2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00148

ZonedStore: A Concurrent ZNS-Aware Cache System for Cloud Data Storage

Yanqi Lv[1], Peiquan Jin[1], Xiaoliang Wang[1], Ruicheng Liu[1], Liming Fang[2], Yuanjin Lin[2], Kuankuan Guo[2]
[1]*University of Science and Technology of China, Hefei, China*
[2]*ByteDance Inc., Beijing, China*

***Abstract*—Cloud data storage relies on efficient cache systems to offer high performance for intensive reads/writes on big data. Due to the large data volume of cloud data storage and the limited capacity of DRAM, current cloud vendors prefer to use SSDs (Solid State Drives) but not DRAM to build the cache system. However, traditional SSDs have a serious over-provisioning problem and a high cost in garbage collection. Thus, the performance of SSD-based cache systems will drop quickly when the usage of SSDs increases. Recently, Zoned Namespaces (ZNS) SSDs have emerged as a hot topic in both academics and industries. Compared to conventional SSDs, ZNS SSDs have the advantages of less overhead of garbage collection and lower over-provisioning costs. Therefore, ZNS SSDs have been a better candidate for the cache system for cloud storage. However, ZNS SSDs only accept sequential writes, and the zones inside ZNS SSDs need to be carefully managed to maximize the advantages of ZNS SSDs. Therefore, making the cache system adapt to ZNS SSDs is becoming a challenging issue. In this paper, we demonstrate ZonedStore, a novel ZNS-aware cache system for cloud data storage. After a brief introduction to the architecture of ZonedStore, we present the key designs of ZonedStore, including a Zone Manager to control the space allocation and operations on ZNS SSDs, a Multi-Layer Buffer Manager, and an In-Memory Concurrent Index to accelerate accesses. Finally, we present a case study to demonstrate the working process and performance of ZonedStore.**

***Index Terms*—Zoned namespaces, Solid-state drive, Cloud storage, Cache system**

I. Introduction

Cloud data storage has been one of the main data-storage solutions in distributed computing environments. However, due to the big data storage of cloud data, cloud storage vendors need to build a high-performance cache system to accelerate data access on cloud storage. Although DRAM has been widely used in operating systems or DBMSs as the cache media, it is not suitable for the cache for cloud data storage because of its limited capacity. Therefore, current cloud storage prefers to use flash-memory-based SSDs (Solid State Drives) as the cache.

However, many experiences with conventional SSDs have shown that SSDs will incur severe throughput drops when their usage is over 50% [1]–[3], which is caused by the internal garbage collection processes of SSDs. In addition, conventional SSDs have a high over-provisioning cost, meaning that they have to remain a few spaces to undergo the out-of-place updates to SSDs. This is because that flash memory does not support in-place updates. When an update happens, we have to write the updated data to a new page and mark the old page unavailable. Therefore, some spaces must be remained for storing the updated data. Meanwhile, garbage collection operations will remove the unavailable pages caused by updates. As a garbage collection will use costly erasure operations [4], frequently garbage collections will worsen the performance of SSDs. Unfortunately, when the usage of an SSD is over 50%, garbage collections will be frequently invoked, leading to the performance drop of the SSD.

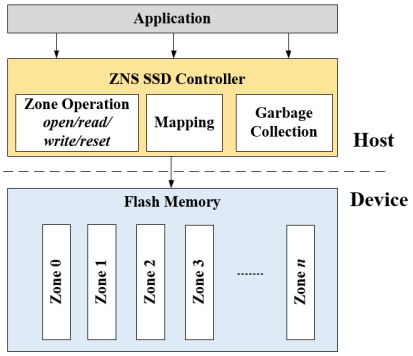

Fig. 1. The architecture of ZNS SSDs

Recently, a new type of SSDs called ZNS SSDs (Zoned Namespaces Solid-State Drives) [1] has come to the big data research community [2], [5]–[8]. The most critical technology in ZNS SSDs is the zone-based storage architecture, as shown in Fig. 1. A ZNS SSD groups its storage capacity into zones, and each zone can be read in any order but must be written sequentially. In addition, ZNS SSDs move most FTL(Flash Translation Layer) [9] functionalities into the host, enabling developers to implement effective algorithms to control the zones in ZNS SSDs. According to some early experiments [1], [2], ZNS SSDs exhibit much higher and stabler performance than conventional SSDs. For example, Western Digital reported that 60 percent lower average read latency and three times higher write throughput were achieved in testing on ZNS SSDs prototypes [3]. Other researchers also reported that the write amplification of RocksDB dropped from 5 times to 1.2 times on ZNS SSDs [10].

