CSAL: the Next-Gen Local Disks for the Cloud

Yanbo Zhou[1], Erci Xu[*1], Li Zhang[1], Kapil Karkra[2], Mariusz Barczak[2], Wayne Gao[2], Wojciech Malikowski[2], Mateusz Kozlowski[2], Łukasz Łasek[2], Ruiming Lu[1], Feng Yang[1], Lilong Huang[1], Xiaolu Zhang[1], Keqiang Niu[1], Jiaji Zhu[1], Jiesheng Wu[1]
[1]Alibaba Group; [2]Solidigm

Abstract

Cloud local disks are attractive for their affordable price and high performance. The recent advancement in CPUs motivates cloud vendors to further multiplex the computing resources to serve more users. Unfortunately, such proposals are constrained by the limited offerings of cloud local disks per server as the underlying storage devices are either large but slow (e.g., HDDs) or fast yet small (e.g., NVMe SSDs).

In this paper, we explore the possibility of leveraging high-capacity QLC-based SSDs for cloud local disks. However, the three preliminary unsuccessful attempts indicate that QLC SSDs cannot simply work as drop-in replacement. The root cause is the two levels of write amplification caused by device-level address mapping with Indirection Unit and NAND-level garbage collection.

With these lessons learned, we propose CSAL, the next-gen local disks in Alibaba Cloud. CSAL includes a high-performance SSD as write buffers and a large-capacity QLC SSD for persistence. With a two-level Logical to Physical (L2P) address mapping table, CSAL achieves fine-grained (4KB) data accessing and significantly alleviates the two levels of write amplification. Results show that CSAL always prevails with superior performance and can achieve up to 2.22×, 1.82×, and 2.03× speedups against the second-best peers in micro, application, and deployment benchmarking, respectively. As of now, we have deployed CSAL on thousands of servers and made CSAL open-source to the public.

CCS Concepts: • **Computer systems organization** → **Secondary storage organization**; • **Information systems** → **Hierarchical storage management**; **Cloud based storage**.

*Corresponding author. erc.xec@alibaba-inc.com

EuroSys '24, April 22–25, 2024, Athens, Greece

ACM ISBN 979-8-4007-0437-6/24/04…$15.00
https://doi.org/10.1145/3627703.3629566

Keywords: NAND Flash, Caching, Cloud Storage

ACM Reference Format:
Yanbo Zhou[1], Erci Xu[*1], Li Zhang[1], Kapil Karkra[2], Mariusz Barczak[2], Wayne Gao[2], Wojciech Malikowski[2], Mateusz Kozlowski[2], Łukasz Łasek[2], Ruiming Lu[1], Feng Yang[1], Lilong Huang[1], Xiaolu Zhang[1], Keqiang Niu[1], Jiaji Zhu[1], Jiesheng Wu[1]. 2024. CSAL: the Next-Gen Local Disks for the Cloud. In *European Conference on Computer Systems (EuroSys '24), April 22–25, 2024, Athens, Greece.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3627703.3629566

1 Introduction

Cloud local disk is a popular service (e.g., AWS I3[13, 14], Azure Lsv3 [15, 16], Alibaba Cloud I3 [20]) for its performance and affordable prices. In cloud local disks, the physical storage devices are directly attached to the compute servers and virtualized as block devices to the virtual machines (VMs). Given this setup, a compute node, constrained by its finite compute and storage resources, can only offer a limited number of VMs (a.k.a., ECS instances).

Recently, the performance of CPUs has been growing at a fast pace. For example, the number of CPU cores has doubled from Intel's Cooper Lake [23] to Sapphire Rapids [42]. Meanwhile, the per-core efficiency has also significantly improved [23, 42, 43, 55]. The increasing computing power motivates to further multiplex resources and accommodate more instances per server. However, this idea is constrained by the development of storage devices. On the one hand, while HDD vendors continue to supply the markets with high-density drives (e.g., Seagate and Western Digital announced 20TB and 22TB HDD [24, 31]), the bandwidth of HDDs is still around 250MB/s per drive. Therefore, their performance per TB actually drops and thus prevents the local disks to increase capacity while achieving the same Service Level Objectives (SLOs). On the other hand, high-performance Storage Class Memory (SCM) SSDs, such as Samsung Z-NAND SSD [48], Intel Optane SSD [6], Solidigm SLC SSD [53], and KIOXIA XL-Flash SSD [34], provide competitive performance but can have limited storage capacity and much higher costs [22].

One intuitive approach to break such a dilemma is to adopt the emerging QLC SSDs [26, 32, 39, 40, 50] as the local disks. Thanks to their high-density (4× higher than SLC) and low-cost (less than 1/4 of SLC), QLC SSDs provide us with large capacity at an affordable cost. Following this direction, we have experimented with three preliminary attempts based on existing techniques, including deploying the QLC SSD as

a drop-in replacement, building a layered system (i.e., Open-CAS [8]) with a high-performance SSD (HP-SSD), and using the dm-zoned (a kernel device mapper) [4].

However, all three attempts fail to deliver expected performance. Through analysis, we conclude that the root causes are the two levels of write amplification. First, QLC SSDs employ coarse-grained logical to physical (L2P) mapping (e.g., 64KB indirection unit in Intel P5316 QLC SSD [21, 26, 28, 51]). The 64KB is much larger than 4KB L2P entry in traditional NVMe SSDs, thereby incurring high device-level write amplification (i.e., 4KB logical writes become 64KB NAND writes). Second, using a Flash Translation Layer (FTL) to manage the address mapping inside QLC can also lead to severe NAND-level write amplification under garbage collection as data with different lifespans are now mixed together [17].

Based on the lessons we have learned, we propose CSAL, the next-gen cloud local disks in Alibaba Cloud. CSAL requires around 3GB DRAM for in-memory data structure, manages an HP-SSD as a write buffer, and uses a large-capacity Zoned Namespace (ZNS) QLC SSD for persistence. There are two simple and effective features in CSAL. First, we design a two-level L2P table to achieve fine-grained (4KB) data accessing and thus alleviating the device-level write amplification. Second, CSAL performs periodical compaction that reclaims write buffer space by aggregating cold data as large sequential writes to the underlying ZNS QLC SSD. The aggregation is by users (i.e., VMs) and hence can reduce NAND-level write amplification with the best effort.

We extensively evaluate CSAL against the alternatives (i.e., QLC SSD, Open-CAS and dm-zoned) with microbenchmarks (sequential/random workloads with uniform/skewed distributions), end-to-end tests on Aerospike database and RocksDB, and field-level evaluations on HDFS clusters. We observe that CSAL-based solutions delivers the highest performance and achieves up to 2.22×, 1.82×, and 2.03× speedup against the second-best option with microbenchmarks, application evaluation, and field deployment, respectively.

Here, we summarize the contributions in this paper:

- We evaluate and analyze three potential approaches to employ high-density QLC SSDs as local disks based on existing techniques. We also summarize three lessons from our exploration to guide CSAL designs.
- We present CSAL, our proposal for the next-gen cloud local disks with detailed descriptions on the in-memory data structures, on-disk layouts, and the key procedures.
- We extensively evaluate CSAL and other alternatives on raw device performance, write amplification reduction, and end-to-end applications.
- We deploy CSAL in the real-world local disk systems across thousands of nodes, and release the source code of CSAL to the public by upstreaming CSAL into SPDK [†], an open-source storage performance development kit.

[†]https://github.com/spdk/spdk (see detailed user guide in Appendix §A)

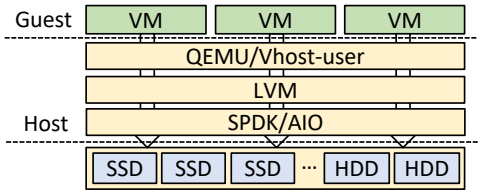

Figure 1. The Local Disk Architecture.

For the rest of the paper, we first introduce the background of the local disk in the Alibaba Cloud and QLC SSD (§2). Then we perform and analyze the three motivational studies (§3). Afterwards, we introduce the design (§4) and implementation (§5) of CSAL. Finally, we evaluate our prototype (§6), survey the related works (§7), and conclude our work (§8).

2 Background

2.1 Cloud Local Disk: (Beyond) Status Quo

Overview. In Figure 1, we show the architectural overview of the local disks in Alibaba Cloud. The host mounts the storage devices as block devices. Then, the Logical Volume Manager (LVM) further wraps the block device(s) as multiple virtual logical devices. A Virtual Machine (VM) can subscribe to multiple such virtual logical devices as its local disks via storage virtualization technologies (e.g., vhost-user-blk [58] and vhost-NVMe [61]). We use Linux AIO [41] to access HDDs (capacity-oriented local disks), and SPDK NVMe driver [60] for SSDs (performance-oriented ones). In this paper, we focus on the design for the capacity-oriented local disks.

Resource granularity. A common practice in cloud local disks (e.g., adopted by AWS[13, 14], Azure [15, 16], Alibaba Cloud [18, 19]) is that vendors set a proportion (usually an eighth) of the whole physical resources as the finest granularity. Users can subscribe to one or multiple such proportion(s) as an instance. For example, the smallest AWS D3.xlarge instance [12] has 4 vCPUs, 32GB memory, and 6TB storage space (backed by 3×2TB HDDs) whereas the largest (8× more) has 32 vCPUs, 256GB memory, and 48TB space (i.e., 24×2TB HDDs). We have the similar hardware configurations and also followed this routine. The reason behind the fixed combination is to avoid wasting resources led by over-subscribing. For example, if we allow one VM to occupy all CPU cores but only half the memory and storage space, we would not be able to accommodate other VMs even if other resources (e.g., memory and disks) remain available.

Opportunity. The recent advancement in CPU—in terms of numbers of cores and per-core processing power (e.g., from Cooper Lake to Emerald Rapids [23, 42, 43, 55])—provides us with the possibility to further multiplex the resources for more users. For example, with computing power doubled, we should be able to accommodate twice as many instances (e.g., allowing 1/16 as the finest granularity) while keeping the same Service Level Objects (SLOs).

	SLC	MLC	TLC	QLC	HDD
Space (TB)	0.6	1.2	6.4	23	22
Read (GB/s)	7.2	7.0	6.8	6.0	0.26
Write (GB/s)	6.1	5.2	4.2	2.5	0.26
IOPS-R (K)	1500	1300	1000	800	0.24
IOPS-W (K)	1350	800	150	5.7	0.46
Endurance (%)	2000	1500	333	100	-
Cost (%)	400	200	133	100	-

Table 1. Storage Comparison. *For NAND SSDs, data are provided by Solidigm [52]. Capacity is the median of current commodity SSDs; endurance, density, and cost are normalized number. Data of HDDs are from Western Digital [24].*

Challenge. While adding more memory can be straightforward, simultaneously increasing the performance and capacity of storage devices is challenging. First, if we continue to use HDDs, increasing the capacity is achievable thanks to the emergence of high-capacity HDDs, such as Western Digital's 22TB CMR HDDs [24]. While the space is quadrupled (or more), these HDDs still rave similar bandwidth (250MB/s) as earlier models. Hence, the bandwidth per TB drops to 25% or even less, thereby failing to keep the same Service Level Object (SLO). On the contrary, high-performance NVMe SSDs provide enough throughput (5-6GB/s for reads and 2-2.5GB/s for writes) but can be constrained by the capacities and much higher CapEx per TB. Note that, for efficiency, the size (i.e., rack unit) of our physical servers is 2U. Hence, we cannot simply solve the dilemma by installing more drives.

