Append is Near: Log-based Data Management on ZNS SSDs

Devashish R. Purandare
devashish@ucsc.edu
University of California, Santa Cruz
Santa Cruz, CA, USA

Peter Wilcox
pcwilcox@ucsc.edu
University of California, Santa Cruz
Santa Cruz, CA, USA

Heiner Litz
hlitz@ucsc.edu
University of California, Santa Cruz
Santa Cruz, CA, USA

Shel Finkelstein
shel@ucsc.edu
University of California, Santa Cruz
Santa Cruz, CA, USA

ABSTRACT

Log-based data management systems use storage as if it were an append-only medium, transforming random writes into sequential writes, which delivers significant benefits when logs are persisted on hard disks. Although solid-state drives (SSDs) offer improved random write capabilities, sequential writes continue to be advantageous due to locality and space efficiency. However, the inherent properties of flash-based SSDs induce major disadvantages when used with a random write block interface, causing write amplification, uneven wear, log stacking, and garbage collection overheads. To eliminate these disadvantages, Zoned Namespace (ZNS) SSDs have recently been introduced. They offer increased capacity, reduced write amplification, and open up data placement and garbage collection to the host through zones, which have sequential-write semantics and must be explicitly reset.

We explain how the new ZNS Zone Append primitive, which supports pushing fine-grained data placement onto the device, along with our proposal for "Group Append", which enables sub-block sized appends, could benefit log-structured data management systems. We explore advantages of ZNS SSDs with Zone Append, Group Append, and computational storage in four log-based data management areas: (i) log-based file systems, (ii) LSM trees such as RocksDB, (iii) database systems, and (iv) event logs/shared logs. Furthermore, we propose research directions for each of these data management systems using ZNS SSDs.

1 INTRODUCTION

Data management systems have used the same block storage interface for decades, while assuming that the underlying storage devices have similar characteristics. Recently, both interfaces and storage device characteristics have changed, becoming more capable and computationally intelligent by leveraging smart controllers, accelerators, and disaggregation[38, 64]. As shown by Lerner [42], there exists a co-design continuum across applications, storage systems, programmable devices, and workloads. In particular, ZNS SSDs offer a fertile ground for innovation, unlocking the potential of SSDs, and offering advantages over hard disks, and traditional SSDs.

 12th Annual Conference on Innovative Data Systems Research (CIDR '22). January 9-12, 2022, Chaminade, USA..

The conventional block interface maps a range of Logical Block Addresses (LBAs) onto a fixed address space and allows random reads and writes. ZNS SSDs [62] partition the storage device into a set of equally-sized zones supporting append-only writes. While this represents a limitation over traditional block-based SSDs, it offers greater control to the host over internal SSD operations such as data placement and erase. For a ZNS SSD, reads are the same as for a conventional SSDs; both sequential and random reads are supported. Data becomes immutable whenever appended to the tail of a zone, and entire zones can be made immutable either by an explicit *finish* command or automatically when they fill up. ZNS requires the host to perform resets at zone granularity, eliminating on-device garbage collection and device-side media over-provisioning.

The general advantages of ZNS SSDs have been described at length in literature [18, 57]. We focus on advantages that are of particular relevance for systems architecture:

- **Data Placement:** ZNS SSDs provide greater control over data placement. For example, the device can align the media boundaries to zones and eliminate the mismatch between write size and block size of the random-write block interface when using flash-based SSDs.
- **Data Immutability:** Immutability of zones greatly simplifies [14, 33, 43, 44] operations such as version management, replication, and reorganization.
- **Hardware Alignment:** ZNS SSDs allow the host to align data with fixed hardware boundaries. This approach eliminates write amplification caused by performing garbage collection at multiple levels in the storage stack [65].
- **Optimizing Log Operations:** Host applications can use the Zone Append primitive to simplify the bookkeeping typically needed to support log-structured [54] applications.

With ZNS SSDs, we have an opportunity to align log-based systems with the log-based interface exposed by the hardware. Elimination of random writes and host-controlled garbage collection create new challenges and opportunities for data and metadata management. The ZNS append-only interface may not be appropriate for all aspects of data management. Additionally, ZNS SSDs have a resource limit on the number of zones that can be "active" (available for writes or appends), and managing these resource constraints is an additional responsibility of the host. We will discuss ways to address these challenges later in the paper.

