# Balloon-ZNS: Constructing High-Capacity and Low-Cost ZNS SSDs with Built-in Compression

Yu Wang*
Huazhong University of Science and Technology

Zibin Sun*
Huazhong University of Science and Technology

You Zhou†
Huazhong University of Science and Technology

Tao Lu
DapuStor Corporation

Changsheng Xie
Huazhong University of Science and Technology

Fei Wu†
Huazhong University of Science and Technology

## ABSTRACT

ZNS SSDs are emerging storage devices promising low cost, high performance, and software definability. This paper proposes Balloon-ZNS that enables transparent compression in ZNS SSDs to enhance cost efficiency. ZNS SSDs, unlike traditional SSDs, require data pages to be stored in logical zones and flash blocks with aligned offsets, conflicting with the management of compressed, variable-length pages. Motivated by a key observation that compressibility locality widely exists in data streams, Balloon-ZNS employs a compressibility-adaptive, slot-aligned storage management scheme to address the intractable conflict. Evaluation with RocksDB shows Balloon-ZNS can reap more than 80% of the compression gain while achieving 7.3% lower to 14.4% higher throughput than a vanilla ZNS SSD, on average, when data compressibility is not poor.

## 1 INTRODUCTION

Flash-based solid-state drives (SSDs) have emerged as the predominant choice for high-performance storage systems. However, SSDs still come at a much higher cost than conventional hard disk drives. Moreover, the intricate internal architecture and background activities of SSDs, such as address mapping and *garbage collection (GC)*, induce not only a high DRAM requirement for logical-to-physical page-level mapping table but also significant performance fluctuation and lifespan degradation. An SSD also reserves additional and user-invisible flash capacity, called *over-provisioning (OP)* space, to enhance GC efficiency and SSD reliability.

Zoned namespace (ZNS) is an emerging NVMe SSD interface that addresses the intricacies and high cost of SSDs. It is gaining increasing interest and support in Linux, SPDK, and RocksDB [8]. Unlike traditional block-interface SSDs allowing random writes/updates, ZNS SSDs divide the storage space into fixed-size zones (e.g., 2GB), where sequential writes are enforced and overwrites can only be conducted after the entire zone is reset. Each logical zone can be aligned and mapped to a group of flash blocks, called a *superblock*.

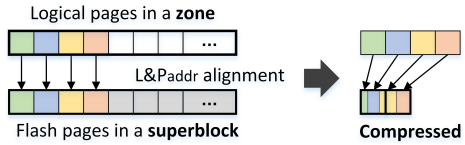


**Figure 1: Zone write constraint vs. management of compressed, variable-sized data pages in a ZNS SSD.**

The operational characteristics of writing/resetting zones are consistent with the physical write/erase constraints of flash superblocks. This simplifies SSD design and decreases the cost substantially [8]. As a coarse-grained zone-level mapping table is maintained, production ZNS SSDs employ 10× to 20× smaller DRAM than block-interface SSDs (which require 1GB DRAM per 1TB storage) [7]. Flash superblocks would be invalidated entirely when the relevant zones are reset, thus no valid data migrations are performed during GC. Hence, the performance predictability and lifespan of ZNS SSDs can be improved. The OP space, which accounts for 7% to 28% of the advertised storage capacity and increases SSD cost [8], can also be eliminated. Furthermore, the ZNS interface provides software-defined capabilities. Logical offsets of data pages in a zone are naturally coupled with their offsets in the superblock. The host can control data placement and I/O scheduling in logical zones and thus in flash memory in an application-aware manner.

In this paper, we aim to integrate transparent compression into ZNS SSDs (for the first time) to further enhance cost efficiency. The objective is inspired by three technical insights. First, compression stands as a widely adopted technique for reducing storage costs and, particularly for flash storage, extending device lifespan and improving write performance. Such benefits are highly desirable for the latest and future generations of high-density flash memory, which have degraded write speed and endurance. Second, prior research [17, 21] has highlighted the significant advantages of intra-SSD compression compared to host-side compression (e.g., at the application or file system level). These benefits include generality, transparency, and the avoidance of CPU overhead or the need for additional PCIe-based accelerators. Third, the integration of compression in SSDs aligns with the prevailing trend of offloading computation to storage for efficient near-data processing. It should be noted that computational SSDs with built-in compression capabilities have already been introduced [17].

However, unlike traditional block-interface SSDs, it is very challenging to enable compression in ZNS SSDs due to a fundamental

*Both authors contributed equally to this research.
†Corresponding authors: You Zhou and Fei Wu ({zhouyou2, wufei}@hust.edu.cn).