2.2 High-density NAND Flash SSD

Characteristic. For off-the-shelf SSDs, the NAND density has been gradually increasing from SLC (one bit per cell) to QLC (four bits per cell) with PLC (five bits per cell) looming on the horizon [26, 32, 39, 40, 44, 50]. Table 1 shows a comparison of SSDs based on different types of NAND. The QLC-based SSDs can provide comparable storage space to high-capacity HDDs while delivering 10× more throughput (i.e., 2.5GB/s for writes and 6.0GB/s for reads), thereby serving as a promising candidate for the local disks.

Interface. Currently, there are two types of interfaces for QLC SSDs. First, QLC SSDs can be exposed as generic block devices to serve block I/O requests, including random and sequential reads/writes. The on-device Flash Translation Layer (FTL) performs logical-to-physical address translation, garbage collection, and wear-leveling. The other interface is the Zoned Namespace (ZNS) [17] where the SSD formats its internal space into multiple sequential zones to the host. In this setup, the host needs to directly manage the random-to-sequential translation but can avoid interface tax [17].

3 Preliminary Attempts

With significantly higher throughput (up to 2.5 GB/s for writes, around 10× higher than HDD) and large capacity (20TB or more), one QLC-based SSD can replace at least 8×2TB HDDs (our last-gen local disk), yielding enough headroom for accommodating more instances. In this section, we describe three preliminary attempts at enabling the QLC SSDs. We highlight the pros and cons of these solutions, and conclude this section with our lessons learned.

3.1 QLC SSD as a Drop-in Replacement.

Setup. The much higher bandwidth motivates us to directly adopt the generic block interface QLC SSD as the local disk (Figure 2(b)) to replace the HDDs in the current approach (Figure 2(a)). In this case, the host can multiplex a large-capacity QLC SSD as multiple logical devices via LVM partitioning. Here, under the Alibaba Cloud local disk setup, we compare this solution with 8 independent 2TB HDDs. Specifically, we set 8 VMs, each with 7 vCPU cores and 28GB memory. Moreover, we use FIO inside VMs to benchmark both random and sequential write performance with 8 jobs (i.e., one for a virtual disk inside one VM) and a queue depth of 128.

Evaluation results. In Figure 3, we demonstrate performance comparisons in random and sequential writes scenarios. Obviously, in random writes workloads, the QLC SSD delivers comparable (4KB block size) or higher (8-128KB) performance. This is anticipated as the HDD can suffer significant performance loss from random workloads. Surprisingly, we notice that, in Figure 3(b), while HDDs maintain a constant throughput, the QLC delivers rather subpar performance.

Performance analysis. After discussion with the vendor, we conclude that the unsatisfying performance of the QLC SSD is caused by NAND-level and device-level write amplification. First, an outstanding characteristic of NAND is the "erase-before-write". For QLC NAND, the super block [*] size becomes much larger (e.g., 67.4GB in Intel P5316 [51]). As a result, frequent small writes lead to a significant increase in NAND-level write amplification triggered by garbage collection. Figure 4(a) lists the block size distribution of Alibaba Cloud local disks. We can see that small block writes (e.g., 4KB-16KB) account for more than 60% in local disks. Moreover, in Figure 4(b), we list the NAND-level write amplification calculated from different typical applications on non-shared (i.e., each application owns an independent SSD) TLC-based SSDs in the field deployment (blue bars). The average write amplification is 2.3 of these four applications. However, when these four applications share a single QLC SSD, the average write amplification of them increases to 3.8 (orange bars). This is because sharing the same large QLC SSD will introduce additional write amplification as the different write patterns of applications would cause data

[*]A super block in the NAND Flash SSD refers to a collection of blocks from different planes that are treated as a single unit for operations such as erasing and programming.

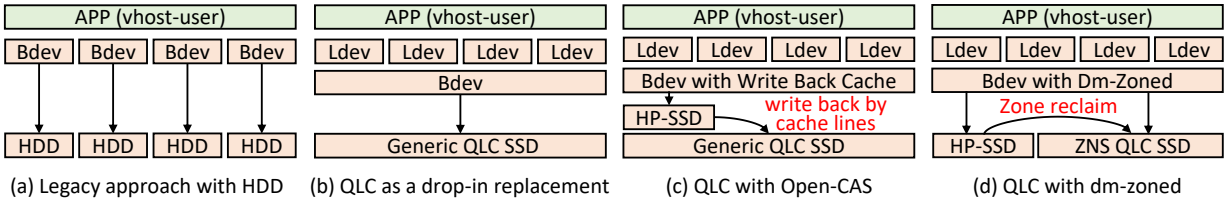

Figure 2. Comparison among Existing Approaches to enable QLC SSDs in the Local Disk Instance.

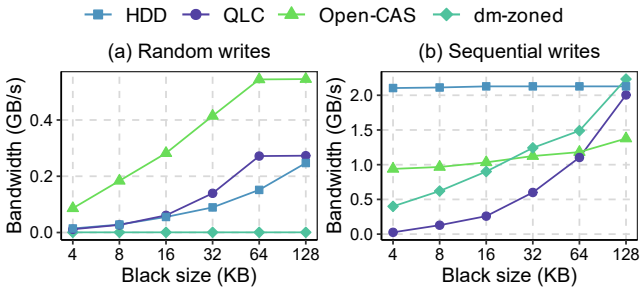

Figure 3. Performance of Different Approaches.

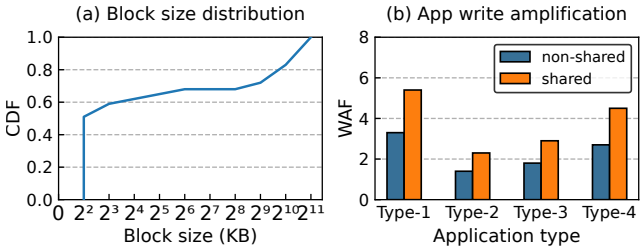

Figure 4. Real Workload Observation.

with different lifespans (from different users) to be stored in the same erase block.

Second, the generic block interface SSD employs an address translation table to map logical block address (i.e., host LBA) to physical block address (PBA) on NANDs. The Flash Translation Layer (FTL) maintains this Logical-to-Physical (L2P) mapping table in the on-chip DRAM. For a common NAND SSD, the size of each entry (some SSD vendors call "indirection unit", IU [21]) in the L2P table is usually 4KB to align with the host OS page size. The typical size of an L2P mapping table in a 1TB TLC-based SSD can be as much as 1GB. In a QLC-based SSD, if the IU size remains at 4KB, the total size of the mapping table would grow to 16-32GB, which is too large for the on-chip memory to accommodate. Therefore, current QLC-based SSDs set a larger indirection unit size (e.g., 64KB in our Intel P5316 SSD) to shrink the mapping table.

Consequently, for alignment, small writes in QLC-based SSD would suffer device-level write amplification and performance slowdown. In Figure 5, we showcase two examples to illustrate such impact. In Figure 5(a), assume the host issues a 4KB update request. The FTL first looks up the entry in the L2P table and reads the corresponding 64KB data to the

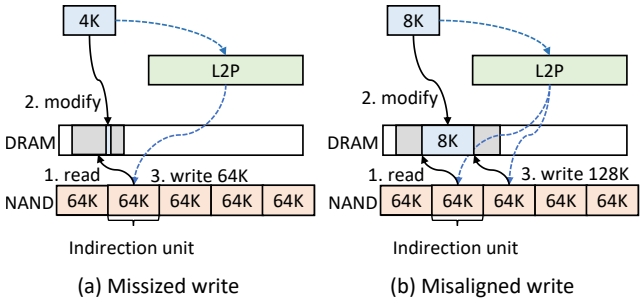

Figure 5. Non-optimal Writes under 64KB Indirection Unit. *Solid lines are data flows; dotted lines are data indexs.*

on-chip DRAM. After updating the 4KB, the SSD controller then flushes the whole 64KB back to the NAND media, causing a 16× device-level amplification. The second example, as shown in Figure 5(b), is when a larger write request (e.g., 8KB) involves updating to two IUs (i.e., 4KB from each IU), meaning the writes are not IU-aligned. Such misalignment would also cause significant device-level write amplification (16× in our 8KB example).

Endurance impact. Apart from performance influence, there is another significant problem associated with the two levels of write amplification: the endurance impact. As we introduced in Table 1, a side effect of squeezing more bits into a NAND cell is the shortened lifespan (i.e., less number of allowed Program/Erase cycles) [30]. Unfortunately, the aforementioned double write amplification and large volumes of workloads of local disks further exacerbate this issue, leading to the premature retirement of QLC SSDs. In our case, the 15.36TB QLC SSD (i.e., the Intel P5316) offers a 5-year 1.83 Drive Writes Per Day warranty under the optimized writes (e.g., 64KB IU-aligned sequential writes), indicating the SSD can endure a 28.78TB (15.36TB×1.83) of writes per day for five years. Note that 28.78TB can be directly regarded as NAND writes, as there is no NAND-level or device-level write amplification under 64KB IU-aligned sequential writes.

In Figure 4(a) and Table 2, we respectively list the distribution of write size and amount of writes-per-day on ECS instances. Also, the FTL reports the average NAND-level write amplification on a local disk backed by TLC-based SSD is 2.2 in the field. Therefore, we can calculate the actual daily NAND writes on a local disk backed by a QLC-based SSD. For example, a 4KB write has a Write Amplification Factor (WAF)

	Logical Writes (TB)	NAND Writes (TB)
p50	1.23	25.07
p75	1.42	28.94
p90	1.58	32.20
p99	2.20	44.84
p999	2.54	51.76
p100	2.94	59.92

Table 2. Distribution of Writes-per-day on 8× 2TB Local Disks of ECS Instances.

of 35.2 (i.e., 16×2.2, device-level write amplification multiplying NAND-level write amplification) and hence inflates a 140.8KB NAND write. In the rightmost column of Table 2, we list the corresponding NAND writes after calibration with WAF and the real data block size distribution in Figure 4(a). We can see that 75% of tenants with at most 28.94TB NAND writes (≈ NAND writes per day QLC can endure) for a drive per day can use QLC SSDs for 5-years. However, the rest 25% of tenants incur large volumes of NAND writes which are beyond QLC SSD endurance. In order to satisfy 99.9% of tenants, the QLC SSD has to endure 51.76TB NAND write per day if we directly employ QLC SSDs. Note that this is still an underestimation as we use the NAND-level WAF from TLC-based SSDs for calculation. Moreover, if we use one QLC SSD to replace multiple HDDs (e.g., 8 in this example), the QLC SSD would have to endure even more writes.