Writing data to a log may be gated by the cost of determining where the log ends (the log tail), since new appends must be written to the tail of the log. Implementations may have a single writer, or

a single source of truth about the current tail, or a single means of assigning tail locations for append as in Hyder [16]. Even when that singleton is not a bottleneck (again as in Hyder), a true append operation is simpler and potentially more efficient for some applications. Group Commit [32] improves throughput both by reducing the number of log forces (possibly increasing latency slightly) and decreasing wasted space in log blocks (due to partially-filled log blocks). ZNS introduces a new primitive, Zone Append, which implements these capabilities. We propose extending this capability further in §2.

Contributions and Paper Overview: Our paper explores how data management systems can advantageously exploit ZNS operations, and proposes evolutionary and revolutionary approaches for such exploitation. Through §4–7 we talk about how commonly used log-based data management systems, including file systems, LSM trees, databases, and event/shared logs can evolve to benefit from append-only zoned storage. We conclude with some recommendations for general research directions and a conclusion that we agree with [42] that data management systems need to change significantly to leverage ZNS Storage and that collaborative research and experimentation are necessary to determine successful directions for change. We motivate the need for *Group Append,* a primitive similar to Zone Append which can work at byte-granularity.

2 ZONE APPEND AND GROUP APPEND

2.1 Zone Append

The ZNS Zone Append command enables multiple loggers to write to the tail of the log without knowing the exact location of the tail. The host indicates which zone to append to, offloading the placement and the ordering to the controller, which responds with the assigned LBA once the write has been completed. Zone Append delegates finer-grained data placement to the controller, and helps reduce contention on the write pointer. Zone Append is an optional feature in the ZNS specification, but our discussion focuses on advantages of this approach. There is an alternative in Zone Random Write Area (ZRWA) which we briefly discuss in §8.6.

2.2 Group Append

We propose a finer-grained alternative to Zone Append called "Group Append" which allows appends of data that is smaller (or possibly larger) than a block, offloading data buffering to the controller. In practice, an *append-bytes* library call can be used to write to an append-only buffer that's on the SSD controller where "log zones" are located. When the append-only buffer outgrows the block size, the controller can append it to the tail of the log. Conventional writes could also be used, but would introduce the same challenges that we've previously described for a single writer, since (unlike for Zone Append) the LBA is an input to the write command. Alternatively, the tail of the log could be maintained in the controller, as proposed in Hyder [52]. Although approaches similar to Group Append can be implemented on conventional SSDs, they would not achieve all the benefits that ZNS SSDs provide, as blocks will be interleaved across the SSD log structure than discrete zones.

Data loss can be avoided by ensuring the persistence of the controller memory buffer. The *append-bytes* library call we propose returns the starting LBA and byte offset where the host data will be appended within the specified zone. The controller receives and responds to a series of *append-bytes* requests; it can compute and return the correct LBA and byte offset for each request in the series without waiting for the buffer to fill up. When blocks are full, the controller appends them to the tail of the current log zone. An append that fails will be retried until it either is successful or results in a write error. Group Append avoids space amplification due to partially full blocks without increasing write latency because the controller is able to respond quickly to each *append-bytes* request.

3 RELATED WORK

Stavrinos et al. [57] pointed out the possibilities with ZNS SSDs and their potential to replace block-based SSDs. We present specific design decisions required across log-based systems to achieve this goal. Bjørling et al. [18] demonstrated the benefits of ZNS on log-based filesystems and RocksDB. We explore further changes required to the internal operations within these systems, like compaction and checkpoints, to utilize ZNS to its full potential. Lerner et al. [42] explore SSD and software co-design.

Our recommendations, such as Group Append, present how such co-design can benefit both the SSD and the data management systems. Maheshwari [47] proposes enabling variable-sized sub-block writes called "rocks", which would allow ZNS SSDs to perform writes as small as 16B in a special namespace. While Group Append would work with this approach, it does not need a special namespace and could be achieved with a single command on top of the current NVMe framework. Group Append would allow an application to issue sub-block appends to the tail, and the controller will group them into a block. Since the zone with these appends is dedicated to a specific log, the base LBA or zone number will be sufficient for a lookup.

Key-Value SSDs (KV-SSDs) [55, 63] adapt the internal layout of SSDs to an LSM Tree interface, allowing key-value operations from the host, and offloading operations such as compaction to the device. While these special-purpose devices avoid the log-on-log issue and offer better throughput and lower latency, they are use-case specific. Modern data stores like RocksDB involve several operations beyond K-V operations such as write-ahead logging, replication, indexes, and metadata management, which do not use a Key-Value interface and hence are not supported by a KV-SSD.