In this paper, we present a novel ZNS-SSD-optimized cache system for cloud data storage, which is named ZonedStore. The most promising feature of ZonedStore is that it is compatible to and optimized for ZNS SSDs, meaning that it can run on ZNS SSDs directly and efficiently. To the best of

2575-8411/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDCS54860.2022.00148

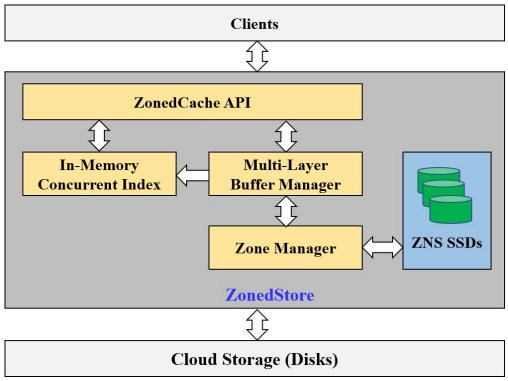

Fig. 2. The architecture of ZonedStore

our knowledge, ZonedStore is the first cache system that can adapt to ZNS SSDs. Briefly, ZonedStore has the following new features:

- We propose a Zone Manager in ZonedStore to manage the zones effectively and efficiently. The zoned address space gives a chance to decrease write amplification by allocating data into separate zones and reducing the usage of the over-provisioning area in SSDs. However, the Zone Manager has to handle a zone explicitly and ensure that all data in a zone is written sequentially.
- We present a new Multi-Layer Buffer Manager in ZonedStore to cache hot data from cloud data storage efficiently. The buffer is composed of three layers, including a write layer consisting of many write buffers for clients, an immutable layer composed of immutable buffers, and a flush layer with lock-free flush buffers.
- We design an efficient In-Memory Concurrent Index to accelerate data operations. The index is built with concurrent hashmaps and can support concurrent read/write operations on the cached data.

II. Architecture and Key Designs of ZonedStore

Figure 2 shows the architecture of ZonedStore. It mainly includes four components. The Zone Manager is responsible for space management and operations of ZNS SSDs. The Multi-Layer Buffer Manager is used as the write buffer for the writes from clients. The In-Memory Concurrent Index maintains the index for the buffered data to accelerate query operations. The ZonedStore API is the interface provided for clients to read and write data.

A. Zone Manager

The Zone Manager is responsible for the management of ZNS SSDs. A few tasks are processed by the Zone Manager, which are summarized as follows.

(1) It provides a page abstraction for the Multi-Layer Buffer Manager so that the buffer manager can process data using page-based interfaces.

(2) It maintains zone-related metadata, e.g., the write pointer, the garbage ratio, and the space usage of each zone.

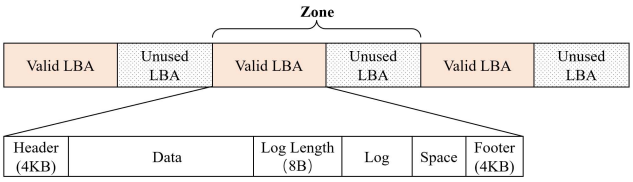

Fig. 3. The structure of the zones' space

The metadata will be used by other tasks, such as data flush, zone selection and garbage collection.

(3) It manages the data allocation and garbage collection inside zones. When a flush buffer is flushed by the buffer manager, the Zone Manager will select an appropriate zone to store the data based on the metadata maintained for each zone. The garbage collection is performed at a zone granularity, in which we select the zone with the highest garbage ratio to reset.

(4) It can simulate a conventional SSD as a ZNS SSD, making it applicable to conventional SSDs. Such a design enables ZonedStore compatible with traditional SSD-based cache systems.

The data in ZNS SSDs are organized using a key-value separation strategy [11], [12]. In general, we suppose that the key size is much larger than the value size. The address space of each zone is organized as a set of LBA (Logical Block Address), as shown in Fig. 3. Note that the logs for zones record the key information, which can benefit garbage collection operations and quick recovery. To be more specific, when performing a garbage collection operation, we need not read all the data within the zone to be recycled but only need to read the key information from the log region and the necessary data to be migrated. When recovering, we only need to read logs and ignore all data. The size of logs is far less than that of data.