*DAC '24, June 23–27, 2024, San Francisco, CA, USA*

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3657368

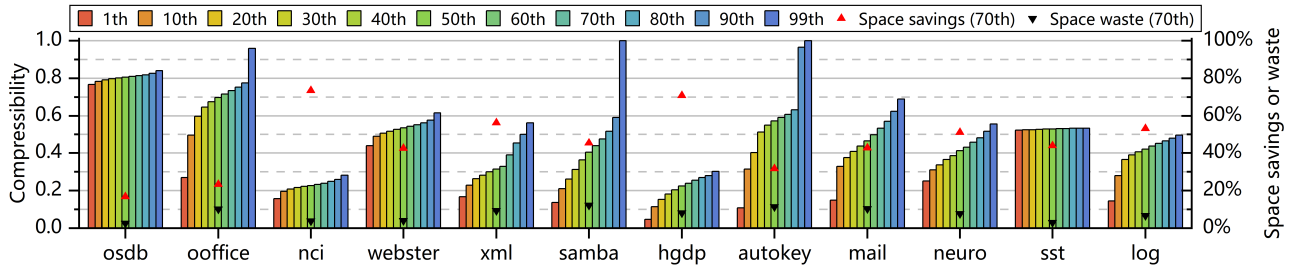


**Figure 2: Distribution of data page compressibility (the reciprocal of compression ratio) in real-world datasets. Small gaps between percentiles indicate strong compressibility locality. The space savings/waste denotes the storage space saved/wasted when compressed pages are stored in a slot-aligned way (slot size is set as 4KB divided by 70th-percentile compression ratio).**

conflict. The ZNS interface requires data pages to be stored in logical zones and relevant physical flash superblocks with aligned address offsets, which we refer to as *L&Paddr alignment*. As shown in Figure 1, compression changes fixed-size data pages to variable-length snippets compacted in flash pages. This breaks the L&Paddr alignment, and the zone-level mapping table in ZNS SSDs would be incapable of locating data pages. Maintaining a fine-grained page-level mapping table (like in block-interface SSDs) is not a feasible solution due to the excessively high DRAM cost for ZNS SSDs.

To address the intractable challenge, we conduct a comprehensive profiling study on the compressibility of real-world datasets (refer to §2). A crucial insight derived from this study is the widespread existence of *compressibility locality*, where a majority of data pages within a stream or file exhibit similar *compression ratios (CRs)*. Based on this observation, we propose *Balloon-ZNS*, which leverages compressibility locality through the implementation of a *compressibility-adaptive and slot-aligned storage management scheme* tailored for compressed, variable-length data pages.

When a zone is opened to be written, Balloon-ZNS maps it to flash sub-superblocks and profiles page compressibility of the data stream. Appropriate slot sizes are determined based on compressibility, and flash pages are divided into slots to accommodate compressed pages in an aligned manner. As some data pages may have poor compressibility and lead to larger footprints than the slots, Balloon-ZNS maintains the residues exceeding the slots separately and tracks their locations as housekeeping metadata in the slots. Evaluation with RocksDB shows that Balloon-ZNS can reap most of the compression gain while achieving high performance comparable to a vanilla ZNS SSD when data compressibility is not poor.

## 2 MOTIVATION

We have characterized the 4KB-page compressibility distribution in diverse real-world datasets and present some of the results in Figure 2 (space is limited). The datasets include the well-known Silesia compression corpus [1] and some other publicly available datasets [2–5]. They cover a wide spectrum of application scenarios, such as databases ('osdb' and 'nci'), code and library files ('ooffice' and 'samba'), XML, web, mail, text ('autokey'), and biological information ('hgdb' and 'neuro'). In particular, we also collected the SSTable and log files ('sst' and 'log') of RocksDB by running the dbbench benchmark with a moderate CR of two. The dataset size ranges from 2.2MB to 7.6GB. The compression algorithm in use is Deflate with static Huffman encoding.

***Observation 1: Failure to enable in-ZNS compression would hinder over 2× potential gains in TCO reduction.*** As illustrated in Figure 2, the page CRs of diverse datasets span from 1 to 20, indicating real-world data exhibits good compressibility. For example, the 50th percentile CRs across the datasets range from 1.2 to 4.5 with an average of 2.5.