Open-Channel SSDs (OC-SSDs) [30, 45, 51] are precursors to ZNS SSDs, which provide host-side FTLs greater control over flash internals. On OC-SSDs the host is responsible for low-level operations such as maintaining a map of blocks, wear-leveling and buffering. Systems such as LightNVM [19] perform operations in a sequential-write only "Chunk" interface which is similar to the zone interface on ZNS SSDs. ZNS offers a more generalized and less complex approach to this abstraction, handling SSD internal operations like wear-leveling, block metadata and buffering on the device, along with finer granularity placement within a zone. The host gets access to the zone abstraction, as well as the ability to do garbage collection without maintaining a host-side FTL.

Multi-Stream SSDs [36, 37] (MS-SSDs) offer abstractions for data to be split into multiple streams, based on the lifetime of the data,

Work by \ SSD Type	Traditional SSD	Open Channel SSD	ZNS SSD
Host	Block Interface	Data Placement Garbage Collection Wear Leveling FTL Operations	Data Placement (Coarse) Garbage Collection
Device	Data Placement Garbage Collection Wear Leveling FTL Operations Buffers Error Correction	Buffers Error Correction	Data Placement (Fine) Wear Leveling FTL Operations Buffers Error Correction

Table 1: Comparison of different types of SSDs with the traditional SSD interface.

which the device stores in separate physical blocks. Our recommendations for data management with multiple zones is similar to this approach, with certain important distinctions. MS-SSDs allow random writes within streams, and commit streams at the block level, as opposed to zones that are sequential-write only and a granularity of hundreds of megabytes. In MS-SSDs the controller is responsible for garbage collection which occurs at erase block level, while in ZNS SSDs GC is the responsibility of the host and performed at the zone level, which can be hundreds of megabytes as opposed to erase blocks which are a few megabytes in size.

These devices take different approaches to reach similar goals, where Open Channel SSDs push the FTL and block management to the host, MS-SSDs and KV-SSDs allow the device greater control over data layout while adding constraints on the application or the interface to provide information about the stored data. As seen in table 1, ZNS SSDs offer a compromise between traditional SSDs, and Open-Channel SSDs. The host gets greater control over data placement, data organization, and garbage collection, while the controller performs finer-grained placement along with low-level device operations such as wear leveling, error correction, and bad block management.

Many data management systems and papers have addressed design and implementation using SSDs. Some of that work directly relates to the approaches proposed in this paper. But papers written before ZNS Storage emerged did not (and could not) consider the benefits of ZNS storage. Hyder [15, 16, 52] enables efficient database server scale-out, with arbitration required only for log-append to a network-attached log. Zone Append and Group Append provide similar capabilities with additional advantages like better space utilization. Do [26] moved log-structuring and garbage collection from an open-channel raw flash SSD to the host. They also suggest the possibility of using NVRAM or DRAM with power backup to improve latency. Chakraborttii [21] utilizes multiple open blocks as provided by ZNS and death-time prediction to reduce GC overheads. SaS: SSD as SQL Database System [50] proposes moving all database functionality to a computational SSD, experimenting with an initial SQLite prototype built on OpenSSD. However, SaS does not utilize ZNS or Group Append; it suggests using of NVM to improve logging performance.

4 LOG-BASED FILE SYSTEMS

In this section, we analyze the traditional approaches used by log-based file systems (LFS) and describe why they are a natural fit for ZNS SSDs.[1] An LFS stores data using *data segments* in the *main region* of persistent storage, while associated metadata (used to locate data blocks within each segment) is persisted in reserved *checkpoint regions* (CRs).

4.1 Evolution to ZNS SSDs

4.1.1 ***Hardware Alignment:*** Log-based file systems are a natural fit for ZNS SSDs because their characteristics align closely with the platform's constraints. The LFS can designate each zone as either checkpoint region or main region. Data segments are written atomically, so it is simple to align their size with zone boundaries, while checkpoints are updated using an append-only log. By aligning the size of each region with zone boundaries the LFS can activate a zone only when writing a new data segment to the drive or when appending new checkpoint data. This allows the LFS to minimize the number of active zones required for file system overhead.