3.2 Write-Back Cache

Setup. Alternatively, we can use a write-back cache to absorb small writes and thereby reduce the write amplification. To ensure data persistence, we employ an 800GB high-performance SSD (HP-SSD, Intel P5800X [6]) as our cache. We use Open-CAS [8] to build our LRU-based cache with a line granularity of 64KB to align with the 64KB indirection unit of QLC SSD (see Figure 2(c)). We reuse the environment from the previous experiment and rerun the same workloads with FIO.

Evaluation results. Figure 3 shows that adding cache considerably improves the random writes performance as frequent small random writes are absorbed (i.e., writes hit) and merged by the cache. Regarding sequential writes, while we observe performance improvements (compared to the single QLC solution) on small writes, its throughput still lags behind the traditional HDD. Note that such performance gap persists under various cache eviction policies (e.g., LRU, LFU) and different cache line granularity (e.g., 32K, 64K, and 128K). Therefore, inserting a cache layer still fails to meet our goal.

Performance analysis. The main rationale for adopting a write-back cache is to absorb user writes in a cache layer and group small writes that target the same IU (i.e., cache line), thereby reducing both NAND-level and device-level write amplification. If the capacity of the write-back cache is sufficient (i.e., larger than the underlying QLC SSDs combined), we can simply enforce the cache to write back a line only when it is full (i.e., all 64KB are updated), resulting in a device-level WAF of 1. Apparently, such solution is impractical due to the high cost and the limited capacity of HP-SSDs. Therefore, we need to periodically flush the cache, thereby incurring considerable device-level write amplification and performance loss.

3.3 ZNS-based Solution

Setup. From the previous experiments, we notice that the major bottleneck that limits the performance of QLC SSD is the indirection unit size of the L2P table. Now, with Zoned Namespace (ZNS) support, we are able to lift this constraint by managing the mapping in the host. Additionally, ZNS SSD provides zoned layout to route different write streams into isolated erase blocks (i.e., zones), thus allowing host to perform tenant-/workload-aware allocation to reduce NAND-level write amplification.

Currently, Linux provides a device mapper, called the dm-zoned [4], to adopt Zoned storage devices while maintaining the random write interface. Unfortunately, ZNS-based QLC SSDs are still not readily available so that we use a West Digital ZNS-based TLC SSD [7] to build our prototype. Figure 2(d) outlines the architecture where the dm-zoned splits zones into two parts including a series of random zones (backed by the HP-SSD) and a series of sequential zones (backed by the ZNS SSD). Moreover, the dm-zoned maintains a mapping table in the host memory to translate logical block address to the location in zone. In this setup, all sequential writes will directly route to the sequential zones while random writes are first allocated to the random zones and then migrated to the sequential zones in bulk (i.e., an entire zone).

Evaluation results. In Figure 3, we can see that, in the case of sequential writes, dm-zoned solution is consistently faster than the QLC SSD. Compared with write-back cache, dm-zoned delivers lower throughput among writes (i.e., 4-16KB) and almost the same throughput on 32KB, but eventually overtakes cache-based solution with 13.72%-62.20% speedup on large writes (i.e., 64-128KB). The reason is that the small sequential writes are merged in write-back cache. But, for large writes, directly appending them to the sequential zones avoid paying the interface overhead and improves performance. In stark contrast, for random writes, the dm-zoned solution snails with average 10MB/s throughput.

Performance analysis. Now, we further analyze the root cause behind the performance of dm-zoned. In practice, dm-zoned relies on random zones to allocate random writes. Due to the limited capacity of HP-SSD, the random zones would soon run out and thus require the dm-zoned to reclaim space by migrating the data from random zones to sequential ones in ZNS SSD. Note that the current design of dm-zoned enforces "one-for-one" policy where the entire random zone

would be migrated to a sequential one and empty areas would be padded with zero. Under high-pressure workloads, the dm-zoned will be constantly busy with migrating data, resulting in huge performance loss. The dm-zoned community has confirmed our analysis and suggests that the current dm-zoned, designed for shingled magnetic recording (SMR) HDDs with sequential-intensive writes, only provides "experimental" support for random-intensive writes.

3.4 Lessons Learned

Our three preliminary attempts fail to achieve our goals as the next-generation local disks. Nevertheless, we obtain the following lessons to guide our design.

- The two levels (i.e., NAND-/device-level) of write amplification significantly influence the efficiency and endurance of QLC-based SSDs.
- The ZNS interface allows the host to perform workload-aware allocation (i.e., separating hot/cold data by zones), thereby alleviating NAND-level write amplification.
- ZNS-based solutions (e.g., dm-zoned) can invoke severe performance penalties on random writes due to the larger granularity (i.e., zone) of address management.

4 Design

In this section, we illustrate and discuss the design of CSAL at length. We start by introducing the core ideas. Then, we present the architecture and data layout. Finally, we go through the key procedures in CSAL.

4.1 Design Overview

Core ideas. CSAL adopts two straightforward ideas, using a page table to achieve fine-grained (i.e., 4KB) logical to physical address translation and using a high-performance SSD (HP-SSD) as a log-structured write buffer. First, we build a two-level mapping table. The table is set with a 4KB granularity where the index is the host LBA and each entry contains the corresponding PBA. Due to memory capacity constraints, we do not store the entire table in the memory. Rather, we hold the top-level in the host DRAM and swap parts of the second-level table (i.e., pages) between memory and the high-performance SSD (HP-SSD) based on an LRU policy. Second, we route all incoming writes (random or sequential) to the HP-SSD as append-only writes and then periodically aggregate and flush the data to the zones in the underlying QLC SSD with workload-aware compaction.

Benefits. The advantages of CSAL design are two-fold. First, directly mapping the LBA to PBA with 4KB granularity alleviates the device-level write amplification while keeping a controllable DRAM footprint by leveraging a two-level page table with LRU policy. Second, CSAL groups data with similar lifespans (i.e., from same users) in HP-SSD and flush to the QLC SSD at the granularity of zones (1GB in our case), thereby reducing NAND-level write amplification.

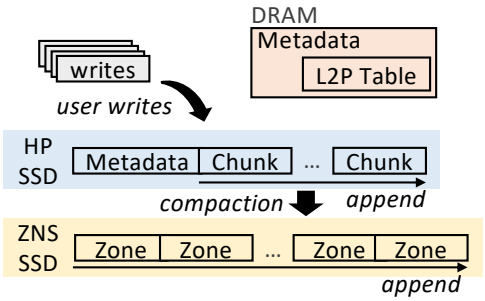

Figure 6. Architecture Overview.

Architecture. At a high level, Figure 6 describes the architecture of CSAL, including the memory, the high-performance SSD (HP-SSD) and the ZNS QLC SSD. Note that each CSAL system includes a configurable length of memory space (3GB by default), one high-performance SSD, and one QLC SSD. The memory holds the L2P table, including the entire top level and a proportion of the second level. The high-performance SSD stores the entire two-level L2P mapping table and the write buffer. The CSAL organizes the write buffer as a series of logs (i.e., data chunks). Each data chunk comprises multiple 4KB data segments, each with their metadata (as a 64B padding). The QLC SSD is composed of multiple fixed-size append-only zones and each zone has the same format as the chunk in HP-SSD. We further describe the detailed layouts of the above in §4.2. Note that the QLC-based ZNS SSD may not be readily accessible at the moment. Instead, we can use multiple TLC ZNS SSDs or standard QLC SSD with ZNS simulation (via SPDK zoned block device).

Procedures. CSAL has three main write procedures including user write, data compaction and garbage collection. First, when a user (i.e., VM) issues a write request, the CSAL packs the data as a series of 4KB data segments and 64B metadata padding, then appends them to the end of the current chunk in the high-performance SSD. Then, CSAL updates the L2P table in the memory and returns a "write_success" to the client. Periodically, CSAL performs compaction, which merges valid data segments (data segments can be obsolete after an update or delete) based on the workload information, flushes them to the QLC SSD as large (128KB) sequential writes and updates the L2P table. When the available space on the QLC SSD is running low, CSAL performs garbage collection in the QLC SSD and updates the L2P table accordingly. Note that the client read is straightforward—locating the PBA via the L2P table and reading data from the corresponding high-performance and/or QLC SSD. We discuss the procedures at length in §4.3.

Key challenge. During the design discussion and test runs, we realize a major challenge in CSAL is crash consistency. At any given time, a LBA address may be touched by three write procedures (i.e., client write, compaction and GC). In §4.4, we discuss different scenarios and corresponding solutions.

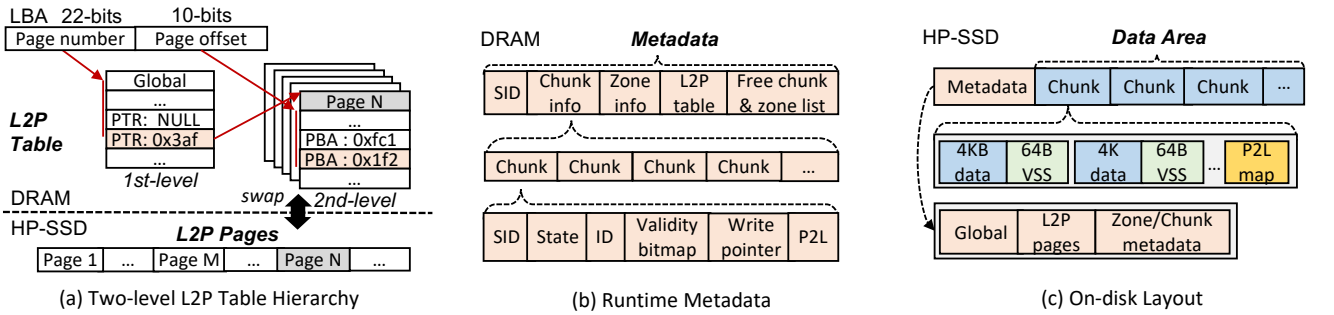

Figure 7. CSAL Data Structures and Layout.

4.2 Data Structures & layouts

L2P table. Figure 7(a) shows the design of the L2P table. The index of each entry is the 4KB host LBA and the content is the corresponding 4KB PBA on the devices (HP-SSD or QLC SSD), thereby achieving a direct mapping from host to device. With a 16TB (730GB from high-performance SSD and 15.36TB from QLC SSD) address space, the LBA and PBA are both 32 bits long. We adopt a two-level table design where we group every 1K table entries (4 Byte each) as a 4KB page. Hence, to locate the PBA of an incoming I/O, CSAL first finds the page by *LBA*/*page_size* and then identifies the entry via calculating the page offset (i.e., *LBA%page_size*). In practice, CSAL only stores the page table (around 32MB) in the memory and uses another 2GB DRAM to cache the recently accessed pages. We will further discuss how to protect crash consistency for L2P table in §4.4.