***Observation 2: Most datasets show good compressibility locality.*** From Figure 2, it is evident that the majority of data pages within a stream or file tend to exhibit similar CRs, with small variations between different percentiles of CRs. Considering the L&Paddr alignment requirement of ZNS SSDs, we suppose a *slot-aligned storage layout* for compressed data pages. Flash pages can be divided into smaller fixed-size slots and one compressed page is stored in each slot. If the compressed page's footprint is smaller than the slot, some space is wasted. If the footprint is larger than the slot, the page is truncated and the residue is maintained out of place. The residues of truncated data pages can be compacted into a log on flash memory. In this case, two flash pages need to be read to restore a truncated data page. When the slot size is set so that 70% of the pages in a dataset can be entirely contained in slots after being compressed, the slot alignment causes 7.5% of storage space being wasted in the slots, on average across the datasets. Still, compression results in 16.8% to 73.4% of storage savings.

Applications and file systems commonly adopt the practice of allocating contiguous storage space for easy management and rapid access to the same data stream or file. The *Ext4* file system, for example, employs an extent tree to handle contiguous blocks of data, known as extents, effectively reducing fragmentation and enhancing overall file system performance. Allocating contiguous storage space is also consistent with the prospect that ZNS SSDs are favored because they allow the host to control data placement on the underlying flash storage, such as separating write streams of different files or applications to independent zones. With compressibility locality, contiguous data pages tend to have similar footprint sizes after compression.

***Motivation: Compressibility locality presents an opportunity to maintain the L&Paddr alignment and most of the compression gain in ZNS SSDs with a minor trade-off in read efficiency for some truncated data pages.***

## 3 DESIGN OF BALLOON-ZNS

In this section, we present Balloon-ZNS, a ZNS SSD with built-in data compression capability. Data pages are compressed online before being written to flash memory and are decompressed when

being read. As in conventional SSDs supporting in-device compression, we assume a hardware compression/decompression engine is integrated with the SSD controller of Balloon-ZNS. Typically, the engine can compress/decompress a data page in a few microseconds, which would not become a performance bottleneck.

The key challenge is to maintain the L&Paddr alignment for compressed, variable-length data pages, required by ZNS SSDs. Balloon-ZNS resolves it by devising a *compressibility-adaptive and slot-aligned storage layout*. We present the complete storage management in §3.1 and discuss some concerns and software-defined potentials in §3.2.

## 3.1 Storage Management in Balloon-ZNS

For ZNS SSDs, when a zone is to be written, it is opened first and the SSD maps it to a flash superblock, i.e., a GC unit. Generally, a superblock consists of flash blocks with the same offset across many or all the parallel flash chips/dies. Then, data pages are written to flash pages in a sequential and aligned manner. As shown in Figure 3, Balloon-ZNS enables data compression and realizes several techniques for efficient storage management, including zone-to-flash mapping, slot-aligned data layout, compressibility-adaptive slotting, and residue space reclamation.

***Zone-to-Flash Mapping.*** With compression, the storage footprint of a zone is reduced and becomes changeable depending on the data compressibility. Balloon-ZNS partitions flash superblocks into smaller sub-superblocks, which is the basic unit for zone mapping. A zone is mapped to one sub-superblock at the first write and more sub-superblocks could be allocated on demand when the current sub-superblock still cannot satisfy the space requirement. As the sub-superblock size is large (typically hundreds of MBs), the zone-level mapping table is small (i.e., tens of KBs per TB storage). This table can be cached entirely in SSD-internal RAM and protected against sudden power loss by super-capacitors.

***Slot-Aligned Data Layout.*** To retain the L&Paddr alignment, Balloon-ZNS divides contiguous flash pages into smaller slots. The slot size is configured according to the data compressibility (see the subsequent Compressibility-Adaptive Slotting technique). For example, if the slot size is 1/3 or 2/3 of the page size, one or two flash pages are partitioned into three slots, respectively. Each compressed data page fits into and aligns with one slot. If a compressed page's footprint is smaller than the slot, some space is wasted in the slot. If the footprint is larger than the slot, the page is truncated and the residue exceeding the slot is maintained separately in a log (e.g., data page `C` in Figure 3).

We refer to the sub-superblocks being slotted as *home* sub-superblocks and the sub-superblocks used to log the residues of overlarge pages as *extra* sub-superblocks. Home sub-superblocks are mapped exclusively to each zone, while extra sub-superblocks can be shared by multiple zones. An extra sub-superblock is owned and can only be written by one open zone until it is closed. Then, the extra sub-superblock can be assigned to another newly opened zone for appending its residues. By doing so, the residues of each zone are recorded in continuous physical space, i.e., a consecutive and independent set of flash pages.