B. Multi-Layer Buffer Manager

Figure 4 shows the multi-layer buffer structure. Each layer is composed of multiple small buffers. The novel design of such a multi-layer and multi-buffer structure can be summarized into two aspects. First, we allocate multiple small buffers for each layer to enable the buffer manager with the concurrent processing functionality, which is especially important for cloud environments that involve a great number of clients. Second, we use multi-layer buffers to offer a high write throughput for big write streams. When a write request arrives, the new record will be put into the write buffer and a response will be returned to the client simultaneously.[1] When the write buffers become full, we migrate the data in the write buffers to the immutable buffers. The immutable buffers are not allowed to append data. When the immutable buffers complete serialization, they will be transformed into flush buffers. Note that the key point for the immutable buffers and the flush buffers is that all the operations to these two buffers use a

[1]Currently, the WAL logging is not enabled in ZonedStore, but it is easy to be added.

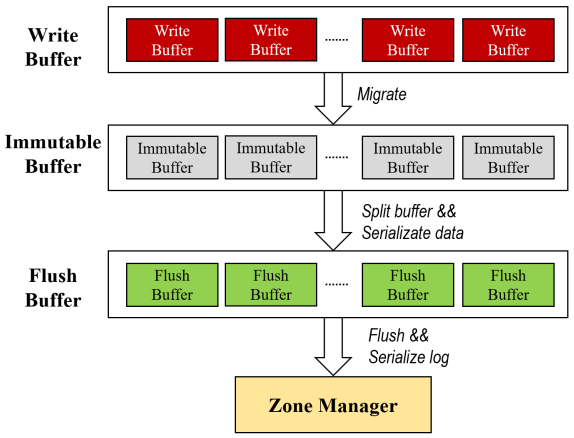

Fig. 4. The multi-layer buffer structure

lock-free multiple producer/consumer queue [13]. Therefore, ZonedStore can maintain stable and high performance on write-intensive workloads.

Note that the data in the flush buffers will be flushed to ZNS SSDs via the Zone Manager, but the logs are still in memory. We flush logs until the zone is not able to accommodate more data. Such a mechanism ensures that all writes to ZNS SSDs are sequential writes. To this end, ZonedStore is ZNS friendly because it will not incur random writes to ZNS SSDs.

C. In-Memory Concurrent Index

The In-Memory Concurrent Index is used to accelerate query performance. To make the index support concurrent requests efficiently, we implement a concurrent hash index in ZonedStore, which can support concurrent point queries efficiently. Note that the index entries are created by the Multi-Layer Buffer Manager, as shown in Fig. 2. When the buffer manager reads valid data from the Zone Manager during garbage collection, it will update the entries in the index.

D. ZonedStore API

The APIs provided by ZonedStore include `Get`, `Put`, and `Delete`. The `Get` operation first checks the in-memory index to locate the buffered data. If the data is found in the buffer, the buffer manager will return a copy of the data to the client directly. If the data is not in the buffer, the buffer manager will request the Zone Manager to read the data from ZNS SSDs. For the `Put` operation, ZonedStore will first write the new data to a write buffer (a lock to the write buffer is required). Then, the new index entry is inserted into the index. If the write buffer becomes full after the new writing, it will be migrated into an immutable buffer. For the `Delete` operation, we simply remove the index entry of the key to be deleted from the in-memory index. To this end, the delete operations in ZonedStore are actually logical deletions. The physical space of deleted data will be reclaimed during the next garbage collection in the corresponding zone.

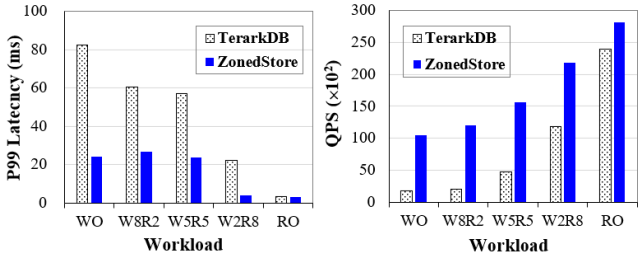

Fig. 5. Comparison of ZonedStore with TerarkDB on conventional SSDs

TABLE I
PERFORMANCE OF ZONEDSTORE ON ZNS SSDS.

