Achieving Performance Isolation in Docker Environments with ZNS SSDs

Anonymous for Blind review

Abstract

Checkpointing is an essential ingredient in Docker environments for fast recovery and migration. However, it incurs serious interference with normal I/Os and other Docker containers running in parallel. To address this problem, we propose a new checkpoint scheme based on Zoned Namespace (ZNS) SSDs. ZNS SSDs are a novel type of SSDs that support a zone interface, allowing workload separation and performance isolation. On ZNS SSDs, we devise two techniques: 1) independent zone allocation and 2) striping across multiple zones to achieve both interference mitigation and Quality of Service (QoS) support. Real implementation-based experiments show that our proposal can reduce interference by up to 40% when multiple dockers perform checkpoints concurrently.

CCS Concepts: • **Information systems** → **Information storage systems**; • **Computer systems organization** → **Reliability**.

Keywords: ZNS SSD, Docker, Checkpointing, Performance Interference

ACM Reference Format:
Anonymous for Blind review. 2023. Achieving Performance Isolation in Docker Environments with ZNS SSDs. In *Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '23), May 08–12, 2023, ROME, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Docker is a software platform that offers a variety of benefits, including lightweight virtualization, low-performance overhead, and fast deployment. In the Docker environment, checkpointing is actively used to provide fast recovery from faults and enables migration for load balancing [4, 19, 21]. However, because checkpointing is a heavy I/O job, it may interfere with the performance of other applications that share storage.

CHEOPS '23, June 03–05, 2023, ROME, Italy

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

This paper explores the use of Zoned Namespace (ZNS) SSDs to mitigate performance interference in Docker environments. ZNS SSDs are recently emerging SSDs that support a zone interface and provide the advantage of reduced write amplification factor (WAF) through workload separation [2, 12]. In particular, *small zone* ZNS SSDs give a foundation for reducing performance interference, as each zone is mapped to a different channel/way within the internal structure of SSDs [1, 7, 8].

In this paper, we compare the performance of a real small zone ZNS SSD device to a traditional SSD device. Experimental results show that performance interference is noticeable as the number of threads increases in traditional SSD, while in ZNS SSD, this interference is eliminated when allocating each thread to an independent zone. However, our experiments also demonstrate that there is performance degradation when using small zone ZNS SSD compared to traditional SSD.

In order to alleviate performance interference from Docker container checkpointing and overcome the performance degradation problem of ZNS SSD simultaneously, we suggest two techniques: (1) independent zone allocation to each Docker container for preventing interference and (2) zone striping based on multiple zones for improving performance. Zones can be allocated through either static or dynamic methods. We choose the static approach due to its greater predictability. Along with this approach, each Docker container specifies its required bandwidth, and we allocate the appropriate number of zones to satisfy the requirement. To improve performance, we distribute writes in an interleaved manner, achieving higher throughput.

We implement our proposed techniques by modifying the Linux software tool CRIU (Checkpoint and Restore In Userspace) [5]. The modified CRIU provides not only two techniques but also other facilities such as ZNS SSD controls (e.g. zone reset) and Docker interfaces (e.g. required bandwidth). Evaluation results show that our independent zone allocation technique can prevent normal I/Os from being interfered by checkpointing. It also mitigates interference among multiple checkpoints conducted in parallel. In addition, the striping technique makes container checkpointing performance in ZNS SSDs comparable to traditional SSDs.

The contributions of this paper are as follows.

- To the best of our knowledge, this is the first study that utilizes ZNS SSDs for reducing performance interference in Docker environments.

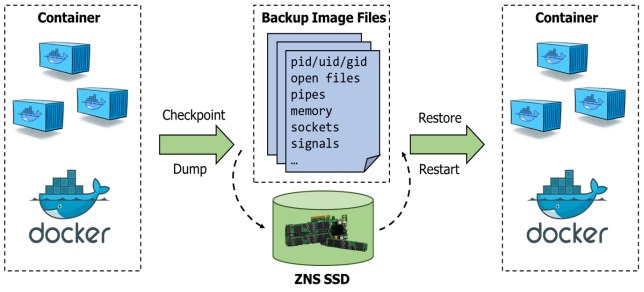

Figure 1. Docker container checkpoint and restore

- We devise two techniques, independent zone allocation and striping on multiple zones, to reap the benefit of performance isolation of ZNS SSDs and to overcome the limited bandwidth of a zone.
- We present various evaluation results including characteristics of ZNS/traditional SSDs and the effectiveness of two techniques using real devices.