A truncated data page is stored in two separate flash pages. The first flash page containing the leading part can be located through the zone-level mapping table. Assuming a flash page is divided into

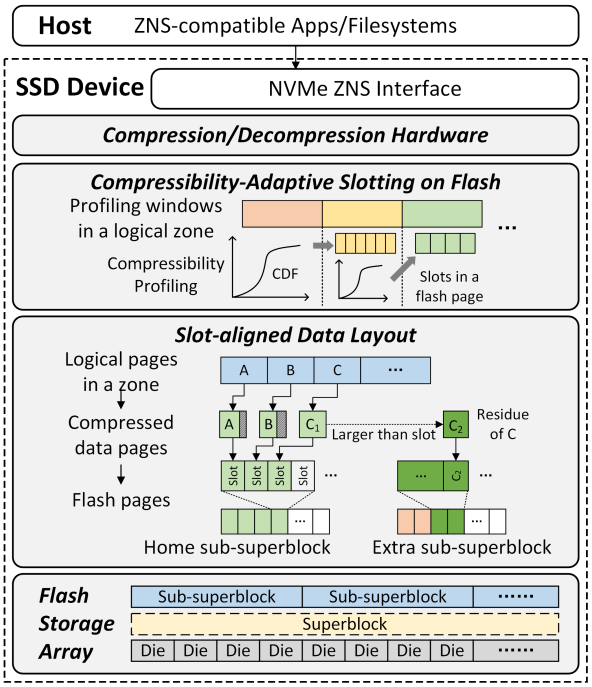


**Figure 3: Overview of Balloon-ZNS.**

`N` slots, a truncated data page whose logical address is `L` in Zone `0` can be found in the `(L%N)`*th* slot of the `(L/N)`*th* physical page among the home sub-superblock(s) mapped to Zone `0`. To locate the residue residing in an extra sub-superblock, Balloon-ZNS embeds its physical address and length, as housekeeping metadata, into the leading part when the data page is written. Hence, a truncated data page can be obtained by reading the two flash pages in sequence. Although the two flash pages could be read in parallel if standalone page-level mapping metadata is maintained for tracking the residues, this method is not considered due to an excessive RAM requirement for ZNS SSDs.

***Compressibility-Adaptive Slotting.*** The slot size should be adaptive to data page compressibility for high storage efficiency. Balloon-ZNS performs compressibility profiling to determine an appropriate slot size. Note that different ZNS SSDs may have various zone size configurations and the zone size can be large (e.g., several GBs). We introduce an independent granularity, called *profiling window*. A logical zone is partitioned into multiple fixed-size profiling windows. As a zone can only be written sequentially, the windows are written successively. Balloon-ZNS configures a specific slot size for each window according to the compressibility profiling result of the last window, assuming compressibility locality exists in the subsequent data pages.

Specifically, Balloon-ZNS records the size of each compressed data page when writing it to the zone, and maintains the CR distribution of data pages in the current profiling window (through a simple bucket sorting algorithm). When the next window is to be written, the relevant flash pages are slotted with a size, determined based on the value of a certain percentile in the CR distribution of the last window. In addition, to realize adaptive-sized slotting, the zone-level mapping table is enhanced to maintain the starting flash page address of each profiling window.

The profiling window size, the percentile of CR value, and the slot size for the initial window in a zone are configurable parameters. In the current implementation, they are set as 1/8 of the zone size, 70th percentile, and 1/2 of the page size, respectively, by default. To simplify storage management, slot sizes and the residues are aligned with one or multiple 256B units.

***Residue Space Reclamation.*** When the host needs to update a logical zone, it is reset and all its data pages become invalid first. In conventional ZNS SSDs, the corresponding flash superblocks would be invalidated entirely and erased, during which no GC (i.e., migrations of valid data on flash memory) is required. By contrast, Balloon-ZNS employs extra sub-superblocks to keep the residues of truncated data pages from different zones. When some of the zones are reset, their home sub-superblocks are invalidated entirely. However, extra sub-superblocks could contain a mix of valid and invalid flash pages, necessitating residue space reclamation.

When free space runs short and an extra sub-superblock is reclaimed, valid residues in it are relocated to another sub-superblock. The residues' addresses have already been recorded as housekeeping metadata with the leading parts of truncated data pages and cannot be updated on flash memory. Note that the residues of each zone are stored in a set of continuous flash pages. To avoid data inconsistency, Balloon-ZNS keeps the storage offsets and lengths of the residues, rather than flash addresses, in the housekeeping metadata. Meanwhile, the starting flash page's address is maintained for each zone in the zone-level mapping table and can be updated when the relevant sub-superblock is reclaimed.