Cache Size	P50(ms)	P99(ms)	QPS	Throughput(GB/s)
450 GB	1.1749	21.8444	17719	1.6502
600 GB	1.1822	21.9372	17733	1.6515
750 GB	1.1789	21.7754	17736	1.6518

III. PERFORMANCE EVALUATION

The ZonedStore system has been implemented and deployed in ByteDance's cloud storage. Currently, there is no cache system that can support ZNS SSDs, so we first evaluate the performance of ZonedStore on a conventional SSD and compare it with the existing SSD-based cache system deployed in ByteDance, which is called TerarkDB [14], [15]. Then, we report the performance of ZonedStore on real ZNS SSDs provided by Western Digital.

In all experiments, the key size is set to 32 bytes, and the value size is set to 100 KB to avoid the impact of the time difference in reading data with different sizes. We set the cache capacity to 600 GB. In addition, we insert six million key-value pairs to fill the cache and then run four million requests. To make the comparison fair, we close the WAL logging of TerarkDB and use the `DIRECT_IO` mode to remove the influence of the page cache in the operating system.

The workloads include five types, namely **WO** (write-only), **W8R2** (80% writes and 20% reads), **W5R5** (50% writes and 50% reads), **W2R8** (20% writes and 80% reads), and **RO** (read-only). We mainly focus on the comparison of latency and throughput.

Figure 5 shows the latency and throughput comparison between ZonedStore and TerarkDB when running on the conventional SSD. We can see that ZonedStore outperforms TerarkDB in both latency and throughput.

Table I shows the performance of ZonedStore on real ZNS SSDs when the cache size varies from 450 GB to 750 GB. In this experiment, we first insert enough random keys to fill the cache. Then we run three million requests on the 450 GB cache, four million requests on the 600 GB cache, and five million requests on the 750 GB cache. All the workloads consist of 50% writes and 50% reads. We can see that ZonedStore can always maintain stable latency and QPS under various cache sizes.

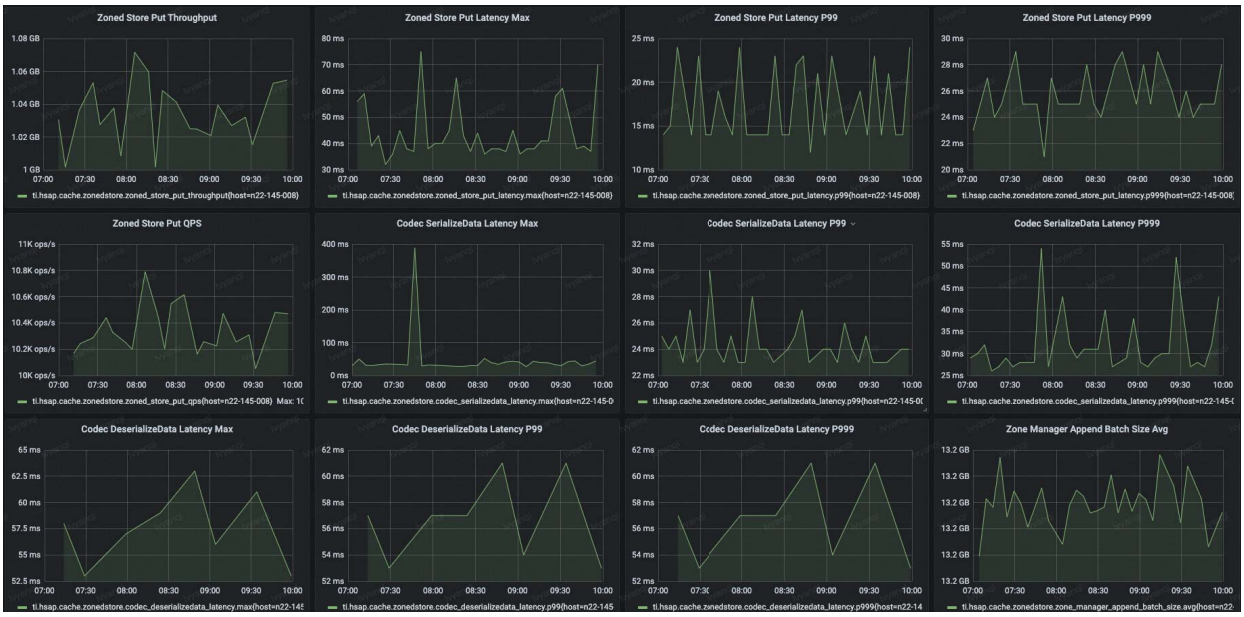

Fig. 6. Performance monitoring of ZonedStore

IV. Demonstration

In the demonstration, we will run ZonedStore on both conventional SSDs and real ZNS SSDs. The demonstration includes the following steps:

(1) We first configure the environment of the cloud storage, TerarkDB, and ZonedStore. Then, we load four million key-value pairs to TerarkDB and ZonedStore to initialize the cache system.