2 Background

2.1 Zoned Namespace (ZNS) SSDs

A solid-state drive (SSD) is a data storage that uses flash-based memory. Due to the nature of flash memory such as erase-before-write and limited endurance, SSDs make use of a software layer, called Flash Translation Layer (FTL), to address these issues. In addition, to improve performance, modern SSDs utilize internal parallel units such as multiple channels, ways, planes, and so on.

ZNS SSDs [17] are one on the most recently proposed SSDs, having two characteristics: 1) They divide the whole logical block address (LBA) into zones and provide a new zone interface and 2) a host has complete control over data stored in zones [1–3, 7–10, 14]. ZNS SSDs can provide several benefits over traditional SSDs that support the conventional block interface only. By separating different workloads into different zones, ZNS SSDs can reduce WAF, which gives a positive impact on performance and reliability. The host level direct management can reduce required FTL functionalities, which eventually lowers SSD cost by decreasing device-level DRAM and over-provisioning area.

This new type of SSDs can be classified into two categories: *large zone* and *small zone* ZNS SSDs. Large zone ZNS SSDs support a larger zone size (e.g., 1GB), which is mapped into multiple channels/ways. On the contrary, small zone ZNS SSDs provide a smaller zone size (e.g., 72MB), and zones are mapped in fewer channels/ways. Our interest is to mitigate performance interference, so this paper mainly focuses on small zone ZNS SSDs.

2.2 Docker container checkpoint

Container-based virtualization such as Docker is widely employed in cloud environments. Containers enable fast application deployment with lower overhead than virtual machines, so they can efficiently manage software services in a cloud. In order to manage the container lifecycle, checkpoint and restore (C/R) are indispensable in a virtual environment with a large number of containers.

Docker integrates Linux software tool CRIU (Checkpoint and Restore In Userspace) for container checkpoint and restore. Figure 1 shows the overall process of container checkpoint and restore. In an environment where multiple Docker containers are running, the checkpoint command stops the container and dumps the current state. During the checkpoint process, the container status is saved as image files to the SSD. This status includes pid/uid/gid, open files, pipes, memory pages, network sockets, signals, and so on. After checkpointing, we can restart the container at the same host (fault recovery case) or at another host (load balancing case) from the previous checkpointed status, by reading the saved image files from the storage.

3 Motivation

Traditional SSDs have been commonly used as storage for checkpoints. However, they lack support for workload isolation per Docker. This can lead to interference between different Dockers, potentially causing unexpected delays in cloud environments that require guaranteed QoS. In contrast, ZNS SSDs have the potential to support such requirements. To explore these characteristics, we do experiments using both traditional SSD and ZNS SSD.

We set up an experimental environment using two SSD devices: traditional SSD and ZNS SSD. Both devices have the same internal architecture, 8-channels and 2-ways, and provide the same capacity, 2 TB. But, they differ in that the traditional SSD device provides the conventional block interface while the ZNS SSD device provides the ZNS interface. The zone size of the ZNS SSD device is 72MB, which is mapped only one channel and one way.

Figure 2 presents a performance comparison between traditional SSD and ZNS SSD using the FIO [6] benchmark. In this experiment, we make five threads that write data sequentially to both SSDs. Each thread starts at different times, 0, 30, 60, 90, and 120 seconds. We throttle the write performance of each thread to 500MiB/s to observe under a more realistic cloud scenario [15, 18].