As the footprint of residues and thus the number of extra sub-superblocks are very small, residue space reclamation rarely happens and the overhead is negligible. In addition, compressed data pages of a zone may occupy only part of the home sub-superblock mapped to the zone. To improve storage efficiency, Balloon-ZNS migrates the residues of a zone from the extra sub-superblock to the zone's partially filled home sub-superblock if it has sufficient free space when the logical zone is fully written.

## 3.2 Discussion

***Capacity Management.*** With built-in compression, an SSD is able to expose a larger logical address space to the host than the physical capacity. To this end, Balloon-ZNS provides an NVMe management command to allow the user to specify a capacity expansion ratio. For example, when the expansion ratio is set as 2, a Balloon-ZNS SSD with 2TB raw capacity would present 4TB user-visible capacity. If the expansion ratio is higher than the actual compression ratio of user data, the SSD could run out of flash storage before the logical address space exhausts. In this case, Balloon-ZNS would report an out-of-space error to the operating system and require the user to intervene, similar to the behavior of existing block devices.

***Incompressible Data Streams.*** Balloon-ZNS uses flash sub-superblocks to store compressed data, while superblocks can provide full parallelism and thus higher read/write performance. To avoid unnecessary performance degradation, Balloon-ZNS can set a low watermark of CR for compressibility profiling. If data CR is lower than the watermark, the zone is regarded as incompressible and mapped to a superblock as in conventional ZNS SSDs.

***Software Definability.*** The compression ability of Balloon-ZNS can be utilized in a transparent way by the host. Alternatively, it can also be utilized as a useful feature and in a software-defined manner for high efficiency. The host can manage write streams by being aware of data compressibility. Writing data pages with similar compressibility to the same zone can minimize the space waste caused by slot alignment and the overhead of reading truncated data pages. Moreover, the host can explicitly disable the compression for a zone (e.g., through a hinted zone open command) if the data stream has poor compressibility or is read-critical, and map the zone to a flash superblock. While this paper demonstrates the feasibility of enabling compression in ZNS SSDs and the design of Balloon-ZNS, we leave it as future work to exploit the software definability and hardware/software co-designs for ZNS-compatible applications.

**Table 1: Characteristics of Six YCSB workloads**

| | | | |
|---|---|---|---|
| A | 50% read, 50% update | B | 95% read, 5% update |
| C | 100% read | D | 95% read latest, 5% update |
| E | 95% scan, 5% update | F | 50% read, 50% read-modify-write |

**Table 2: Real-world datasets used in RocksDB evaluation**

| Label | Dataset | Label | Dataset | Label | Dataset |
|---|---|---|---|---|---|
| **h1** | nci | **m1** | webster | **l1** | osdb |
| **h2** | hgdp | **m2** | xml | **l2** | ooffice |

* *'**h**', '**m**', or '**l**' means the dataset has a relatively high, medium, or low CR. The '**1**'-series datasets have stronger compressibility locality than the '**2**'-series ones.*

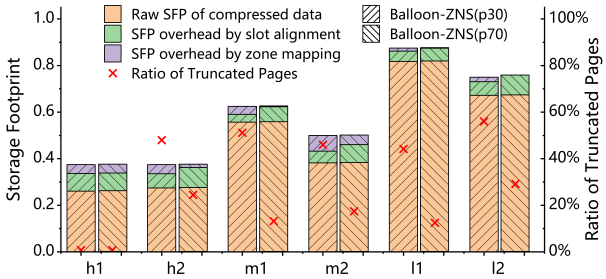


**Figure 4: Storage footprint (SFP) and ratio of truncated pages in Balloon-ZNS under RocksDB with six datasets. The slot size is set as the 30th/70th percentile (p30/p70) in CR profiling. The SFPs are normalized to the SFP of uncompressed data.**

# 4 EVALUATION

## 4.1 Experimental Setup

We implement Balloon-ZNS on FEMU [13], a widely used NVMe SSD emulator supporting full-system research. The ZNS SSD has 256GB storage capacity and contains 8 channels each connecting 8 flash dies. Each die has 128 flash blocks, and each block includes 8,192 pages of 4KB size. A superblock consists of 64 blocks spreading across all the dies. Both the zone size and superblock size are 2GB. The sub-superblock size is configured as 1/8 of the superblock.

