively compared with traditional SSDs, 3) unexpected performance drop is not monitored, and 4) performance differs based on LBAs (Logical Block Addresses), which can be exploited usefully when we devise a new algorithm for ZNS SSDs.**

***Index Terms*—ZNS SSD, Tool, Performance, Analysis**

I. Introduction

These days, many researches have been proposed that uncover SSD internals to enhance performance and I/O determinism [1], [2]. ZNS SSDs are one of these efforts, under standardization by the NVM Express, that expose their address space using the concept of zone [3]. By distributing diverse workloads into different zones, they give an opportunity to reduce WAF (Write Amplification Factor), eventually improving performance and lifespan [4].

Another feature of ZNS SSDs is that flash managements such as mapping and garbage collection are conducted at a host-level. Traditional SSDs deal with flash memory idiosyncrasies such as erase-before-write requirement and limited endurance using FTL (Flash Translation Layer) at a device-level [5]. However, ZNS SSDs move most FTL functionalities into a host, which allows to reduce DRAM usage and over-provisioning area in SSDs. To obtain these merits, many vendors actively announce to launch their ZNS SSDs solutions [6]–[8].

However, ZNS SSDs raise several new issues. One issue is how to manage zones including zone reset and garbage collection at a host-level. In addition, ZNS SSDs have the unique constraint, called sequential write constraint, meaning that data must be written sequentially in a zone [4]. To design an algorithm that manages ZNS SSDs appropriately, we need to understand performance characteristics of ZNS SSDs.

In this paper, we investigate performance characteristics of ZNS SSDs under various workloads. We first secure an ZNS SSD prototype, which is built for research purpose by a company. Then, we design an analysis tool that can configure workloads such as total I/Os, request size, number of threads and access patterns. We also implement software modules that control ZNS SSDs directly at a user layer. Besides, we make our tool run on traditional SSDs so that we can compare characteristics of ZNS SSDs with those of traditional SSDs.

Using the tool, we make the following observations. First, we find that requesting I/Os with a large unit provides considerable performance enhancement in ZNS SSDs. This is because a zone consists of multiple flash chips in general, which enables a large size request to be processed in an interleaved manner. The second observation is that, even though ZNS SSDs show better isolation, interference still exists among zones. This implies that not only the zone concept but also zone hints are useful for boosting I/O determinism. Finally, we observe that unexpected performance drops are not monitored in ZNS SSDs which is a good property to provide predictable services.

The rest of this paper is organized as follows. In Section II, we compare ZNS SSDs with Traditional SSDs and survey related work. Our proposed tool is explained in Section III. Analysis results are discussed in Section IV. Finally, we summarize conclusion and future work in Section V.

II. Background

In this section, we first examine the features of ZNS SSDs. Then, we discuss previous works related to ours.

A. Features of ZNS SSDs

Figure 1 illustrates the differences between ZNS SSDs and Traditional SSDs. SSDs are composed of flash memory, which has several features such as overwrite limitation and limited endurance. To overcome these limitations, various functionalities such as out-place update and mapping, garbage collection, wear-leveling and bad block handling are developed. Now, the question is where these functionalities are deployed.

978-1-7281-8482-1/20/$31.00 ©2020 IEEE

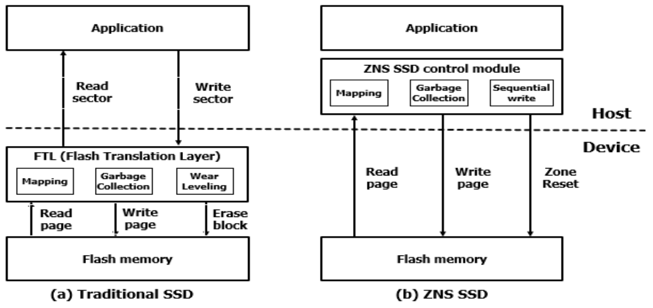


Fig. 1: Comparison between Traditional SSDs and ZNS SSDs.

Conceptually, a computer system can be divided into two levels, a host and device. In traditional SSDs, these functionalities are deployed at a device-level, commonly known as FTL (Flash Translation Layer). The benefit of this approach is that it hides the idiosyncrasy of flash memory and abstracts SSDs as a general block device like disks that can be accessed via sector interfaces. However, the downside is that it may cause a semantic gap, due to the unawareness and redundant software modules between two levels [9]–[11].

ZNS SSDs take an opposite approach, deploying the functionalities at a host-level. They are a kind of OCSSDs (Open Channel SSDs) that expose SSD internals and allow host-level software to control flash memory directly [12]. Compared with OCSSDs, ZNS SSDs have some distinctions in that 1) they divide their address spaces into multiple zones, 2) they require host-level zone managements such as zone reset and garbage collection and 3) data must be written sequentially in a zone.