Observation 1: *Traditional SSD experiences a noticeable performance drop by up to 60% due to interference by other threads, whereas ZNS SSD does not exhibit such performance degradation.* As shown in Figure 2 (a), the write throughput of traditional SSD decreases significantly from 500MiB/s to 200MiB/s as other threads run in parallel. This performance

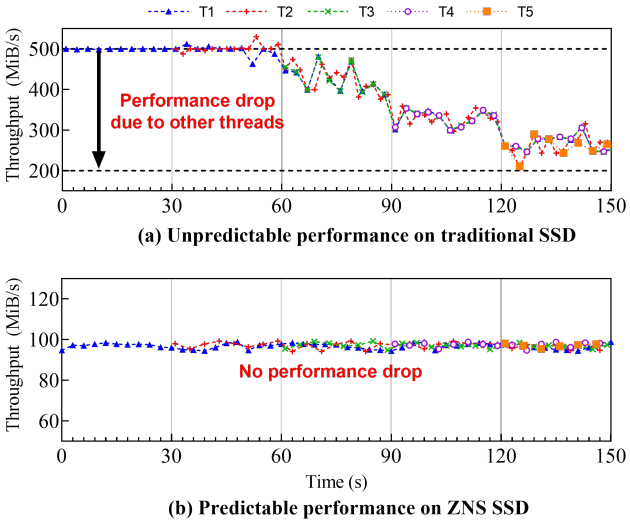

Figure 2. Performance characteristics of traditional SSDs and ZNS SSDs

degradation is because traditional SSDs cannot prevent interference from other users competing for the same resources. In contrast, Figure 2 (b) shows ZNS SSD can support performance isolation, maintaining stable performance even when five threads access the same hardware device concurrently. This is because each thread accesses a different zone, which is mapped into different channel/way in ZNS SSD.

Observation 2: *Achieving performance isolation using ZNS SSD comes with the trade-off of considerably lower throughput.* Even though ZNS SSD can support performance isolation under concurrent jobs, this can result in quite low performance. As shown in Figure 2 (b), ZNS SSD shows roughly 100MiB/s write bandwidth. This is significantly lower than traditional SSD. This is because traditional SSD can use all of the internal channels/ways, while ZNS SSD can only utilize a single channel as one zone is mapped to one channel/way.

Note that the results of ZNS SSD, shown in Figure 2 (b), come from *small zone* ZNS SSDs in specific. We have performed the same experiment with a large zone ZNS SSD device and have observed that it shows similar trends as traditional SSDs since a zone is mapped to multiple channels/ways like traditional SSDs. This is why we focus on small zone ZNS SSDs for achieving performance isolation using ZNS SSDs in Docker environments. We expect that software-level approaches such as fair-share I/O scheduling can be applicable to large zone ZNS SSDs, but we leave this issue as s future work.

These observations motivate our work and give two questions. One is how we can prevent interference of Docker checkpointing by leveraging small zone ZNS SSDs, a compelling foundation for performance isolation. The other is how we can overcome performance degradation when using small zone ZNS SSDs. These two questions will be discussed in the design section.

4 Design

In response to the above two questions, we propose two techniques: 1) independent zone allocation and 2) striping based on multiple zones. For these two techniques, we need to address five challenges, which will be elaborated in the following two subsections.

4.1 Independent zone allocation

Identifying the independent zone First, how can we define independent zones and detect them? An independent zone means there is no performance interference from each other zone. Figure 3 illustrates our small zone ZNS SSD internal architecture. There are 8 channels and 2 ways, and a single zone is mapped into one channel/way.

To maximize parallelism, ZNS SSDs commonly map zones into channels/ways in a channel-interleaved manner first and then in a way-interleaved manner. In this hardware configuration, *zone 0* is independent with *zone 1 - 7*, but not *zone 8* since they share the same *channel 0*. Also, *zone 0* is independent with *zone 9 - 15* but not *zone 16* because they share both the same *channel 0* and same *way 0*. Note that, with recent advances, SSDs can perform to the maximum speed that each channel can support all way mapped. We observe that performance interference of channel conflict is not significant compared to way conflict. Therefore, we decide that zones in the same way are dependent while others are independent.