Flash read, write, and erase latencies are set as 121$\mu$s, 1.86ms, and 6.34ms, respectively (based on a QLC flash chip [9]). We integrate a real hardware card, Intel QAT 8970, with FEMU to emulate the built-in compressor of Balloon-ZNS. The compression and decompression latencies of a page with QAT are around 25$\mu$s and 15$\mu$s, respectively. In contrast, ASIC-based compression engines in modern SSDs can compress/decompress a 4KB page in about 5$\mu$s.

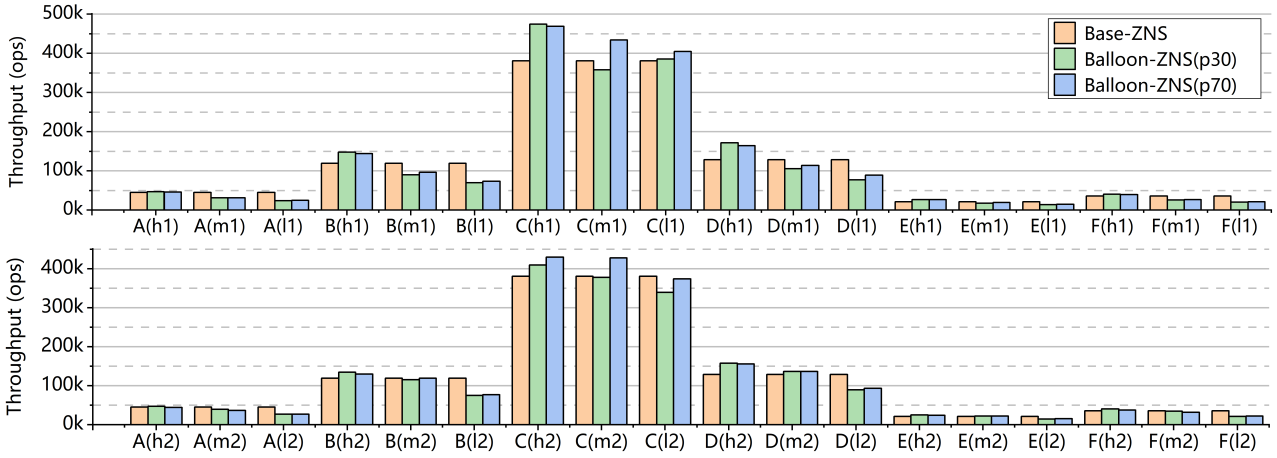


**Figure 5: RocksDB throughputs in six YCSB I/O workloads (A, B, C, D, E, F) and six datasets with various data CRs and compressibility locality. The slot size in Balloon-ZNS is set as the 30th or 70th percentile (p30 or p70) in CR profiling.**

Balloon-ZNS is compared to a traditional ZNS SSD, named *Base-ZNS*, which maps zones to superblocks and does not perform compression. We run extensive experiments on a high-performance key-value store RocksDB, the most popular application on ZNS SSDs. The benchmark is YCSB [20], whose workload characteristics are shown in Table 1. YCSB workloads use the default concurrency configuration, i.e., 64 threads, and contain 60 million key-value pairs of 1KB size. We also use the fio[6] tool to generate synthetic block I/O workloads. As the workloads generate patterned I/Os, we incorporate the diverse data page CR distributions of six real-world datasets (see Figure 2 and Table 2) to configure the compressibility of data streams written on flash memory.

## 4.2 Storage Efficiency Results

Balloon-ZNS enables in-device compression to improve storage efficiency. Figure 4 shows the *storage footprint (SFP)* sizes of Balloon-ZNS in RocksDB evaluation with six datasets with various compressibility. Two slotting configurations are considered: *p30* and *p70* denote that the slot size is set as the 30th and 70th percentiles of data page CRs in the profiling window, respectively. We can see compression leads to raw SFP reductions of 73.9%, 72.6%, 44.3%, 61.8%, 18.2%, and 32.8% in the six datasets, *h1*, *h2*, *m1*, *m2*, *l1*, and *l2*, respectively, compared with the SFPs of uncompressed data. The actual SFP reductions achieved by Balloon-ZNS(p70) are 62.5%, 62.5%, 37.5%, 50%, 12.5%, and 24.2%. That is, such reductions account for 68.6% to 86.1% of the raw compression gains across all the datasets, while more than 80% in medium- and high-CR datasets.