In-memory layout. Figure 7(b) illustrates the in-memory layout of CSAL. Apart from the page table and page caches, we include a Sequential ID (SID), the chunk/zone status and a list of free chunk/zone. Recall that chunk and zone are both append-only and of the same size (1GB in our case) except chunks are in the HP-SSD and the zones are the basic units of QLC SSD. Next, we introduce the functionality of each component.

- *SID.* The SID is 8 byte monotonically increasing Universally Unique IDentifier (UUID) that starts at 0. When a new chunk/zone is opened, the SID is increased by one and assigned to that chunk/zone. We use SID to resolve LBA conflicts (see detailed usage in §4.4).
- *Chunk/Zone Status.* The memory records the status of each zone/chunk, including its SID, the state (e.g., opened, closed or free), the chunk/zone ID, and a write pointer to the location for appending. We also enclose a bitmap to reflect the validity (stale or not) of each 4KB (called a block) inside the chunk/zone.
- *Free Chunk/Zone List.* CSAL maintains a list of currently free chunks and zones to save time from scanning the chunk/zone status during allocation.
- *P2L.* CSAL also builds a physical to logical map to accelerate compaction. Note, in DRAM, CSAL only maintains the P2L of current open chunks/zones. After chunks/zones are closed, the responding P2Ls are flushed to SSDs and freed from memory (see detailed usage in §4.3).

On-Disk Layout. Figure 7(c) shows the two parts of high-performance SSD layout, the metadata region and the data region. The metadata region mainly acts as the persistence of the in-memory data structures that contain the most recently synched SID, the chunk/zone status and the entire two-level L2P table. CSAL organizes the data region as a series of chunks. Each chunk further contains multiple sets of a 4KB block (containing data) along with a 64B padding (containing metadata, including SID and LBA). Only one chunk is open to receive incoming writes. Note that we leverage the Variable Sector Size (VSS) support to enable the 64B padding. For drives without such capability, the CSAL can simply merge the paddings as separate data blocks. The on-disk layout of QLC SSD is the same as the high-performance SSD except that QLC SSD only has a data region and opens two zones (one for compaction and the other for garbage collection).

4.3 Control-/Data-flows

Read. Handling read request is straightforward in CSAL. First, CSAL locates the page by *LBA*/*4KB* (*page_size*). If the targeted page is not loaded to the memory, CSAL fetches the page from the high-performance SSD and obtains the offset within the page via *LBA%4KB*. Finally, CSAL retrieves the data by issuing the PBA to the corresponding device (the HP-SSD or the QLC SSD).

Write. Figure 8(a) shows the flow of user writes. CSAL performs as log-structured block storage that always appends the incoming write to the currently opened chunk in the HP-SSD as a series of sets of 4KB data block and 64B metadata padding. The metadata includes an immutable sequential ID (same as the open chunk), the LBA of the block, and the user ID. As multiple logic disks share same cache space, CSAL groups logic disks as different VMs and tag their data with a unique user ID (represented as different colors in Figure 8). Afterwards, similar as the read procedure, CSAL locates the L2P entries and alters the corresponding LBA to PBA mapping in the L2P table. Finally, CSAL updates the validity

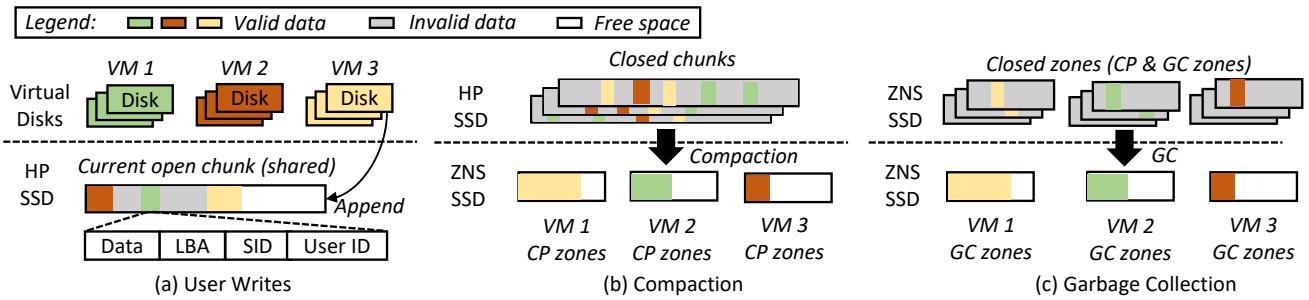

Figure 8. CSAL Data Flow. *The data with different colors (i.e., green, red, and yellow in the figure) means they come from different users (i.e., different VMs). CSAL groups data from same users into same zones in underlying high-density QLC SSDs.*

bitmap to mark previous LBAs as invalid (i.e., stale) and the latest LBAs as valid.

Compaction. The HP-SSD works as the append-only write buffer in CSAL. With limited capability, the HP-SSD would eventually run out of space and need to evict chunks to QLC zones. CSAL periodically invokes compaction when free chunks are less than threshold (by default 20%). Figure 8(b) shows the compaction process. First, CSAL scans all closed chunks and pick the chunk with the lowest valid block ratio by checking validity bitmaps. Then, CSAL reads valid blocks and their VM IDs (stored in 64B padding space) from the selected chunk, then aggregates data into DRAM buffer grouped by user ID. Finally, CSAL batches data appending into per-VM compaction (CP) zones (denoted by different colors in Figure 8(b)) when the buffer reaches the 512KB and free the selected chunk when all the valid data are migrated into zones (reset and move the chunk to free chunk list). CSAL iterates this compaction process until the number of free chunks is larger than or equal to the threshold.

Compared with the dm-zoned solution, there are two outstanding differences. First, unlike a simple one-for-one dumping policy, CSAL would merge multiple (usually 256K) 4KB blocks together to fill an entire zone (1GB each). Second, CSAL also compacts blocks by VMs to group data with similar lifespans (i.e., from same users) and flush them to the same zones (as shown in Figure 8(b), the data from different VMs are migrated into physically isolated zones), a best-effort approach to reduce NAND-level write amplification.

Garbage collection. With the log-structured nature, the QLC SSD inevitably demands garbage collection (GC) to reclaim the space occupied by stale data. Figure 8(c) shows the GC process, which uses the same policy as compaction except the threshold. By default, CSAL reserves 10% space of ZNS SSD for over-provisioning and performs GC when there are less than 5 free zones. GC process periodically checks closed (i.e., used) zones, including both compaction and GC re-written zones, selects the zone with lowest valid block ratio, and finally batches valid data into per-VM GC zones (denoted by different colors in Figure 8(c)). Note that CSAL does not mix compaction and GC writes into the same zones. Instead, CSAL maintains an independent GC zone for each VM.

Reserving over-provisioning space and setting the GC threshold enable CSAL to only perform reclaiming when there is not enough free space. At this moment, the overall system has maximum invalid blocks ratio, thus only requiring to migrate a small number of valid blocks. Moreover, the GC process isolates GC and compaction re-written data, which reduces NAND-level write amplification by separating data with different lifespans (from different VMs).

Moreover, through three types of write streams, user data is placed in three layers. The frequently updated data (short lifespan) is stored in the write buffer of HP-SSD and the rarely updated data (long lifespan) is stored in GC zones of ZNS SSD. The left data is stored in user zones of ZNS SSD (these types of data have varying lifespans of different users and are physically isolated in different zones).

4.4 Crash and Concurrency Consistency

In CSAL, all three write procedures are asynchronous and need to update the in-memory L2P tables. Therefore, CSAL needs to guarantee data consistency in face of writes reordering and crashes. Next, we introduce the detailed mechanisms in CSAL for maintaining consistency. Instead of directly touring the full-fledged version, we, for better clarity, start with a naive design and then discuss the improvements led by various optimizations.

Naive solution. Recall that, for each 4KB data block, CSAL attaches a 64B padding that includes the LBA and an immutable SID. Now, we describe a naive restoring process without using any persisted metadata. Regarding the L2P table, we can simply rebuild the mapping by scanning the data regions on high-performance SSD and QLC SSD to obtain the PBA and the corresponding LBA (from padding) of each data block. After obtaining the L2P table, we can now restore the chunk and zone status. First, we restore the zone states by marking zones without any data blocks as free zones and the rest as closed. Second, we rebuild the validity bitmaps and restore SIDs by scanning the L2P table.

Resolving LBA conflicts. Note that, as data blocks can be invalid, there are three scenarios that could cause multiple PBAs to point to the same LBA during restoring. First, an LBA can be in the open chunk after a recent update while an earlier version is in the QLC SSD after compaction. Second, a crash may occur during compaction/GC, resulting in different PBAs pointing to the same LBA (e.g., with one in chunks and another in zones). Third, the chunk can contain multiple data blocks on the same LBA due to frequent updates. To resolve conflicts, we always trust the data block with the highest SID (i.e., first case). If the SIDs are the same but come from different SSDs (i.e., chunk and zone), it means they hold the same data as the SID is immutable (i.e., compaction/GC does not update SIDs of data blocks). For the third case where SIDs are the same and also from the same chunk, we will use the data block with the largest PBA as data blocks are appended. Note that we can also use the above principles to resolve writes reordering caused by asynchronous writes.

Optimization 1: adding P2L table. Clearly, the naive solution is far from efficient as it requires a whole address scanning. In our case, reading the whole address space (around 16TB) requires nearly an hour under a 6GB/s sequential read throughput—an unbearable overhead. Therefore, when a chunk/zone closes after filled up, we also enclose a Physical to Logical (P2L) table to include the reverse address mapping of all the data blocks within that chunk/zone. Moreover, we persist the chunk/zone states in the metadata region of the high-performance SSD. As a result, during restoring, we no longer need to scan the whole address space but only the tail of each zone/chunk and three open ones (open chunk, two open zones for compaction and GC). This reduces the 16TB reading to 32GB P2L table scanning (i.e., a 64bit SID, a 32bit LBA and a 32bit PBA for each data block) plus checking three open chunks/zones (i.e., 3GB).

Optimization 2: adding checkpoint & clean flag. Recall that in the metadata region of high-performance SSD, CSAL keeps track of the complete two-level L2P table due to limited memory capacity. Therefore, we can leverage the already-persisted L2P table to accelerate restoring by adding a clean flag and a checkpoint. CSAL sets the binary clean flag during a normal shutdown, enabling a simple recovery from reloading the metadata region in high-performance SSD. The checkpoint is essentially a SID, indicating that any L2P entries before this SID are safely recorded in the high-performance SSD. Thus, during recovery, we can now first load the L2P table from the metadata region and then only check the entries with a higher-than-checkpoint SID. Assume we checkpoint every 1TB of writes, then we only need to read the 16GB L2P table, at most 1GB of P2L entries (i.e., from 1TB unpersisted writes) and 3GB of open chunks/zones. More importantly, the checkpoint saves CSAL from the expensive (in terms of CPU and memory) and the large volumes of LBA conflicts resolving. In stark contrast, with the two optimizations, we now need only around 5 seconds to restore from a crash (see detailed results in §6.4).