In our design, zones are divided into M different independent zone groups, where M is defined as $channels * ways$. A zone i is included into a group labeled as $i \% M$. If one Docker is using zones in a group x, then the other zones in the same group cannot be used by other Dockers. This means that

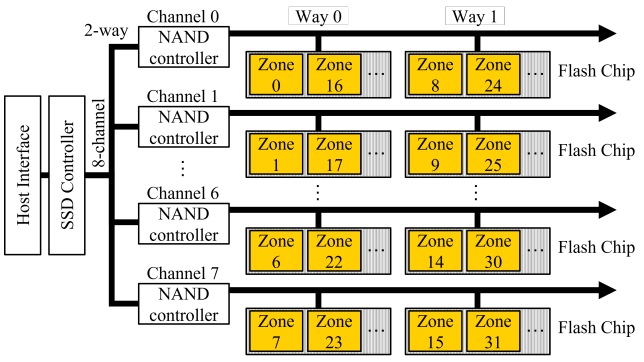

Figure 3. ZNS SSD internal architecture (8 channels, 2 ways, and a zone is mapped in single channel and way)

only one Docker can access zones in one independent zone group at a time.

Approaches of allocating independent zone Second, which approaches do we choose for allocating independent zone groups to a Docker? There are two approaches regarding the allocation of independent zone groups: static allocation and dynamic allocation. The static approach is to assign specific independent zone groups to a Docker. In contrast, the dynamic approach allocates idle zone groups to a Docker at the moment, similar to the on-demand manner. The former has excellent performance predictability, but there can be idle groups. On the other hand, the latter may utilize idle groups but may have a possibility of interference especially for reading. As we consider performance predictability as a top priority, we adopt a static approach in this study.

4.2 Striping across multiple zones

In this section, we will discuss the remaining three challenges related to our second technique, striping based on multiple zones.

Deciding the appropriate number of zones to be allocated For the third question, how many zones will be allocated for striping based on multiple zones to a Docker? We consider both the zone capability and Docker requirement at the same time. For instance, suppose a Docker requests a maximum bandwidth of 500MiB/s, as shown in Figure 2. In this case, we allocate five channels to the Docker because one channel can support around 100MiB/s. Then, the proposed striping technique distributes data across multiple allocated zones in a round-robin style, which enables to overcome the limited performance of a zone in small zone ZNS SSDs.

Managing zone to Docker container mapping The next question is about how to manage mapping information between a checkpointing image and zones. After allocating zones to a Docker, our scheme writes a checkpointing image across multiple zones in a round-robin style. Hence, without the need for an additional mapping data structure, the checkpoint image can be mapped into zones using an equation where an image offset is converted into a zone number based on the number of allocated zones and the size of a write granularity in a zone.

Admission control for multiple containers Our final challenge is an admission control for different Dockers. To guarantee the requirements of all Dockers, we have to equip with enough independent zone groups. However, in ZNS SSDs, the number of groups is limited according to the number of channels and ways. In the case of independent zone groups being insufficient, there are two options, reject a new Docker's request or serve it in a fair-share manner without guaranteeing QoS. In this study, we use the second option and leave the first one as future work.

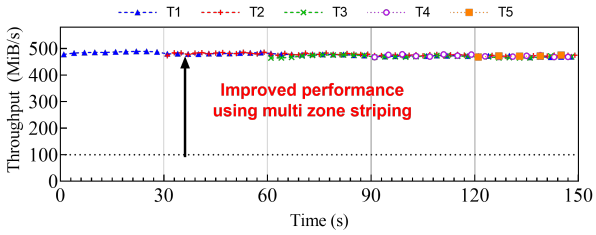

Figure 4. Improved performance of ZNS SSDs using multi-zone striping

5 Evaluation

We modified a Linux checkpointing tool, called CRIU, to implement our proposed techniques. Specifically, we have modified CRIU version 3.8.13. This includes our two techniques, independent zone allocation and zone striping, as well as host-level ZNS SSD direct control and Docker interface, enabling seamless integration of the modified CRIU with ZNS SSDs and Dockers.

We evaluate the performance of our proposal under the following testbed and workloads:

Testbed We use an Intel Core i5-4440 CPU @ 3.10GHz server (Ubuntu 20.04), 32GB DRAM, and two NVMe SSDs including 2TB traditional SSD and ZNS SSD, as mentioned in section 3. The zone, whose size is 72MB, is mapped into one channel and way.