ZNS SSDs have several merits. First, they can reduce WAF by distributing different workloads into different zones, eventually contributing to enhance performance and lifespan. Also, they give a chance to boost I/O determinism by allocating different users (e.g. dockers) into separated zones. In addition, they decrease DRAM usage and media over-provisioning by moving most FTL functionalities into a host. In general, mapping and garbage collection are moved into a host while device-specific functions such as ECC (Error Correction code) and bad block handling remain at a device-level.

B. Related Work

Previous studies related to our work can be grouped into two categories. The first category is exploiting internal information of SSDs to enhance performance and reliability. Ouyang et al. propose SDF (Software Defined Flash) where host software manages raw flash chips in SSDs directly and flexibly so that it can realize the SSDs raw performance potential [13]. Kim et al. design an automatic stream identification scheme based on program contexts on Multi-Streamed SSDs [14].

Zhang et al. devise a new key-value store, named FlashKV, based on OCSSDs that provides a flash-aware parallel data layout, compaction, caching and I/O scheduling mechanisms [15]. Bjorling et al. introduce LightNVM, a Linux subsystem that supports host-level FTL functionalities for OCSSDs [12]. Lee et al. suggest an interesting approach, application-managed flash, where a file system and FTL incorporate for better performance and for reducing DRAM in SSDs [11].

The second category is investigating ZNS SSDs. ZNS SSDs gain attention recently [2], [3], so studies about them are in an initial stage. Chung introduces several ZNS SSD prototypes and demonstrates their benefits in data center [16]. Choi et al. design a new LSM (Log-Structure Merge) style garbage collection scheme for reclaiming a zone in ZNS SSDs [17]. Linux communities announce new facilities such as ZNS-aware file system, device mapper, block layer and utilities for ZNS SSDs [4].

Note that there are different types of zoned storage devices that also provide the zone concept. ZNS SSDs are one example of zoned storage and another well-known example is SMR (Shingled Magnetic Recording) HDDs [18], [19]. Since SMR HDDs and ZNS SSDs have comparable features such as zone management and sequential write constraint, schemes designed for SMR HDDs are also applicable to ZNS SSDs. For instance, the idea, called Gear compaction, proposed in [20] that merges all SSTables in a zone at once can be used for zone reclaiming in ZNS SSDs.

III. Methodology

To evaluate performance characteristics of ZNS SSDs quantitatively, we design an analysis tool, presented in Figure 2. It consists of three key software components, namely, workload generator, performance monitor and generic SSD manager.

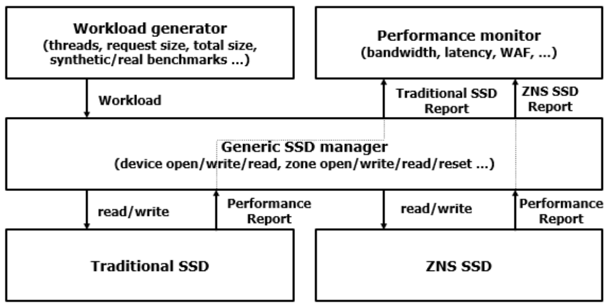


Fig. 2: Details of Analysis Tool.

The workload generator creates I/O requests with diverse patterns such as sequential and random using synthetic and real benchmarks (e.g. fio benchmark [21]). It also allows to configure several control parameters including the number of threads, total I/O size, each request size and LBAs. The performance monitor measures metrics such as latency, bandwidth and WAF while executing workloads. In addition, it supports a tracing function that shows how each I/O request is handled in a time and space dimension.

The generic SSD manager manipulates ZNS SSDs directly and supports interfaces for zone reset/open/close and page read/write. In this study, we use a real ZNS SSD prototype that is offered by a ZNS SSD vendor. The capacity of this prototype is 1TB, which is divided into 1024 zones whose size is 1GB, as summarized in Table I. In fact, details of this prototype such as flash type (e.g. MLC or TLC) and

TABLE I: ZNS SSD prototype specification

Item	Specification
SSD Capacity	1TB
Size of a Zone	1GB
Number of Zones	1024
Interface	PCIe Gen3
Protocol	NVMe 1.2.1

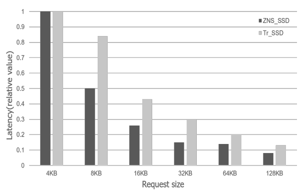
the number of channels and ways are unknown since they are proprietary information of the vendor. In other words, our study is a kind of blackbox approach, inferring characteristics from input and output relations. In addition to ZNS SSDs, the tool also supports management interfaces for traditional SSDs for comparison purpose, this is why we refer to it as a generic manager.