4.1.2 ***Segment Construction with Group Append:*** A conventional LFS assembles the data segment on the host and then writes it to the device. We propose instead assembling data segments at the controller. The host would use Group Append to write data, enabling writes smaller than one block. The controller would maintain a write buffer for each open zone, and incoming data would be appended to the tail. Once the controller has assembled a data segment, it can be written to the zone. The write request can be returned immediately to the host without waiting for the segment to be written to the zone because the controller can precompute the eventual assigned LBA. Data loss can be avoided by ensuring the persistence of the buffer (using NVRAM for example). This approach allows the host to offload data segment construction to a storage device capable of performing this task.

4.2 Research Directions

Log-based file systems are a great fit for ZNS SSDs, and we see the main avenues for research in the following areas:

[1] For brevity, we focus our discussion on POSIX-based file systems. We describe general characteristics of an LFS; for specific implementations, see [18, 41, 53].

- **Data Layout:** The host can co-locate data with similar lifetime into zones, reducing the overall host-side garbage collection overhead. However, if the common data management unit used by the host application does not align with the size of the zone on the device, this may introduce additional I/Os. Recent work proposes informing the host about underlying storage architecture to better align data with architectural boundaries [6]. Can we incorporate this approach, using data segments of smaller than zone size and leveraging the host context to better facilitate data placement and thereby reduce the overall garbage collection?
- **Disaggregated Metadata:** Storing file system metadata on ZNS SSDs introduce new challenges and complexities. An alternative approach is to utilize disaggregated storage and place metadata in conventional namespaces that support random writes. Do the performance, reliability, and complexity tradeoffs render this approach worthwhile?

5 LSM TREES AND ROCKSDB

Log-Structured Merge trees [49] (LSM trees) store data in key-value pairs in a sequential log-based structure called Sorted String Tables (SSTables). Looking through the lens of the RUM Conjecture [9], LSM trees are optimized to minimize update and memory overhead, at the cost of an increased read overhead. The sequential and immutable nature of SSTables ensures similar lifetimes for adjacent data. Further, the ability to easily partition an LSM tree in fixed-size chunks render them a natural fit for ZNS. Variants of LSM trees are present in several widely adopted data management systems such as Cassandra [40], RocksDB [2], and BigTable [22]. In this section, we will focus on the RocksDB implementation of an LSM tree.

RocksDB:. Writes to RocksDB are written to an in-memory structure called a Memtable and are simultaneously logged to a Write-Ahead Log (WAL) on persistent storage. Depending on the configuration, writes may also be synced to an SSTable at Level 0 (L_0). When these Memtables fill up, RocksDB flushes them to persistent media. RocksDB maintains SSTables on persistent media in the form of levels labeled L_0, L_1, ...As SSTables at each level fill up, RocksDB uses a process called *Compaction* to merge multiple SSTables and push them to the next level. RocksDB depends on a storage backend like a filesystem to manage SSTable files, WAL files, and other metadata.

Systems such as ZenFS [18] and ZoneFS [28] are part of the ongoing effort to adapt the RocksDB filesystem backend to use ZNS SSDs. However, adapting the architecture of RocksDB to use ZNS SSDs will allow us to improve garbage collection, reduce write amplification caused by logging, and optimize compaction to reduce data movement and compute.

5.1 Evolution to ZNS SSDs

5.1.1 ***Updating SSTable layout:*** SSTables and zones are a natural fit, as both guarantee sequential write interface, immutability when filled up, and a similar lifetime for data within. However, SSTable sizes can vary across levels, and zone sizes can vary across devices. Aligning SSTables with zone boundaries is essential, as we avoid the write amplification caused by the Log-on-Log problem [65]. Mapping zones to SSTables can be challenging in practice when their sizes differ.

(1) Mapping multiple zones to a single SSTable could allow the host to distribute these SSTables across devices, and the fixed-sized immutable chunks of data could simplify replication. However, distributing appends across multiple zones becomes a challenge as the ordering of appends is left to the controller.
(2) Mapping multiple SSTables to a single zone could increase data movement if the SSTables are compacted at different times. To ensure space can be freed up with minimal data movement, the host must ensure that SSTables stored in a zone have similar data lifetimes and will be compacted together.

Inter-Zone Dependencies: If the data structures, like SSTables are larger than the zone size, they will be distributed across multiple zones. This can cause dependencies across zones, for instance, in the RocksDB SSTable layout, as shown in Fig. 1, the last zone contains the metadata, accessing data on other zones may depend on this metadata. This will increase access latency as many operations will be reading from the same zone. This could be addressed by picking a different compaction scheme, or by storing metadata in a known location, possibly on a different device.