Workloads We build several Docker images using various applications such as Redis, Nginx, MongoDB, and MySQL. We assume that each Docker requires 500MiB/s bandwidth for checkpointing. For generating normal I/Os, we run the FIO benchmark in a separate Docker.

5.1 Zone striping impact on performance

Figure 4 shows how our zone striping technique impacts on the performance of small zone ZNS SSDs. For this evaluation, we conduct the same experiment as described in Figure 2. Each five sequential write threads start every 30 seconds. The only difference is applying our striping technique when accessing ZNS SSDs. We allocate 5 independent zones for each thread to support the 500MiB/s bandwidth and distribute data across the allocated zones.

Note that there are 5 threads and each thread requires 5 independent zones. It implies that we need a total of 25 independent zone groups. However, there exist 18 groups in our experimental ZNS SSD device. Hence, for the first three threads, we can allocate free independent zone groups for them. But, when the fourth thread arrives, we serve it in a fair-share manner by allocating 3 zones from the remaining three free groups, while 2 zones from groups used that have already been allocated to previous threads. We select groups mapped into the smallest threads first for fairness. For the

fifth thread, we allocate zones from groups used again in a fair-share manner.

In Figure 2 (b), we have observed that the throughput for a single-thread in ZNS SSDs was significantly lower than that of traditional SSDs, as marked by the dotted line in Figure 4. However, if we stripe data across multiple zones, the write throughput of ZNS SSDs increases almost to 500MiB/s, while providing stable performance regardless of the number of concurrent threads. There is small degradation when the number of threads becomes 4 and 5, but our fair-share manner makes it possible to avoid a severe conflict channel or way, eventually preventing a performance drop observed in traditional SSDs. This shows that with our proposed scheme applied, small zone ZNS SSDs can provide both performance isolation and improved performance.

5.2 Workload separation using zone allocation

We now discuss the result of our independent zone allocation scheme that enables workload separation, leading to high predictability. We evaluate whether independent zone allocation can support performance isolation of Docker checkpoint in other tasks as named to normal I/Os. As we mentioned in section 3, we throttle the bandwidth to 500MiB/s.

Figure 5 shows a comparison between traditional SSDs and ZNS SSDs when Docker checkpointing is performed parallel to normal I/Os generated by FIO. On traditional SSDs, checkpointing and normal I/Os can go into same channels and ways, resulting in throughput fluctuations. The throughput of normal I/Os varies from 100 to 900 MiB/s, which significantly produces long tail latencies.

Conversely, but as expected, we can hardly see performance interference caused by Docker checkpointing on ZNS SSDs. This is because our proposal can separate Docker checkpointing from normal I/Os by allocating different zones. Along with the advantage of performance isolation from different zone allocation, Figure 5 also shows that ZNS SSDs can support improved performance, comparable to traditional SSDs.

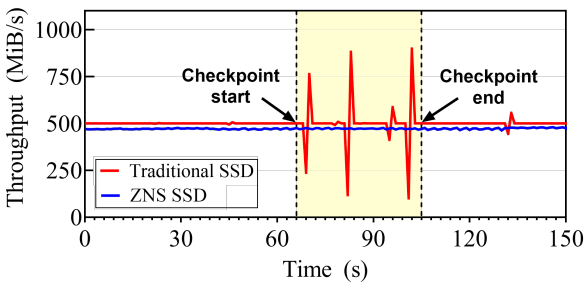

Figure 5. Docker checkpoint impacts on traditional SSDs and small zone ZNS SSDs

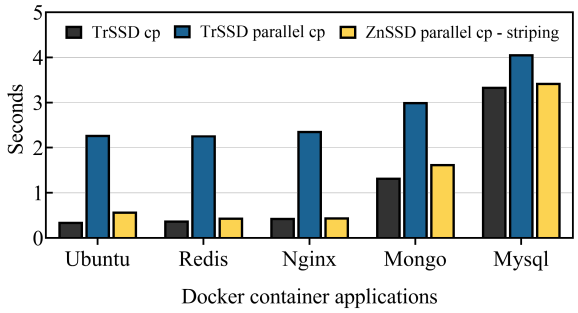

Figure 6. Performance comparison when multiple checkpoints are conducted in parallel.