IV. Analysis

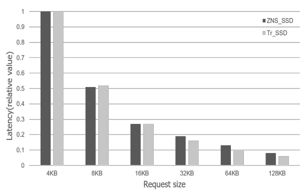
In this section, we present analysis results with an aspect of parallelism, isolation and predictability. In addition, we discuss implication of our observations for both host software and ZNS SSD developers.

A. Parallelism

Figure 3 displays the write and read latency when we change the request size ranging from 4KB to 128KB. For this experiment, we implement a worker thread that writes or reads a 1GB file sequentially with different I/O request sizes given as an argument, denoted in the X-axis. Note that the Y-axis corresponds to values relative to that of 4KB, while the actual measured latency is given in Table II. Since the ZNS SSD and traditional SSD device used in this study have different specifications, comparing them using actual measured values is unfair. Besides, the goal of this study is analyzing performance characteristics of ZNS SSDs, not comparing directly with traditional SSDs, that leads us to draw this figure with relative values to reveal characteristics more clearly.



(a) Write



(b) Read

Fig. 3: Latency results according to different request sizes (Relative to 4KB).

TABLE II: Write and Read latency (in seconds)

SSD	Request size (KB)					
	4KB	8KB	16KB	32KB	64KB	128KB
ZNS SSD (write)	31.4	15.9	8.14	4.77	4.29	2.6
Tr SSD (write)	8.42	7.05	3.59	2.5	1.7	1.12
ZNS SSD (read)	43.5	22.1	11.9	8.44	5.87	3.48
Tr SSD (read)	30.6	15.8	8.13	4.9	3.03	1.89

From Figure 3, we can observe that the request size impact on the write latency greatly in ZNS SSDs. For instance, when we change the request size from 4KB to 8KB, ZNS SSDs can reduce latency by 50%, while the reduction is 16% in traditional SSDs. For the read case, ZNS SSDs and traditional SSDs shows similar trends.

Larger request size accompanies several advantages. First, it can reduce the total number of requests, which decreases the queuing delay. In addition, it can make it possible to utilize the channel and way interleaving in SSDs. Hence, when we double the request size, the latency decreases a half except the write case in traditional SSDs. We think that the exception is mainly due to the DRAM write buffer. Another thing we observe is that, at first, we expect that performance gain obtained by larger request size is higher in ZNS SSDs since they usually map a zone into multiple channels to maximize parallelism. But, traditional SSDs also deal with larger requests efficiently, showing similar trends as ZNS SSDs.

This observation implies that host software developers for ZNS SSDs need to design an algorithm that triggers I/Os in larger request size. Random reads may incur more than 10 times performance degradation. Reading in larger unit might be better even though there exist unnecessary data partially in the unit. Using ZNS SSDs for writing log is not a good idea. For ZNS SSD developers, they need to design a mechanism for amortizing write requests in smaller unit. Using DRAM write buffer is not a good choice since it is an opposite direction that ZNS SSDs pursue.

B. Isolation

To evaluate the isolation capability of ZNS SSDs, we execute the worker thread with multiple other threads at the same time. Specifically, we run all threads on different zones and measure the IOPS of worker, as shown in Figure 4.

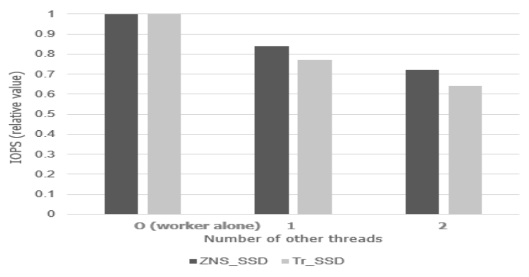


Fig. 4: Effect of other threads.

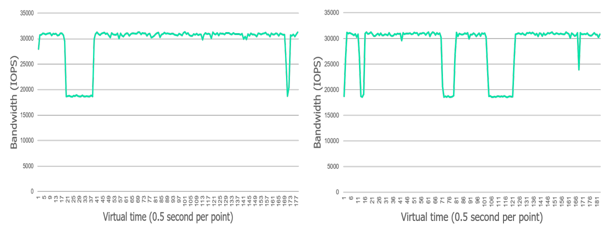
Figure 4 reveals that ZNS SSDs show better isolation than traditional SSDs. For instance, when we run two other threads simultaneously, the performance of the worker thread drops 28% in ZNS SSDs, while that becomes 36% in traditional SSDs. This is because traditional SSDs have more shared resources such as FTL and overlapped address spaces.