(2) We run db_bench to issue different workloads. Each workload involves four million requests focusing on the key-value pairs written at the first step. In the demonstration, we will only include `Put` and `Get` operations.

(3) The performance of ZonedStore is shown on a dashboard that is built on top of an open-source visualization tool GrafanaA few metrics are visualized on the dashboard, through which users can know the real-time performance of ZonedStore.

Figure 6 shows a screenshot of the online dashboard when running ZonedStore. All the metrics shown on the dashboard are real-time values calculated by ZonedStore. We visualize several metrics on the dashboard, including the latency, QPS (queries per second), and throughput (megabytes per second). Also, we monitor the average batch size of the Zone Manager in the demonstration. Note that the running and visualization can be paused and resumed, offering the users flexible control to understand the working process of ZonedStore.

Acknowledgment

This work is supported by the National Science Foundation of China (62072419) and ByteDance Inc. Peiquan Jin and Kuankuan Guo are the joint corresponding authors of the paper.

References

[1] ZonedStorage. (2021) Zoned namespaces (ZNS) SSDs. [Online]. Available: https://zonedstorage.io/introduction/zns/

[2] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. L. Moal, G. R. Ganger, and G. Amvrosiadis, "ZNS: avoiding the block interface tax for flash-based SSDs," in *USENIX ATC*, 2021, pp. 689–703.

[3] Western Digital Corporation. (2020) Ultrastar DC ZN540 now sampling. [Online]. Available: https://blog.westerndigital.com/zns-ssd-ultrastar-dc-zn540-sampling/, 2020

[4] H. Zhao, P. Jin, P. Yang, and L. Yue, "BPCLC: An efficient write buffer management scheme for flash-based solid state disks," *International Journal of Digital Content Technology and its Applications*, vol. 4, no. 6, pp. 123–133, 2010.

[5] K. Han, H. Gwak, D. Shin, and J. Hwang, "ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction," in *OSDI*, 2021, pp. 147–162.

[6] U. Maheshwari, "From blocks to rocks: A natural extension of zoned namespaces," in *HotStorage*, 2021, pp. 21–27.

[7] T. Stavrinos, D. S. Berger, E. Katz-Bassett, and W. Lloyd, "Don't be a blockhead: zoned namespaces make work on conventional SSDs obsolete," in *HotOS*, 2021, pp. 144–151.

[8] P. Jin, X. Zhuang, Y. Luo, and M. Lu, "Exploring index structures for zoned namespaces ssds."

[9] K. Lu, P. Jin, P. Yang, S. Wan, and L. Yue, "Adaptive in-page logging for flash-memory storage systems," *Frontiers in Computer Sciences*, vol. 8, no. 1, pp. 131–144, 2014.

[10] A. Aghayev, "Adopting zoned storage in distributed storage systems," Ph.D. dissertation, Carnegie Mellon University, 2020.

[11] L. Lu, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Wisckey: Separating keys from values in ssd-conscious storage," in *USENIX FAST*, 2016, pp. 133–148.

[12] L. Lu, T. S. Pillai, H. Gopalakrishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Wisckey: Separating keys from values in ssd-conscious storage," *ACM Trans. Storage*, vol. 13, no. 1, pp. 5:1–5:28, 2017.

[13] A. Gidenstam, H. Sundell, and P. Tsigas, "Cache-aware lock-free queues for multiple producers/consumers and weak memory consistency," in *OPODIS*, 2010, pp. 302–317.

[14] ByteDance. (2021) Terarkdb. [Online]. Available: https://github.com/bytedance/terarkdb

[15] J. Li, P. Jin, Y. Lin, M. Zhao, Y. Wang, and K. Guo, "Elastic and stable compaction for lsm-tree: A faas-based approach on terarkdb," in *CIKM*, 2021, pp. 3906–3915.