5.3 Consolidated impact to multiple Docker checkpointing job

Figure 6 shows a consolidated result applied to our proposal. In this evaluation, we deploy five Docker containers, each running Ubuntu, Redis, Nginx, MongoDB, and MySQL, respectively. In this figure, *TrSSD cp* represents the checkpoint latency of a single Docker, denoted in the x-axis, on traditional SSDs. *TrSSD Parallel cp* and *ZnSSD parallel cp striping* are the checkpoint latency of a single Docker under traditional SSDs and ZNS SSDs, respectively, when five Docker checkpoints are performed concurrently.

As the size of Docker image increases (Ubuntu: 63.1MB, Redis:105MB, Nginx: 133MB, Mongo: 303MB, and Mysql: 447MB), the checkpoint latency also increases. When multiple Dockers are checkpointed concurrently, the performance degradation is significant on traditional SSDs. This stems from the lack of performance isolation in traditional SSDs. However, when using ZNS SSDs, the workload of five parallel checkpoints shows comparable performance to that of a single checkpoint on traditional SSDs. This demonstrates that zone-aware allocation and striping based on multiple zones can support isolation even under multi-tenant environments.

6 Related Work

Existing studies related to our research can be classified into two categories. First, based on ZNS SSDs, several research work on performance improvement or workload isolation [1, 2, 7, 8, 10, 14]. Bjørling et al. [2] advocate ZNS SSDs since they can avoid device-side and host-side storage overhead, referred to as block interface tax. They also suggest a zone-aware storage backend for RocksDB. Han et al. [7] show how flash-level copy-offloading can overcome host-level file system overhead on ZNS SSDs. They use large zone ZNS SSDs to utilize the SSD-internal flash chip parallelism but do not cover channel-level isolation.

Bae et al. [1] evaluate performance characteristics of both large zone and small zone SSDs. In addition, they suggest

a novel I/O scheduler that can mitigate zone contention in small zone ZNS SSDs. Im et al. [8] examine the lack of internal parallelism of small zone ZNS SSDs and propose I/O mechanisms applied to ZenFS to exploit external parallelism. However, they do not consider the usage of ZNS SSDs in a cloud environment.

Second, there have been some studies about performance isolation and QoS support in the Docker environment [11, 13, 16, 20]. Kwon et al. [11] propose hardware and software support to isolate noisy neighbor containers and avoid I/O interference. Xavier et al. [20] discuss performance interference from disk-intensive workloads in containerized platforms where QoS is a crucial factor. They propose a workload-balanced scenario to prevent performance degradation.

Li et al. [13] address the performance isolation optimization to allocate the storage resource according to their performance behaviors under resource competition situations. McDaniel et al. [16] point out that existing solutions, such as guaranteeing QoS to mitigate performance loss, do not target managing I/O contention. To control the I/O of Docker containers, they suggest cluster/node approaches. These prior works involve performance isolation and QoS control in Docker environments, similar to our motivation, but they primarily investigate the problems in the context of traditional SSDs.

Contrary to the aforementioned studies, our work is the first proposal about applying ZNS SSDs in a Docker environment. This opens up the possibility that ZNS SSDs can be actively utilized in cloud environments.

7 Conclusion

In this work, we explore the feasibility of ZNS SSDs in Docker environments. For supporting Docker isolation, traditional schemes make use of I/O scheduling, load balancing, and resource reservation. On the contrary, our scheme takes a new approach, making use of ZNS SSDs. Based on ZNS SSDs, we devise the independent zone allocation to support isolation and striping across multiple zones to provide comparable performance to traditional SSDs.

Our work leaves several opportunities for improvement. We aim to complete our checkpoint scheme and conduct a systematic evaluation. For that, we will add admission control under a multi-tenant environment when free zone groups are not sufficient. We also plan to compare our scheme and software-based interference mitigation technology.

References

[1] Hanyeoreum Bae, Jiseon Kim, Miryeong Kwon, and Myoungsoo Jung. 2022. What You Can't Forget: Exploiting Parallelism for Zoned Namespaces. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems* (Virtual Event) *(HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 79–85.