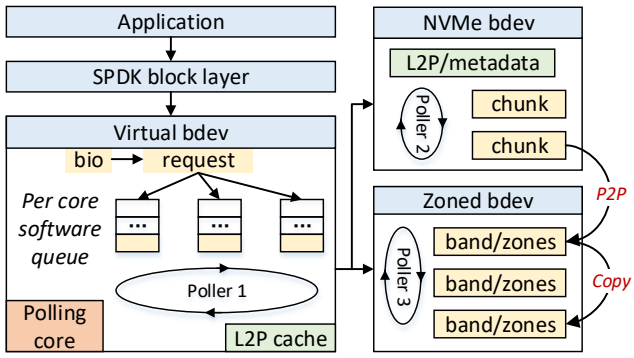

Figure 9. CSAL Implementation.

5 Implementation

In Figure 9, we depict the implementation of CSAL. We expose CSAL as a virtual block device with the help of SPDK [3], a user-space storage kit. CSAL contains two block devices, a high-performance SSD and a ZNS QLC SSD. Via SPDK, CSAL registers a virtual block device to the SPDK block layer serving applications that use the generic block interface.

Since SPDK applications use a dedicated CPU core for each thread, CSAL creates multiple I/O queues for each application thread. I/Os from different threads (i.e., users) are dispatched to different I/O queues, which avoids threaded logging overhead. Further, CSAL uses a dedicated CPU core, with an abstract thread on top of that, in polling mode for internal events processing. A single dedicated CPU core can eliminate software overhead caused by locks in the I/O path and guarantee consistency without any locks. CSAL registers three *pollers* in its thread for I/O processing of client, compaction, and GC tasks (Note *poller* is an abstraction for a function that should be repeatedly called on the given thread in SPDK).

6 Evaluation

In this section, we compare CSAL against our HDD-based solution and three earlier attempts. We evaluate these candidates by measuring the raw performance (§6.2), the write amplification (§6.3), the effectiveness of our techniques (§6.4), the application-based benchmarks (§6.5), the resource consumption (§6.6) and field deployment statistics (§6.7).

6.1 Experimental Setup

Environment. In Table 3, we list the basic information. We use an 800GB Intel P5800X Optane SSD [6] as the high-performance SSD (HP-SSD). As the ZNS QLC SSDs are not readily available at the moment, we provide two alternatives, a standard non-ZNS Intel P5316 QLC SSD with 15.36TB capacity [51] (emulating ZNS functionalities with SPDK zone

CPU	2x Intel 8369B @ 2.90GHz
Memory	256GB DDR4 Memory
High Performance SSD	1× Intel P5800X 800GB
Non-ZNS QLC SSD	1× Intel P5316 15.36TB
ZNS TLC SSD	4× WD ZN540 4TB (ZNS)
OS	Linux CentOS Kernel 4.19

Table 3. System Configuration.

bdev [11]), and 4× ZNS-ready ZN540 TLC SSDs with 4TB capacity each [7]. For fairness, we bundle 4× ZN540 SSDs together and throttle their performance to match the capacity and the performance of a P5316 SSD (i.e., 15.36TB capacity with 6GB/s for read and 2.5GB/s for write).

Test candidates. We compare CSAL against our legacy 8 HDDs solution and three earlier attempts. We use the deployed systems as the HDD-based solution (denoted as HDDs and described in §2). We reuse the same setups for three preliminary attempts (denoted as QLC, Open-CAS and dm-zoned with description in §3, respectively) and stick to the implementation details for CSAL (denoted as CSAL-BLK for CSAL on Intel QLC P5316 SSD and CSAL-ZNS for CSAL on four ZN540 TLC SSDs).

Resource allocation. For raw QLC SSD, Open-CAS, CSAL-BLK, and CSAL-ZNS, we set one hyper-threaded CPU core for I/O polling in SPDK. For dm-zoned, we do not limit any CPU resource to perform interrupts for the I/Os. We limit the memory usage to 3GB for each solution, enough to hold all the cache line metadata for Open-CAS. CSAL-BLK/ZNS utilizes the 3GB memory for storing the top-level L2P table and caching L2P pages. The HDD, the raw QLC SSD, and the dm-zoned all have minimal memory footprints and thus are not bounded by the allocated 3GB DRAM.

6.2 Raw Device Performance

Setup. We test the candidates as raw block devices in direct mode (direct=1 in FIO [5]) with no file systems mounted. We use the FIO to generate workloads. Specifically, for QLC SSD, Open-CAS, and CSAL-BLK/ZNS, we use user-space spdk_bdev as the io_engine. For dm-zoned, we use Linux AIO [41]. Before the tests, we warm up the SSDs with twice whole-disk sequential writes followed by four hours of 4KB random writes. For raw device, we first evaluate writes and reads (including uniform and skewed workloads for both of them), and then we evaluate mixed reads and writes.

Uniform writes. We rerun the tests in Section §3, where we set several sequential and random workloads. In each workload, 8 FIO jobs issue I/Os with queue depth 128 to 8 virtual disks from 8 VMs. All workloads cover the entire LBA range. Note the performance of the HDDs, raw QLC, Open-CAS, and dm-zoned in Figure 10 are the same as in Figure 3. Figure 10(a) compares the performance of random writes. CSAL-BLK and CSAL-ZNS achieve similar performance all along

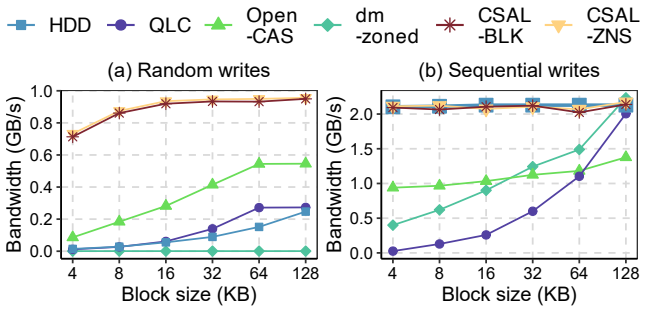

Figure 10. Uniform Writes Workloads.

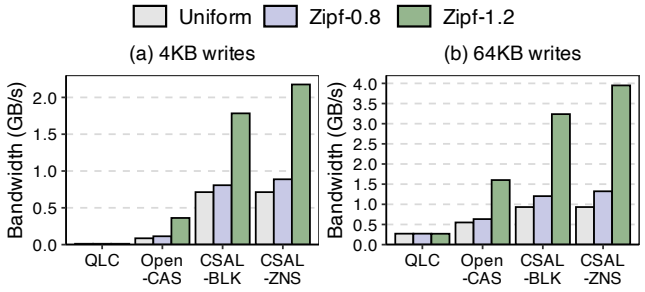

Figure 11. Random Writes with Zipf Distribution.

and can be up to 97.37×, 61.6×, 8.28× faster than HDDs, raw QLC, Open-CAS, respectively. Figure 10(b) shows sequential writes performance among candidates. HDDs, CSAL-BLK and CSAL-ZNS all constantly deliver 2GB/s bandwidth and achieve an up to 81.45×, 2.22×, and 5.24× speedup against raw QLC, Open-CAS, and dm-zoned, respectively.

The superior performance is due to the fact that CSAL compacts updates in HP-SSD by 4KB and flushes them to the underlying TLC/QLC SSD as 512KB sequential writes. Therefore, both CSAL-BLK and CSAL-ZNS achieve near-optimal performance (i.e., bounded by the bandwidth of ZN540 and Intel P5316) in sequential scenario. Due to GC, the performance bandwidth under random writes are relatively lower. Note that SSDs have been warmed up and thus the smaller size of random writes and the bigger influence of GC.

Skewed writes. Now, we further evaluate the performance of candidates under two sizes of writes (i.e., 4KB and 64KB) with different degrees of skewness, including none (i.e., uniform), light (a Zipf of 0.8) and heavy (a Zipf of 1.2). In Figure 11, we present the results for each candidate, except for dm-zoned, due to its negligible performance—consistently less than 10MB/s. We can see that CSAL-ZNS always delivers the highest bandwidth followed by CSAL-BLK. Note that more skewed writes indicate more frequent updates on the same LBAs, thereby preferring the cache-based solutions (i.e., Open-CAS, CSAL-BLK and CSAL-ZNS). Open-CAS suffers performance loss due to the large granularity of IU (64KB). Further, CSAL-BLK shows lower performance due to NAND-level write amplification caused by the SPDK ZNS

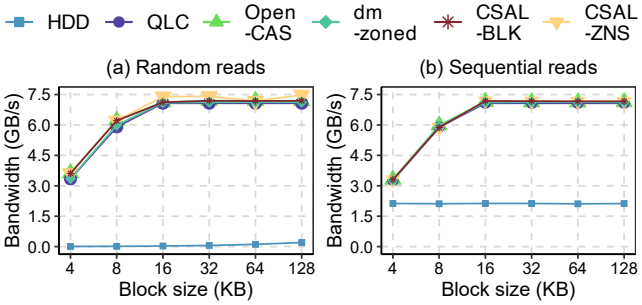

Figure 12. Uniform Reads Workloads.

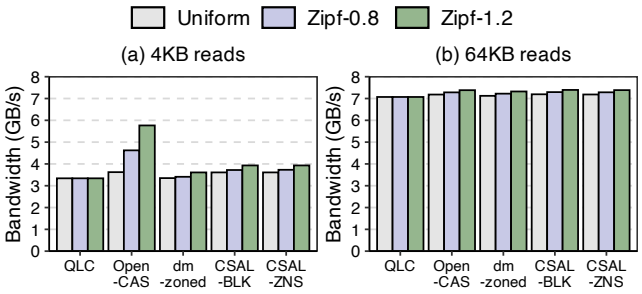

Figure 13. Random Reads with Zipf Distribution.

emulation—as GC and compaction streams would still be merged by the Flash Translation Layer (FTL) of the non-ZNS Intel P5316 SSD.

Uniform reads. Figure 12 presents uniform read workloads, including random reads (Figure 12(a)) and sequential reads (Figure 12(b)). All of the candidates using QLC show significantly higher performance results than the combined performance of 8 HDDs in both random and sequential workloads. Open-CAS and CSAL-BLK/ZNS demonstrate similar performance while dm-zoned shows a bit lower performance than Open-CAS and CSAL-BLK/ZNS. QLC itself already achieves much higher read performance than HDDs, therefore the read performance of QLC is satisfying for the service-level objective of the local disk (i.e., same or higher throughput than 8× HDDs combined). As dm-zoned is not optimized for NVMe SSDs, software routines result in a certain amount of overhead, leading to slightly lower performance than Open-CAS and CSAL.