However, the isolation capability is rather weaker than our expectation. At first, we anticipate that the worker is not interfered that much by other threads since they access different zones. But our observation uncovers that zones share resources such as channels and chips in ZNS SSDs. This is inevitable since the numbers of channels and chips are much smaller than the number of zones. Also, there is a tradeoff between parallelism in a zone and isolation among zones.

We carefully argue that there exist feasible solutions that reinforce the isolation capability. We suggest ZNS SSDs developers to export information about zones that are more isolated (e.g. zone 1 and 2 are mapped into different channels while zone 1 and 3 are mapped into same ones). This information will become a valuable hint for host software developers. Another interesting solution is skewing among zones like track skewing in HDDs [22].

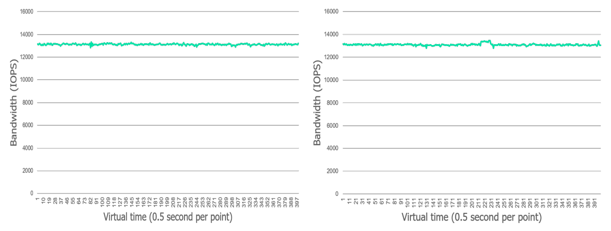
C. Predictability

One notorious downside of traditional SSDs is the unexpected performance degradation due to the garbage collection triggered by FTL in SSDs. To examine this phenomenon, we conduct an experiment that writes a 10GB file from the initial utilization of 60% and 90% in traditional SSDs and ZNS SSDs, shown in Figure 5 and Figure 6, respectively. Each point in figures are bandwidth values measured at every 0.5 second.



(a) 60% utilization (b) 90% utilization

Fig. 5: Unexpected performance drop while writing in Traditional SSDs.



(a) 60% utilization (b) 90% utilization

Fig. 6: Unexpected performance drop while writing in ZNS SSDs.

Unexpected performance degradation indeed occurs in traditional SSDs, which happens more frequently in a higher utilized condition as shown in Figure 5. On the contrary, such degradation is not monitored in ZNS SSDs as presented in Figure 6. This observation uncovers that ZNS SSDs are a good basis to enhance predictability. Note that this result does not imply that there is no garbage collection overhead in ZNS SSDs. Instead, it implies that the overhead can be controlled and can be hidden by employing a background and/or preemptive mechanism at a host-level.

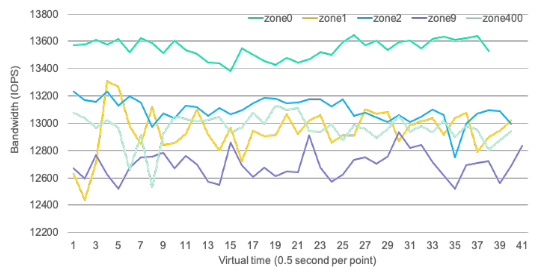


Fig. 7: Performance differences among zones.

Figure 7 presents an interesting result observed by our study. It shows that each zone provides different performance in ZNS SSDs. Specifically in this experiment, executing the worker thread on zone 0 provides the best performance, while executing it on zone 9 performs the worst. This observation indicates that, by exposing address spaces in ZNS SSDs, they change from UMA (Uniform Memory Access) to NUMA (Non-Uniform Memory Access) architecture. For instance, an address range mapped with more worn flash memory might provide longer latency due to ECC retries than other ranges consistently. These characteristics can be utilized by host software developers so that they place more frequently accessed data in faster zones.

V. Conclusion

This paper investigates how features of ZNS SSDs affect performance characteristics. One feature is the zone concept, and our analysis reveals that it allows better isolation among zones and requires larger request size to obtain parallelism in a zone. The second feature is host-level flash management, which enhances predictability and makes ZNS SSDs behave like a NUMA architecture. Our observations can be exploited usefully for both host software and ZNS SSD developers. For instance, ZNS SSD developers can give hints about zone and channel mapping, which can be used beneficially to reinforce I/O determinism.

There are three directions for future research. The first direction is making use of our observation to devise a host-level algorithm for ZNS SSDs. We are currently developing and verifying a new zone garbage collection scheme that reads data in a group manner. The second direction is analyzing I/O behavior at an application level and examining how it is matched with the characteristics of ZNS SSDs. We actually gather I/O traces of LevelDB and RocksDB [23] and analyze how these traces affect ZNS SSD management modules. The final direction is studying the effect of ZNS SSDs in distributed storage backends [24], [25].