[2] 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 2021, July 14-16, 2021*, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, USA.

[3] 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.

[4] Sang-Hoon Choi and Ki-Woong Park. 2022. iContainer: Consecutive checkpointing with rapid resilience for immortal container-based services. *Journal of Network and Computer Applications* 208 (2022), 103494.

[5] CRIU community. 2017. Checkpoint/Restore in Userspace(CRIU). http://criu.org

[6] FIO 2022. Flexible I/O Tester. https://github.com/axboe/fio.

[7] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. 2021. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction.. In *OSDI*. 147–162.

[8] Minwoo Im, Kyungsu Kang, and Heonyoung Yeom. 2022. Accelerating RocksDB for Small-Zone ZNS SSDs by Parallel I/O Mechanism. In *Proceedings of the 23rd International Middleware Conference Industrial Track* (Quebec, Quebec City, Canada) *(Middleware Industrial Track '22)*. Association for Computing Machinery, New York, NY, USA, 15–21.

[9] Jeeyoon Jung and Dongkun Shin. 2022. Lifetime-leveling LSM-tree compaction for ZNS SSD. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*. 100–105.

[10] 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*. 660–673.

[11] Miryeong Kwon, Donghyun Gouk, Changrim Lee, Byounggeun Kim, Jooyoung Hwang, and Myoungsoo Jung. 2020. DC-Store: Eliminating Noisy Neighbor Containers Using Deterministic I/O Performance and Resource Isolation *(FAST'20)*. USENIX Association, USA, 183–192.

[12] Hee-Rock Lee, Chang-Gyu Lee, Seungjin Lee, and Youngjae Kim. 2022. Compaction-Aware Zone Allocation for LSM Based Key-Value Store on ZNS SSDs. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems* (Virtual Event) *(HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 93–99.

[13] Youhuizi Li, Jiancheng Zhang, Congfeng Jiang, Jian Wan, and Zujie Ren. 2019. PINE: Optimizing Performance Isolation in Container Environments. *IEEE Access* 7, 30410–30422.

[14] Renping Liu, Zhenhua Tan, Yan Shen, Linbo Long, and Duo Liu. 2022. Fair-ZNS: Enhancing Fairness in ZNS SSDs through Self-balancing I/O Scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

[15] Parisa Jalili Marandi, Christos Gkantsidis, Flavio Junqueira, and Dushyanth Narayanan. 2016. Filo: Consolidated Consensus as a Cloud Service. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference* (Denver, CO, USA) *(USENIX ATC '16)*. USENIX Association, USA, 237–249.

[16] Sean McDaniel, Stephen Herbein, and Michela Taufer. 2015. A Two-Tiered Approach to I/O Quality of Service in Docker Containers. In *2015 IEEE International Conference on Cluster Computing*. 490–491.

[17] NVMe 2018. NVM Express 2.0 Zoned Namespace Command Set Specification. https://nvmexpress.org/specifications/

[18] Jennifer Ortiz, Brendan Lee, Magdalena Balazinska, Johannes Gehrke, and Joseph L. Hellerstein. 2018. SLAOrchestrator: Reducing the Cost of Performance SLAs for Cloud Data Analytics. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference* (Boston, MA, USA) *(USENIX ATC '18)*. USENIX Association, USA, 547–560.

[19] Ashton Webster, Ryan Eckenrod, and James Purtilo. 2018. Fast and service-preserving recovery from malware infections using CRIU. In *USENIX Security Symposium*. 1199–1211.

[20] Miguel G. Xavier, Israel C. De Oliveira, Fabio D. Rossi, Robson D. Dos Passos, Kassiano J. Matteussi, and Cesar A.F. De Rose. 2015. A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 253–260.

[21] Hanlin Zhang, Ningjiang Chen, Yusi Tang, and Birui Liang. 2019. Multi-Level Container Checkpoint Performance Optimization Strategy in SDDC. In *Proceedings of the 4th International Conference on Big Data and Computing* (Guangzhou, China) *(ICBDC '19)*. Association for Computing Machinery, New York, NY, USA, 253–259.