Skewed reads. Figure 13 shows the skewed read workloads under 4KB (Figure 13(a)) and 64KB (Figure 13(b)) block sizes, including non (i.e., uniform), light (i.e., a Zipf of 0.8), and heavy (i.e., a Zipf of 1.2). We can see that, unlike uniform read workloads, Open-CAS shows higher performance results than all other candidates. This is anticipated because the 4KB random performance of HP-SSDs is higher than QLC SSDs. Additionally, Open-CAS fills back data from QLC to the cache layer for frequently used data, while CSAL and dm-zoned do not migrate data under reads. For 64KB skewed reads, raw QLC achieves similar performance as HD-SSD so

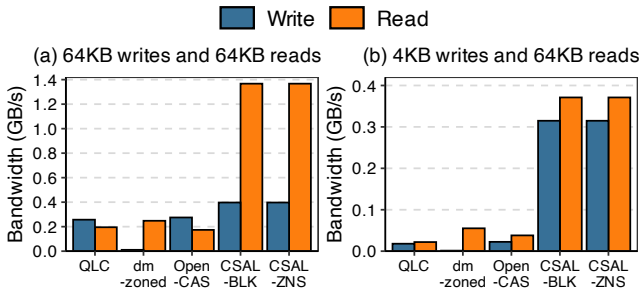

Figure 14. Mixed Random Writes and Reads.

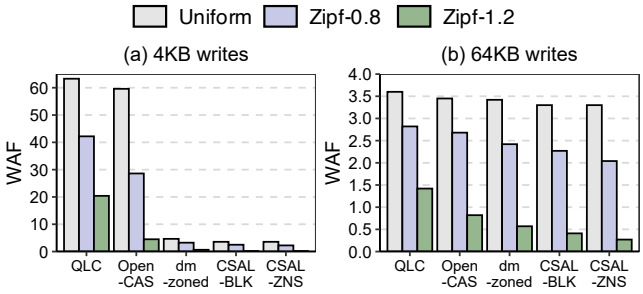

Figure 15. Write Amplification.

that there is no obvious difference between all candidates. We designed CSAL towards optimizing writes and endurance since QLC reads already achieve satisfying results. Therefore, the performance of reading from either HP-SSD or QLC SSD can meet the service-level objective of local disks (i.e., same or higher throughput than 8× HDDs combined).

Mixed reads and writes. For read/write hybrid performance, we set up two scenarios, 64KB random read workloads with 4KB random writes or 64KB random writes workloads. For 4KB writes, we use 7 FIO jobs for writes and 1 FIO job for reads. For the 64KB write case, we employ 4 FIO jobs for writes and other 4 jobs for reads. We limit bandwidth per job as 700MB/s in both scenarios. Again, we observe that CSAL-BLK and CSAL-ZNS prevail with highest performance (see Figure 14). This suggests our thread model and polling modes can work well under mixed read and write workloads, a typical scenario for local disks in Alibaba Cloud.

6.3 Measuring Write Amplification

Next, we measure the Write Amplification Factor (WAF) to further validate the benefits driven by CSAL design. We obtain the WAF by dividing the total size of NAND writes (reported by the device internal monitoring mechanism, i.e., the SMART [25]) by the total logical writes (via FIO).

In Figure 15, we demonstrate the WAFs under 4KB and 64KB random writes for the three levels of skewness, including none (uniform), light (Zipf-0.8) and heavy (Zipf-1.2). First, we observe a notable trend across all cases where a more skewed distribution leads to a lower WAF. This is because frequent updates on the same LBAs (a highly skewed

Figure 16. Backend Traffic under 4KB Random Writes. *dm-zoned performs reclaiming from random to sequential zones (RC-1) and from sequential to random zones (RC-2). Open-CAS only performs flushing. CP and GC mean compaction and garbage collection.*

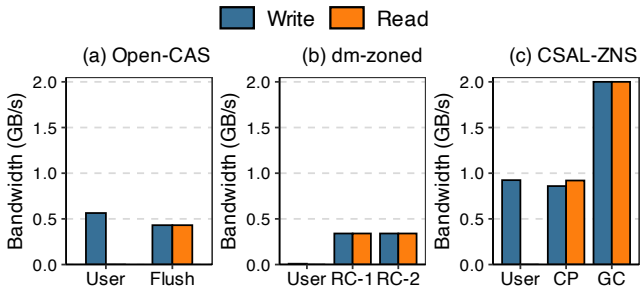

Figure 17. Backend Traffic under 64KB Random Writes. *dm-zoned performs reclaiming from random to sequential zones (RC-1) and from sequential to random zones (RC-2). Open-CAS only performs flushing. CP and GC mean compaction and garbage collection.*

distribution) indicates that more writes are absorbed by the cache (i.e., HP-SSD) and consequently less data are flushed to the underlying TLC/QLC SSD.

Second, we notice that WAFs among candidates show a stark contrast under 4KB random writes workloads, whereas the differences are less evident under the 64KB workloads. The reason is that the raw QLC and Open-CAS are bounded by the large IU size (64KB) in QLC SSD and thus constantly receive considerable device-level write amplification (i.e., 64KB flush from 4KB write under cache miss). Therefore, when the size of writes becomes larger (i.e., 64KB random writes workloads), the write amplification alleviates.

Finally, we can see that CSAL-ZNS always outperforms CSAL-BLK by a small margin (e.g., see Figure 15(b)). This result resonates with our performance comparison in Figure 11, and it validates our previous assumption that zone emulation in CSAL-BLK incurs write amplification as GC and compaction streams can be merged by the FTL.

6.4 Evaluating CSAL Features

Compaction/GC efficiency. Now, we take a closer look at the I/O traffic to analyze the compaction and GC efficiency.

In Figure 16, we plot the read/write throughput—including VM and backend traffic—under 4KB uniform random write workloads. We can see that both Open-CAS and dm-zoned show much lower throughput of their reclaiming processes. This is because Open-CAS generates small random writes when flushing data from cache layer to the QLC SSD. Because of the limited cache space, not all writes can be merged in Open-CAS so that the size of writes in flushing are varying around 4KB to 64KB and thus user writes are constrained by small random writes of the QLC SSD (around 50MB/s). dm-zoned consistently migrates data between sequential zones of ZNS SSD and random zones of HP-SSD in the reclaiming processes. While this process generates around 700MB/s throughput combined by two types of reclaiming processes, the user writes are still blocked (less than 10MB/s) because of the zone-granularity mapping. Unlike the above two, CSAL

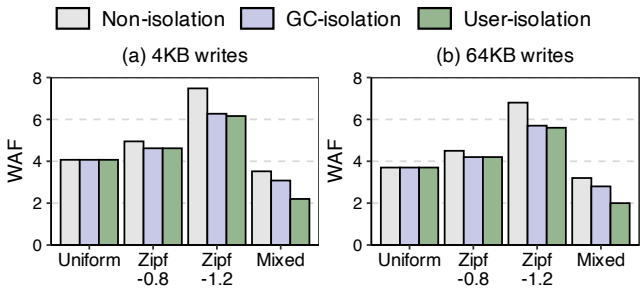

Figure 18. CSAL GC Write Amplification. *User-isolation means adding user-isolation with GC-isolation, which enables both GC-isolation and user-isolation.*

optimizes write traffic from HP-SSD to QLC SSD by aggregating 4KB user writes into large sequential buffer. Therefore, CSAL maximizes the data traffic in this process.

Figure 17 presents the backend throughput under 64KB uniform random write workloads. Open-CAS shows a high backend throughput (around 430MB/s) under 64KB write workloads, but this throughput is still lower than that of CSAL (see difference of flush in Open-CAS and CP in CSAL); dm-zoned still shows similar throughput as 4KB write workloads (around 700MB/s combined by two types of reclaiming processes). This is because Open-CAS generates 64KB random writes during cache flushing, which constrains user writes to the 64KB random writes of the QLC SSD. The difference between 4KB and 64KB writes is that Open-CAS can avoid device-level write amplification caused by 64KB indirection unit. For dm-zoned, it still has to migrate data between random zones and sequential zones to reclaim zone space, which results in user writes being blocked due to the zone-granularity mapping. For CSAL, there is not a significant difference between 4KB and 64KB write workloads due to the write aggregation during the compaction process.

VM and GC isolation. To evaluate the effectiveness of the two isolation techniques in CSAL, namely separating GC from compaction (GC-isolation) and grouping data by users

	Disk	P2L	L2P	Others	Total
Naive (s)	2621	-	-	1.64	2622.64
OPT-1 (s)	0.5	5.3	-	1.64	7.44
OPT-2 (s)	0.5	0.33	2.67	1.64	5.14

Table 4. CSAL Runtime Metadata Recovery Time. *OPT-1 means optimization 1 (P2L Table enabled); OPT-2 means optimization 2 (L2P checkpoint enabled).*

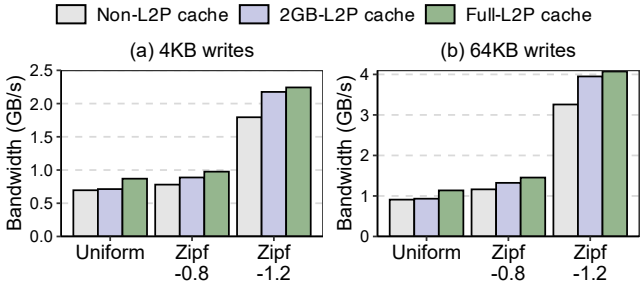

Figure 19. Performance Under Different L2P Cache.

(user-isolation), we allocate the 8 CSAL-ZNS disks (i.e., partitions) to 8 VMs respectively. We run workloads and then measure the WAF under no isolation, GC-isolation, and user-isolation. We run two sets of workloads. The first is to run the skewed workloads, including uniform, Zipf-0.8, and Zipf-1.2, on each VM. The second is to run the mixed workloads with four VMs on sequential writes and the other four on uniform random writes. For each workload, we further specify two write sizes, 4KB and 64KB.

Figure 18 shows the WAF results, calculated during the GC process. In skewed workloads, the more skewed the workloads are, the more benefits CSAL achieves with GC-isolation (gray bar vs. purple bar). Moreover, in mixed workloads, CSAL with user-isolation can further reduce write amplification.

Crash recovery. To measure the failover time, we reuse the same configuration and run uniform workloads as §6.2 in VMs. We wait until compaction and GC are invoked and then crash CSAL runtime process. Table 4 shows the runtime metadata recovery time under different policies proposed in §4.4. With optimization-1 (i.e., using the P2L table) and optimization-2 (i.e., L2P checkpoint), CSAL reduces recovery time from 2622.64 seconds of naive solution to 5.14 seconds. Note that Open-CAS and dm-zoned do not support dirty crash recovery, hence they are not included here.

L2P page cache. Now, we evaluate CSAL under writes workloads with different L2P page cache configurations, including non-cache (i.e., all L2P pages are read from disk), 2GB cache (i.e, the default configuration), and full-cache (i.e., all pages are cached after first loading).