Acknowledgment

We thank the anonymous reviewers for their valuable feedback. This research was supported in part by the MISP (Ministry of Science, ICT Future Planning), Korea, under the National Program for Excellence in SW (2017-0-00091), supervised by the IITP (Institute for Information communications Technology Planning Evaluation) and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2019R1F1A1062284).

REFERENCES

[1] I. Picoli, N. Hedam, P. Bonnet, P. Tzn, “Open-Channel SSD (What is it Good For)”, CIDR, 2020.
[2] B. Matias, “From Open-Channel SSDs to Zoned Namespaces”, USENIX VAULT, 2019.
[3] D. Black, “The NVMe Standard: The Next Five Years”, NVMe Developer Days, Dec. 2018.
[4] NVMe Zoned Namespaces, https://zonedstorage.io/introduction/zns/.
[5] A. Gupta, Y. Kim and B. Urgaonkar, “DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings”, ASPLOS, 2009.
[6] Western Digital Champions Zoned Storage, An Open Initiative to Redefine Data Centers for Zettabyte Scale, https://www.westerndigital.com/company/newsroom/press-releases/2019/2019-06-11-western-digital-champions-zoned-storage.
[7] SK hynix Demonstrates Industrys First ZNS-based SSD Solution for Data Centers, https://news.skhynix.com/sk-hynix-demonstrates-industrys-first-zns-based-ssd-solution-for-data-centers-2/.
[8] J. Hwang, “Towards Even Lower Total Cost of Ownership of Data Center IT Infrastructure”, NVRAMOS Workshop, 2019, http://www.sigfast.or.kr/nvramos/nvramos19/presentation/nvramos19-samsung.pdf.
[9] Z. Shen, F. Chen, Y. Jia and Z. Shao, “DIDACache: A Deep Integration of Device and Application for Flash Based Key-Value Caching”, USENIX FAST, 2017.
[10] J. Yang, N. Plasson, G. Gillis, N. Talagala and S. Sundararaman, “Don’t stack your Log on my Log”, USENIX INFLOW, 2014.
[11] S. Lee, M. Liu, S. Jun, S. Xu, J. Kim and Arvind, “Application-managed flash”, USENIX FAST, 2016.
[12] M. Bjorling, J. Gonzalez and P. Bonnet, “The Linux Open-Channel SSD Subsystem”, USENIX FAST, 2017.
[13] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang and Y. Wang, “SDF: Software defined flash for web-scale internet storage systems”, ASPLOS, 2014.
[14] T. Kim, D. Hong, S. Hahn, M. Chun, S. Lee, J. Hwang, J. Lee and J. Kim, “Fully Automatic Stream Management for Multi-Streamed SSDs Using Program Contexts”, USENIX FAST, 2019.
[15] J. Zhang, Youyou Lu, J. Shu and X. Qin, “FlashKV: Accelerating KV Performance with Open-Channel SSDs”, ACM Transactions on Embedded Computer systems, 2017.
[16] W. Chung, “Benefits of ZNS in Datacenter Storage Systems”, Flash memory summit, 2019.
[17] G. Choi, M. Oh, K. Lee, J. Choi, J. Jin and Y. Oh, “A New LSM-style Garbage Collection Scheme for ZNS SSDs”, USENIX Hotstorage, 2020.
[18] F. Wu, M. Yang, Z. Fan, B. Zhang, X. Ge, and D. Du, “Evaluating Host Aware SMR Drives”, USENIX Hotstorage, 2016.
[19] A. Aghayev and P. Desnoyers, “SkylightA Window on Shingled Disk Operation”, USENIX FAST, 2015.
[20] T. Yao, J. Wan, P. Huang, Y. Zhang, Z. Liu, C. Xie and X. He, “GearDB: A GC-free Key-Value Store on HM-SMR Drives with Gear Compaction”, USENIX FAST, 2019.
[21] Flexible I/O, https://fio.readthedocs.io/en/latest/fio_doc.html.
[22] R. Arpaci-Dusseau and A. Arpaci-Dusseau, “Chapter 37 in Operating Systems: Three Easy Pieces”, http://pages.cs.wisc.edu/ remzi/OSTEP/.
[23] Z. Cao, S. Dong, S. Vemuri and D. Du, “Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook”, USENIX FAST, 2020.
[24] S. Weil, S. Brandt, E. Miller and D. Long, “Ceph: A Scalable, High-Performance Distributed File System”, USENIX OSDI, 2006.
[25] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. Ganger and G. Amvrosiadis, “File Systems Unfit as Distributed Storage Backends: Lessons from 10 Years of Ceph Evolution”, SOSP, 2019.