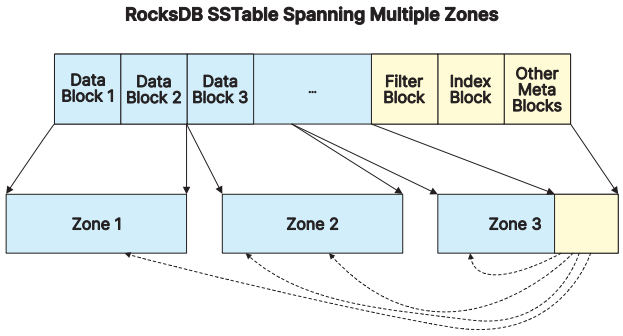

Figure 1: Simply distributing a large SSTable across zones will cause inter-zone dependencies, as access to data as well as metadata will require reading from the last zone of the series where all the metadata and filters are located.

The ability to co-locate data with the same lifetime in a contiguous zone is a powerful tool that conventional SSDs cannot provide. RocksDB can provide Write Hints (RWH) [10] about the lifetime of data being written. SSTables with similar lifetimes could be mapped to the same zone, which would minimize data movement upon data compaction, and is implemented within ZenFS. However, while these hints are supported, techniques to predict data lifetime and use that effectively are a research topic. Column families should be stored in separate zones, ensuring they benefit from the multi-tenancy features of zones.

5.1.2 ***Updates to Write-Ahead Logging:*** Write-Ahead Logging in RocksDB can cause write amplification greater than the write

amplification induced by the LSM tree [1]. Every write to the log causes two writes, one for writing the data and one for updating file metadata. With the minimum write granularity being 4KB, this results in 8KB of writes for every incoming log entry. RocksDB has two mechanisms to address this. It allows group commits up to 1MB when different threads are writing to the same DB, delaying *fsync*, and by recycling log files to reduce metadata updates.

Using the Append primitives, we can offload these group commits to the SSD.

(1) With Zone Append, we can persist changes to the log more efficiently, issuing a Zone Append for every 4KB[2] of log records, significantly reducing the current 1MB granularity of group commit.
(2) With our proposed Group Append, we can issue a *byte-append* for every WAL entry. The controller can group these into the device's block size, greatly reducing space and write amplification.

We plan to explore the impact of these approaches on the space and write amplification caused by WAL. Since logs can be stored in designated zones, and these append primitives can issue writes without knowing the tail, we can eliminate the need for the 4KB update to the file metadata and use device blocks more efficiently for data. We plan to explore the overhead of this mechanism in terms of excess commands on the NVMe queues and additional responsibilities for the controller.

5.1.3 ***Updating Compaction:*** Compaction constrains the shape of the LSM tree, merging SSTables with other SSTables in the same level and pushing this data to the next level (tiered compaction) or merging SSTables from level L_n into larger SSTables at level L_{n+1} (leveled compaction). Compaction ensures that all data in a single SSTables has the same lifetime.

(1) For tiered compaction, garbage collection is greatly simplified by zoned storage; once compacted data has migrated to the next level, the original zones can be reclaimed for use by new SSTabless.
(2) For leveled compaction, the merged zone at L_n can be reset, however the revised zones at L_{n+1} have to be written anew, with their old versions reset, since they cannot be updated in-place. Using leveled compaction could cause massive write amplification on ZNS SSDs as the larger SSTable will have to be completely rewritten after a merge. We plan to explore the layout strategies of larger SSTables and techniques to partition them on a zoned SSD to avoid rewriting the entire SSTable after a merge.

To reduce host overhead, the NVMe Copy command [61] offers the ability for a host to offload copy operations onto the SSD. We can use the Copy command on a ZNS SSD to copy sections from multiple zones into a single zone, and with the right design changes, we could reduce data movement to the host when merging multiple SSTables. The keys can be sorted and remapped by the host while the data is appended from multiple SSTables (on multiple zones) to a new zone. Furthermore, we can completely offload the merge process of compaction to a dedicated compute unit on the SSD. As RocksDB priorities change towards reducing computation [27] the ability to offload merge would reduce the amount of host CPU usage while improving performance.