The SFP overheads originate from slot alignment and zone-to-sub-superblock mapping. It is noted that p30 and p70 induce similar SFP sizes in all the datasets. p70 has more space waste in slots and a smaller number of truncated data pages, while p30 causes more space unused in sub-superblocks. Specifically, in p70, the ratio of truncated pages ranges from 1% to 29% and the footprint of the residues takes no larger than 6.5% of the total SFP of compressed data. As a result of fewer truncated pages, p70 achieves 4.9% and 13.2% higher throughput than p30 in the *h2* and *m2* datasets, respectively, under read-only YCSB C workload (see Figure 5).

***Conclusion 1: Balloon-ZNS can reap most (68.6% to 86.1%) of the compress gain in storage efficiency. In terms of the slot sizing configuration, p70 is preferable to p30 as similar storage efficiency but higher read performance can be obtained.***

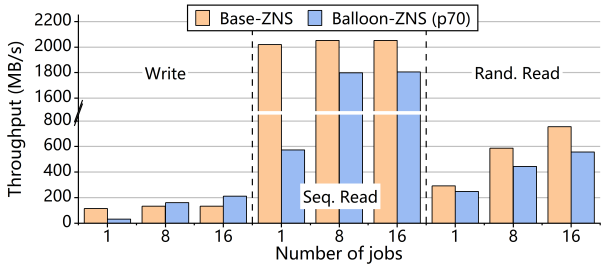


**Figure 6: Throughputs in sequential writes and sequential/random reads (fio workloads) with the *m2* dataset.**

## 4.3 Performance Results

Figure 5 exhibits the RocksDB throughputs under six YCSB I/O workloads and different datasets. The average RocksDB throughput of Balloon-ZNS(p70) is 14.4% higher, 7.3% lower, and 29.3% lower than that of Base-ZNS, respectively, when the data CR is high, medium, and low. The maximum throughput improvements are 28.0% and 14.1% in the high-CR and medium-CR datasets, respectively. The maximum throughput decreases in the medium-CR and low-CR datasets are 30.9% and 45.9%. The performance of Balloon-ZNS is proportional to the data CR, and the performance improvement stems from reduced flash reads and writes since data pages are compressed and compacted.

Enabling compression degrades the flash parallelism in each zone and causes two serialized flash reads for each truncated data page. The degradations can be compensated by concurrent access to multiple zones, which is common in real-world applications and multi-tenant scenarios. For example, Balloon-ZNS(p70) outperforms Base-ZNS by up to 23.2% in the YCSB C workload (100% read), which issues concurrent reads with an average request size larger than 8KB. This read pattern results in good utilization of flash parallelism and compacted data layout.

Figure 6 exhibits the raw device throughputs under sequential writes, sequential reads, and 4KB random reads with medium data compressibility. When only one zone is sequentially written or

read with one job, Balloon-ZNS(p70) has 71.3% or 71.6% lower throughput than Base-ZNS, respectively. This is because Balloon-ZNS maps zones to sub-superblocks, which spread across 8 parallel flash dies (vs. 64 for superblocks in Base-ZNS). For datasets with low I/O concurrency but high I/O performance sensitivity, Balloon-ZNS can disable the compression and map the relevant zones to superblocks as in Base-ZNS. When multiple zones are accessed simultaneously by 8 or 16 jobs, Balloon-ZNS achieves 20.1% or 57.5% higher write throughput and only 12.3% or 12.0% sequential read throughput degradation, respectively. Under multi-job 4KB random reads, Balloon-ZNS has 24.4% or 26.1% lower random read throughput than Base-ZNS because 4KB small reads cause non-trivial flash read amplification.

***Conclusion 2: Despite a lower access speed per zone, Balloon-ZNS demonstrates comparable or superior performance to conventional ZNS SSDs in real-world applications such as RocksDB when the compressibility of the data is not poor.***

## 5 RELATED WORK

ZNS SSDs, standardized by NVMe in 2020, have attracted much attention. Bjørling et al. [8] characterized a real ZNS SSD prototype in RocksDB and F2FS application scenarios, revealing its significant benefits over block interface SSDs in terms of throughput and tail latency. Applications running on ZNS SSDs are responsible for zone GC. To reduce the overhead of zone GC, ZNS + [10] offloads data migrations to the SSD for F2FS, while LIZA [12] separates hot and cold data into different zones in RocksDB. To enhance software-defined capability, eZNS [15] realizes an elastic ZNS interface over commodity ZNS SSDs with small-size zones, and FlexZNS [19] presents a flexible ZNS SSD supporting size-configurable zones. In addition, RAIZN [11] constructs a ZNS SSD RAID for reliability and performance enhancements. Our work integrates compression into ZNS SSDs, which is complementary to these works.