For non-cache mode, we limit the DRAM used for L2P pages to 10KB, which means that almost all L2P pages should swap from the disk. In full-cache mode, we set the maximum DRAM used for L2P pages to 20GB (larger than all pages),

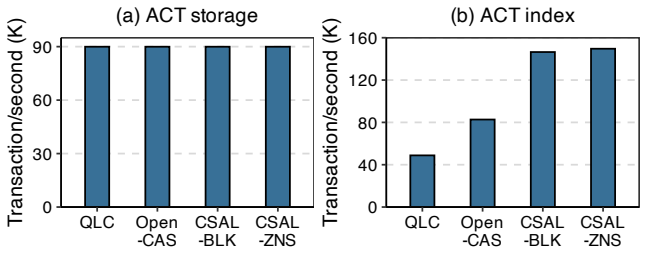

Figure 20. Aerospike Database Performance

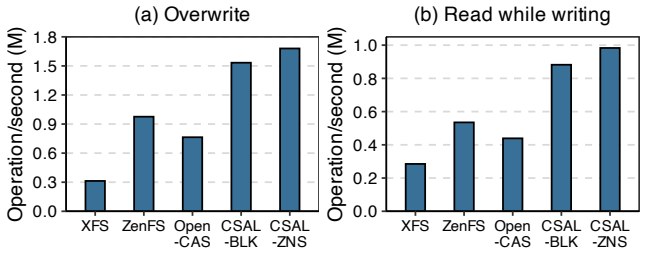

Figure 21. RocksDB Performance

meaning that all pages will be stored in DRAM after the first load from the disk. Figure 19 shows performance results under 4KB and 64KB write workloads. In uniform workloads, CSAL with full-cache shows a superior performance to both non-cache and 2GB cache modes, outperforming them by a large margin by a large margin (23% improvement vs. non-cache mode and 20% improvement vs. 2GB cache mode). When workloads become more skewed (i.e., Zipf-1.2), CSAL with 2GB cache gradually overtakes CSAL with full-cache, showing only a 3% lower performance under 4KB writes and a 2% lower performance under 64KB writes, respectively.

6.5 End-to-End Application Performance

We evaluate the candidates with two widely used databases: Aerospike [10] and RocksDB [9].

Aerospike database. Aerospike Certifying Toolkit (ACT) [1] provides a pair of programs for testing the performance for Aerospike Database data and index storage (i.e., *ACT_storage* and *ACT_index* respectively). ACT adopts a different benchmarking strategy where the ACT issues large volumes of transactions and only certifies ones finished within a latency threshold as valid. In Figure 20, for *ACT_storage*, all candidates achieve the same performance as raw QLC because this workload contains mostly large sequential writes (similar to the 128KB sequential writes in Figure 10(b)). Moreover, in *ACT_index* (a workload dominated by small and random writes), we notice that CSAL-based solutions outperform the rest by a large margin (up to 80.91% more transactions), resonating with the trend in Figure 10(a).

RocksDB. We prepare the experimental setups for RocksDB in the following way. First, to run RocksDB with TLC/QLC SSD only, we mount an XFS file system on the non-ZNS QLC SSD (denoted as XFS) and a ZenFS on the ZNS TLC

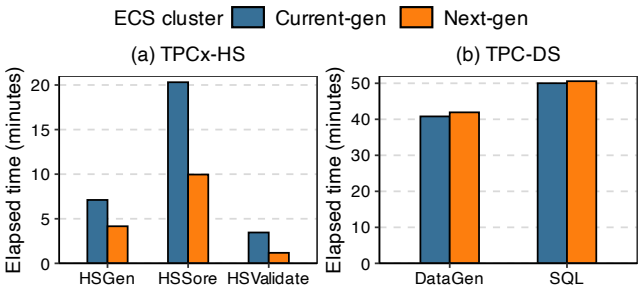

Figure 22. Big Data Evaluation under Real Production. *Elapsed time (lower is better) represents the time used to process data with fixed size (e.g., 3TB in our case) under different types of tasks. This reflects the efficiency including compute and storage of ECS nodes.*

SSD (denoted as ZenFS). Second, we use SPDK and BlobFS to support RocksDB over Open-CAS, CSAL-BLK, and CSAL-ZNS. Note that we do not include dm-zoned in the results due to its negligible performance. Third, we use the *fill_random* workloads from db_bench [2] to warm up the databases. We disable the internal cache and enforce RocksDB to run in direct mode. Finally, we evaluate two workloads from db_bench (i.e., *over_write* and *read_while_writing*), each with one billion keys. The key size is 20 bytes and the value size is 400 bytes; the block size of RockDB is set as 4KB.

Figure 21 shows the evaluation results of using db_bench. In both benchmarks, CSAL-ZNS and CSAL-BLK rank first and second. Specifically, CSAL-ZNS achieves a 1.72× and 2.20× speedup against ZenFS and Open-CAS in the *overwrite* workloads and can be 1.82× and 2.24× faster than ZenFS and Open-CAS in *read_while_writing* workloads. We can see that, by reducing two levels of write amplification, CSAL can provide end-to-end performance improvements for data-intensive applications.

6.6 Resource and Cost Analysis

For CSAL and Open-CAS, one core is used for polling-mode processing. Meanwhile, the CPU utilization of dm-zoned is around 120% (i.e., 1.2 core) due to frequent data migration during the reclaiming process. For memory usage, CSAL uses 2.12GB (2GB for L2P page cache and around 120MB for other runtime metadata); Open-CAS uses 1GB memory (0.125% of cache capacity) for its metadata under the 64KB cache line configuration. The raw QLC and dm-zoned use less than 20MB memory.

6.7 Field Deployment Performance

As of now, we have deployed CSAL on thousands of ECS nodes in Alibaba Cloud. Compared to last-gen HDD-based local disks (24×2TB HDDs with a 48-core Xeon Cascade CPU), CSAL-ready servers (an 800GB HP-SSD and a 15.36TB QLC SSD with a 64-core Xeon Ice Lake CPU) can host twice more instances while achieving the same SLOs. We further set up a current-gen and a next-gen HDFS clusters where each employs three same-level ECS instances. From Figure 22, we can see that the CSAL cluster improves 70.91%, 103.92%, and 193.22% efficiency over HSGen, HSSort, and HSValidate processes in TPCx-HS, respectively. Moreover, in TPC-DS, both clusters deliver similar performance with the CSAL-based instances leading by a small margin (around 1.81%). This indicates that the CSAL-based ECS instances can deliver similar or even better performance than HDD-based ones.

7 Related Works

ZNS-based solutions. A large group of works targets the optimization of the ZNS interface [17] on the block layer, file systems [33, 35, 49, 57], and the Key-Value stores [36–38, 45, 47, 54, 56, 62]. CSAL is different from the above because, instead of optimizing certain layers, our work focuses on building a ZNS-based end-to-end stack for cloud local disks.

Hybrid storage systems. Another spate of related work is the hybrid storage systems that exploit the performance of high-speed devices (e.g., DRAM) and the space of large-capacity drives [26, 29, 32, 39, 40, 50]. Our work differs from the aforementioned works (mostly TLC-based) as CSAL aims at utilizing the large capacities of enterprise-level QLC SSDs while avoiding subsequent write amplification by maintaining a two-level L2P table for fine-grained data accessing.

FTL optimization. Other works focus on the Flash Translation Layer (FTL) optimizations [27, 46]. They address performance overhead from device-level by redesigning data placement strategies. The authors of [59] also give detailed analysis of SSD-based workload characteristics and their performance implications. Our work uses part of key insights learned from above works when we design FTL, but we focus on cloud-based local disks with high-density SSDs.

8 Conclusion

In this paper, we explore a pressing topic, the future of local disks for the cloud. Through various unsuccessful attempts, we have identified the key factors in enabling high-density SSDs as local disks. With these lessons learned, we propose our design CSAL and extensively evaluate its performance. Currently, we have deployed CSAL on large-scale clusters and released the source code.

Acknowledgments

We sincerely thank all the anonymous reviewers and steering committee for their insightful comments to improve the paper. We also thank engineers from Alibaba Cloud and Solidigm for their efforts to make CSAL open-source and deployed in a real production system. This research was supported by NSFC (62102424) and the Alibaba Research Fellow (ARF) program. The authors would also like to thank Amber Bi, Keely Xu, and Livia Liu for their support.

References

[1] [n. d.]. Aerospike Certification Tool (ACT). https://github.com/aerospike/act

[2] [n. d.]. Benchmarking tools. https://github.com/facebook/rocksdb/wiki/Benchmarking-tools

[3] [n. d.]. Build Ultra High-Performance Storage Applications with the Storage Performance Development Kit. https://spdk.io/

[4] [n. d.]. dm-zoned. https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-zoned.html

[5] [n. d.]. FIO. https://fio.readthedocs.io/en/latest/

[6] [n. d.]. Intel Optane™ SSD P5800X Series. https://www.intel.com/content/www/us/en/products/docs/memory-storage/solid-state-drives/data-center-ssds/optane-ssd-p5800x-p5801x-brief.html

[7] [n. d.]. Internal Drives Ultrastar DC ZN540 from Western Digital. https://www.westerndigital.com/en-il/products/internal-drives/ultrastar-dc-zn540-nvme-ssd#0TS2094

[8] [n. d.]. Open Cache Acceleration Software. https://open-cas.github.io/index.html

[9] [n. d.]. A persistent key-value store for fast storage environments. https://rocksdb.org

[10] [n. d.]. The scalable, multi-model real-time database. https://aerospike.com/

[11] 2021. SPDK v21.04: ZNS NVMe bdev, PMR, ADQ initiator, RPM. https://github.com/spdk/spdk/releases/tag/v21.04

[12] AWS. 2022. Amazon EC2 D3 & D3en Instances - Run dense storage workloads with the highest capacity local storage in the cloud. https://aws.amazon.com/ec2/instance-types/d3/

[13] AWS. 2022. Amazon EC2 I3 Instances - Storage optimized for high transaction workloads. https://aws.amazon.com/ec2/instance-types/i3

[14] AWS. 2022. Amazon EC2 I3ne Instances - Dense SSD storage instances for data-intensive workloads. https://aws.amazon.com/ec2/instance-types/i3en

[15] Microsoft Azure. 2022. Lasv3-series - Azure Virtual Machines. https://learn.microsoft.com/en-us/azure/virtual-machines/lasv3-series

[16] Microsoft Azure. 2022. Lsv3-series - Azure Virtual Machines. https://learn.microsoft.com/en-us/azure/virtual-machines/lsv3-series

[17] 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 *Proceedings of the 2021 USENIX Annual Technical Conference (ATC)*.

[18] Alibaba Cloud. 2022. D2c, compute-intensive big data instance family. https://www.alibabacloud.com/help/en/elastic-compute-service/latest/instance-families-with-local-ssds#section-ogb-jlc-y4v

[19] Alibaba Cloud. 2022. D2s, storage-intensive big data instance family. https://www.alibabacloud.com/help/en/elastic-compute-service/latest/big-data-instance-families#section-eum-nil-2ui

[20] Alibaba Cloud. 2022. I3, instance family with local SSDs. https://www.alibabacloud.com/help/en/elastic-compute-service/latest/instance-families-with-local-ssds#section-ogb-jlc-y4v

[21] Intel Corporation. 2021. Achieving Optimal Performance & Endurance on Coarse Indirection Unit SSDs. https://www.colfax-intl.com/downloads/intel-achieving-optimal-perf-iu-ssds.pdf

[22] Tom Coughlin. 2021. Seagate: High Capacity HDDs Have Better TCO Than SSDs. https://www.forbes.com/sites/tomcoughlin/2021/02/25/seagate-high-capacity-hdds-have-better-tco-than-ssds/

[23] Ian Cutress. 2020. Intel Launches Cooper Lake: 3rd Generation Xeon Scalable for 4P/8P Servers. https://www.anandtech.com/show/15862/intel-launches-cooper-lake-3rd-generation-xeon-scalable-for-4p8p-servers

[24] Western Digital. 2022. Western Digital Extends HDD Technology and Areal Density Leadership Across Smart Video, Network Attached Storage (NAS) and IT/Data Center Channel Segments. https://www.westerndigital.com/company/newsroom/press-releases/2022/2022-07-19-western-digital-extends-hdd-technology-and-areal-density-leadership

[25] NVM Express. 2023. Features for Error Reporting, SMART, Log Pages, Failures and management capabilities in NVMe Architectures. https://nvmexpress.org/resource/features-for-error-reporting-smart-log-pages-failures-and-management-capabilities-in-nvme-architectures/

[26] Akira Goda. 2020. 3-D NAND technology achievements and future scaling perspectives. *IEEE Transactions on Electron Devices (TED)* 67, 4 (2020), 1373–1381.

[27] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. *Acm Sigplan Notices* 44, 3 (2009), 229–240.

[28] Jonmichael Hands. 2022. SupermeRAID and Solidigm D5-P5316 QLC NVMe: Case Study. https://www.graidtech.com/case-study-supremeraid-solidigm-d5-p5316-qlc-nvme/

[29] Soojun Im and Dongkun Shin. 2010. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. *Journal of Systems Architecture* 56, 12 (2010), 641–653.

[30] Shehbaz Jaffer, Kaveh Mahdaviani, and Bianca Schroeder. 2022. Improving the Reliability of Next Generation SSDs using WOM-v Codes. In *Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST)*.

[31] Charles P. Jefferies. 2022. Seagate Exos X20 20TB Enterprise HDD Review. https://www.storagereview.com/review/seagate-exos-x20-20tb-enterprise-hdd-review

[32] Pranav Kalavade. 2020. 4 bits/cell 96 Layer Floating Gate 3D NAND with CMOS under Array Technology and SSDs. In *Proceedings of the 2020 IEEE International Memory Workshop (IMW)*.

[33] Juwon Kim, Minsu Kim, Muhammad Danish Tehseen, Joontaek Oh, and YouJip Won. 2022. IPLFS: Log-Structured File System without Garbage Collection. In *Proceedings of the 2022 USENIX Annual Technical Conference (ATC)*.

[34] KIOXIA. 2022. XL-FLASH, Storage Class Memory (SCM). https://www.kioxia.com/en-jp/business/memory/xlflash.html

[35] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. 2015. F2FS: A New File System for Flash Storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*.

[36] 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 (HotStorage)*.

[37] Cheng Li, Hao Chen, Chaoyi Ruan, Xiaosong Ma, and Yinlong Xu. 2021. Leveraging NVMe SSDs for Building a Fast, Cost-effective, LSM-tree-based KV Store. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 1–29.

[38] Jinhong Li, Qiuping Wang, and Patrick PC Lee. 2022. Efficient LSM-Tree Key-Value Data Management on Hybrid SSD/HDD Zoned Storage. *arXiv preprint arXiv:2205.11753* (2022).

[39] Yan Li. 2020. 3D NAND Memory and Its Application in Solid-State Drives: Architecture, Reliability, Flash Management Techniques, and Current Trends. *IEEE Solid-State Circuits Magazine* 12, 4 (2020), 56–65.

[40] Shuwen Liang, Zhi Qiao, Sihai Tang, Jacob Hochstetler, Song Fu, Weisong Shi, and Hsing-Bung Chen. 2019. An Empirical Study of Quad-Level Cell (QLC) Nand Flash SSDs for Big Data Applications. In *Proceedings of the 2019 IEEE International Conference on Big Data (BigData)*.

[41] Linux. 2022. AIO - POSIX asynchronous I/O overview. https://linux.die.net/man/7/aio

[42] Zhiye Liu. 2022. Intel Sapphire Rapids Workstation Specs Leaked: Up To 56 Cores, 350W TDP. https://www.tomshardware.com/news/intel-xeon-sapphire-rapids-ws-specs-leaked-up-to-56-cores-350w-tdp

[43] Zhiye Liu. 2022. Intel's Xeon Emerald Rapids CPUs Could Wield Up To 64 Cores. https://www.tomshardware.com/news/intels-xeon-emerald-rapids-cpus-could-wield-up-to-64-cores
[44] Chris Mellor. 2019. WD and Tosh talk up penta-level cell flash. https://blocksandfiles.com/2019/08/07/penta-level-cell-flash/
[45] Gijun Oh, Junseok Yang, and Sungyong Ahn. 2021. Efficient Key-Value Data Placement for ZNS SSD. *Applied Sciences* 11, 24 (2021), 11842.
[46] Yubiao Pan, Yongkun Li, Huizhen Zhang, Hao Chen, and Mingwei Lin. 2020. GFTL: Group-level mapping in flash translation layer to provide efficient address translation for NAND flash-based SSDs. *IEEE Transactions on Consumer Electronics* 66, 3 (2020), 242–250.
[47] Madhurima Ray, Krishna Kant, Peng Li, and Sanjeev Trika. 2020. FlashKey: A High-Performance Flash Friendly Key-value Store. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.
[48] Samsung. 2022. Samsung Z-SSD SZ985. https://semiconductor.samsung.com/newsroom/tech-blog/samsung-z-ssd-sz985/
[49] Margo Seltzer, Keith Bostic, Marshall McKusick, and Carl Staelin. 1993. An Implementation of a Log-Structured File System for UNIX. In *USENIX Winter*.
[50] Kent Smith. 2019. Using QLC SSDs to Improve Cost/Performance Tradeoffs for Warm Data. In *Proceedings of the 2019 Flash Memory Summit*.
[51] Lyle Smith. 2021. Intel P5316 SSD Review (30.72TB). https://www.storagereview.com/review/intel-p5316-ssd-review-30-72tb
[52] Solidigm. 2022. Solidigm Demonstrates World's First Penta-Level Cell SSD at Flash Memory Summit. http://news.solidigm.com/en-WW/217006-solidigm-demonstrates-world-s-first-penta-level-cell-ssd-at-flash-memory-summit
[53] Solidigm. 2023. Introducing the Solidigm D7-P5810 – an ultra-fast SLC SSD for write-intensive workloads. https://news.solidigm.com/en-WW/230095-introducing-the-solidigm-d7-p5810-an-ultra-fast-slc-ssd-for-write-intensive-workloads
[54] Yuzhe Tang, Arun Iyengar, Wei Tan, Liana Fong, Ling Liu, and Balaji Palanisamy. 2015. Deferred Lightweight Indexing for Log-structured Key-value Stores. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*.
[55] Tiffany Trader. 2021. Intel Launches 10nm 'Ice Lake' Datacenter CPU with Up to 40 Cores. https://www.hpcwire.com/2021/04/06/intel-launches-10nm-ice-lake-datacenter-cpu-with-up-to-40-cores/
[56] Qiuping Wang, Jinhong Li, Patrick PC Lee, Tao Ouyang, Chao Shi, and Lilong Huang. 2022. Separating Data via Block Invalidation Time Inference for Write Amplification Reduction in {Log-Structured} Storage. In *Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST)*.
[57] Jian Xu and Steven Swanson. 2016. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*.
[58] Shuai Xue, Shang Zhao, Quan Chen, Gang Deng, Zheng Liu, Jie Zhang, Zhuo Song, Tao Ma, Yong Yang, Yanbo Zhou, Keqiang Niu, Sijie Sun, and Minyi Guo. 2020. Spool: Reliable Virtualized NVMe Storage Pool in Public Cloud Infrastructure. In *Proceedings of the 2020 USENIX Annual Technical Conference (ATC)*.
[59] Gala Yadgar, Moshe Gabel, Shehbaz Jaffer, and Bianca Schroeder. 2021. SSD-based workload characteristics and their performance implications. *ACM Transactions on Storage (TOS)* 17, 1 (2021), 1–26.
[60] Ziye Yang, James R Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E Paul. 2017. Spdk: A Development Kit to Build High Performance Storage Applications. In *Proceedings of the 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*.
[61] Ziye Yang, Changpeng Liu, Yanbo Zhou, Xiaodong Liu, and Gang Cao. 2018. SPDK Vhost-NVMe: Accelerating I/Os in Virtual Machines on NVMe SSDs via User Space Vhost Target. In *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*. IEEE, 67–76.
[62] Ting Yao, Jiguang Wan, Ping Huang, Yiwen Zhang, Zhiwen Liu, Changsheng Xie, and Xubin He. 2019. GearDB: A GC-free Key-value Store on HM-SMR Drives with Gear Compaction. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST)*.

A Artifact Appendix

A.1 Abstract

We provide the artifact for the EuroSys 2024 paper "CSAL: the Next-Gen Local Disks for the Cloud". The artifact includes how to get project source code, rebuild the project from scratch and reproduce the main experimental results of the paper. Note that CSAL has been upstreamed into SPDK after release v22.09: https://github.com/spdk/spdk under BSD 3-clause license.

All the experiments should be conducted on real hardware platform with high-performance Storage Class Memory (SCM) SSDs and high-density QLC SSDs. At least 3GB DRAM is needed for CSAL runtime.

A.2 Artifact Check-List

- Compilation: GCC 4.9; Python 3.12.
- Run-time environment: running SPDK with CSAL requires root access.
- Hardware: enterprise-grade NVMe SCM SSDs and QLC SSDs. We recommend using Intel P5800X [6] or Solidigm D7-P5810 [53] SCM SSDs for cache, and Intel P5316 [51] QLC SSDs for large-capacity layer.
- Memory requirement: at least 3GB memory reserved for CSAL runtime. You also need more memory for virtual machine (at least 16GB is recommended).
- Artifact guide: https://github.com/ARDICS/CSAL_AE
- Source code: https://github.com/spdk/spdk. CSAL implementation is under "spdk/lib/ftl".
- Code licenses: BSD 3-clause.
- Archived: https://doi.org/10.5281/zenodo.10086260
- Time needed to build CSAL: less than 30 minutes.
- Time needed to complete experiments: around 50 hours.

A.3 Description

How to access. Get the source code from SPDK GitHub repository at https://github.com/spdk/spdk and then use the release v22.09 or later version.

How to build. Detailed user guide about how to build and reproduce the experiments is provided in our artifact repository at https://github.com/ARDICS/CSAL_AE. We also provide scripts for reproducing our experiments. Note that the scripts only cover raw device performance (see Section §6.2).

A.4 Other

We will continue to maintain the CSAL project in SPDK, and regularly release new features and optimizations. For future activities, please pay attention to SPDK community.