Many works from both academia and industry have advocated enabling transparent compression inside SSDs. Zuck et al. [21] identified storage management inefficiencies of introducing compression into applications or file systems. zFTL [16] incorporates data compression as well as a compressibility predictor into the SSD controller. Besides general compression gains in reducing storage footprint and flash writes, Li et al. [14] and Qiao et al. [17] explored the potentials of in-SSD compression for reducing flash wear and improving host-side data management efficiency, respectively. Moreover, computational SSDs with compression (e.g., from Seagate [18] and ScaleFlux [17]) have emerged in the market. These works target block-interface SSDs, while our Balloon-ZNS enables transparent compression in the latest ZNS SSDs for the first time.

## 6 CONCLUSION

This paper demonstrates that it is feasible, although challenging, to enable transparent compression in ZNS SSDs to improve cost efficiency while obtaining decent performance. The key is to exploit compressibility locality in data streams and employ a compressibility-adaptive, slot-aligned storage management scheme. We believe Balloon-ZNS can become a cornerstone for building large-capacity, low-cost, high-performance, and software-defined storage systems.

## ACKNOWLEDGMENTS

We sincerely thank the reviewers for their valuable suggestions. This work is supported by the National Natural Science Foundation of China (under Grants 62372197, U2001203, U22A2071, 62102155), and the Natural Science Foundation of Shandong Province (under Grant ZR2020LZH014).

## REFERENCES

[1] 2003. Silesia compression corpus. https://sun.aei.polsl.pl/~sdeor/index.php?page=silesia.
[2] 2015. Enron Email Dataset. https://www.cs.cmu.edu/~enron/.
[3] 2016. NeuroElectro: organizing information on cellular neurophysiology. https://neuroelectro.org/.
[4] 2018. Datasets for automatic keyphrase extraction task. https://github.com/snkim/AutomaticKeyphraseExtraction/.
[5] 2021. Human Genome Diversity Project. https://www.internationalgenome.org/data-portal/data-collection/hgdp.
[6] Jens Axboe. 2005. Flexible I/O Tester. https://github.com/axboe/fio
[7] Jiseon Bae, Hanyeoreum an 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.*
[8] 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 SSD. In *Proceedings of the USENIX Annual Technical Conference (ATC'21).*
[9] Qihui Chen, Shuai Wang, You Zhou, Fei Wu, Shu Li, Zhengyong Wang, and Changsheng Xie. 2022. PACA: A Page Type Aware Read Cache Scheme in QLC Flash-based SSDs. In *Proceedings of the IEEE 40th International Conference on Computer Design (ICCD'22).*
[10] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. 2021. ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21).*
[11] 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 (ASPLOS'23).*
[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.*
[13] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan 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).*
[14] Jiangpeng Li, Kai Zhao, Xuebin Zhang, Jun Ma, Ming Zhao, and Tong Zhang. 2015. How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime?. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15).*
[15] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. 2023. eZNS: An Elastic Zoned Namespace for Commodity ZNS SSDs. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23).*
[16] Youngjo Park and Jin-Soo Kim. 2011. zFTL: Power-efficient data compression support for NAND flash-based consumer electronics devices. *IEEE transactions on consumer electronics* 57, 3 (2011), 1148–1156.
[17] Yifan Qiao, Xubin Chen, Ning Zheng, Jiangpeng Li, Yang Liu, and Tong Zhang. 2022. Closing the B+-tree vs. LSM-tree Write Amplification Gap on Modern Storage Hardware with Built-in Transparent Compression. In *Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST'22).*
[18] Seagate. 2022. Nytro 1000 SSD Series Datasheet). https://www.seagate.com/www-content/datasheets/pdfs/nytro-1351-1551-sata-ssdDS1992-4-1907US-en_US.pdf. Online; accessed on September 1, 2022.
[19] Yu Wang, You Zhou, Zhonghai Lu, Xiaoyi Zhang, Kun Wang, Feng Zhu, Shu Li, Changsheng Xie, and Fei Wu. 2023. FlexZNS: Building High-Performance ZNS SSDs with Size-Flexible and Parity-Protected Zones. In *Proceedings of the 41st IEEE International Conference on Computer Design (ICCD'23).*
[20] Yahoo. 2010. Yahoo! Cloud Serving Benchmark (YCSB). https://github.com/brianfrankcooper/YCSB.
[21] Aviad Zuck, Sivan Toledo, Dmitry Sotnikov, and Danny Harnik. 2014. Compression and SSDs: Where and How?. In *Proceedings of the 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW'14).*