5.1.4 ***Handling Metadata and Bloom Filters:*** There are many kinds of metadata that RocksDB deals with; there is filesystem metadata of the backing store, metadata in the form of the manifest file for RocksDB, file metadata about WAL files and SSTable files, and finally, the metadata within these files. Updates to metadata generally involve frequent small random writes, which are unsuitable for ZNS SSDs. There are many ways we could store the metadata; (i) an SSD could expose both conventional random-write namespaces and zoned namespaces, where metadata could be routed to the conventional namespace. (ii) Metadata updates could be converted into appends to dedicated metadata zones with Group Appends to improve performance and limit write amplification. In the case of heterogeneous storage, a non-volatile storage-class memory for metadata would be a convenient solution.

In RocksDB, every SSTable file contains a Bloom filter (or in some cases a Ribbon Filter [24]). Combining multiple Bloom filters is not feasible, so Bloom filter recomputation is required after a merge. If the Bloom filter can be replaced with a data structure that allows easy append-friendly merges, the recomputation of a new Bloom filter could be avoided. Since merges of disjoint SSTables could be accomplished via discriminated appends, offloading compaction to computational storage seems feasible.

5.2 Research Directions

LSM trees are well suited for ZNS SSDs, and we see great potential for research in the following areas:

- **Layout of the LSM Tree:** Alignment of LSM trees to zones introduces challenges when their sizes are not the same. We plan to explore layout strategies and data grouping to reduce data movement on operations such as compaction.
- **Data Lifetime:** While RocksDB supports lifetime hints, generating these hints for a workload is an open research area. Knowing the lifetime of a SSTable would allow us to map data with similar lifetimes to the same zone, reducing write amplification.
- **Compaction:** Adapting leveled, tiered, and other compaction techniques to the ZNS interface is essential to use Zone Reset efficiently. We plan to explore the best strategies for compaction.
- **Indexing, Filters, and Metadata:** Since updates to these structures often involve frequent small random writes, managing them on append-only storage is a challenge. We plan to explore how the byte-append capabilities introduced by Group Append could be adopted to work with metadata.
- **Data Structures:** While adapting LSM Tree structures for ZNS is useful, updating data structures in LSM, like SSTables to be more size-sorted, or lifetime-sorted would provide benefits in terms of reducing write amplification, and reducing GC overhead when used with ZNS SSDs. Such 'zone-aware' SSTables could address inter-zone dependencies and replace the traditional SSTables on ZNS SSDs.

[2]Block size can currently be as small as 512 bytes

6 DATABASES

Many database management systems achieve ACID properties using a well-known combination of in-memory data, persisted data, and a persisted database log [32]. Group Commit is used to combine ("boxcar") logging at the logger so that the number of forces to the persistent log is reduced, trading (sometimes slower) commit latency for both improved throughput and fuller log blocks. Transaction Commit ordering dependencies (a partial ordering) must be reflected by Commit record ordering in the log (often a total ordering). There can be multiple loggers writing to the persistent log, Write-Ahead Log (WAL) requires that all log records for a transaction be persisted before that transaction's Commit record. There can also be multiple persistent logs for a database, as long as these logs are coherent, meaning that they become durable in a coherent order [23, 35].

6.1 Evolution to ZNS SSDs

For a database running on ZNS SSDs, logging can be handled using the approach described in §1, pushing grouping down from the logger (which uses Group Commit) to the controller (which uses Group Append). Log records in a log could be transmitted from a single logger thread or from multiple threads, but log sequence order must be preserved. Since forcing records to the log is relatively fast and inexpensive with Group Append, loggers could persist log records more frequently than is typical with Group Commit. That could increase the load on the controller, but it also would decrease Write-Ahead Log (WAL) delays. It would also reduce the number of log blocks written because blocks wouldn't be written until they are (nearly) full. However, a committing transaction would still have to wait until its commit record was acknowledged by the controller.

A log file would consist of a sequence of zones (consisting of at least two zones). When a zone is full, logging continues in the next zone in the sequence. A log file is a circular buffer, which is beheaded when early log records are no longer needed so that there's always room for new log records. With ZNS storage, Zone Erase efficiently clears an unneeded zone at the head of the log.

Checkpoints, which are used for failure recovery, could still be included in the same log, and recovery will have to find the most recent checkpoint. There is a (limited) modifiable metadata area associated with a zone even when using Zone Append, and the location of the recent checkpoint (an LBA) could be stored there. Other anchor data (some small number of MB), such as the sequence of zones comprising the log including the current head and tail zones of the log, could also be stored in the metadata area. It would also be simple to store such metadata on a separate conventional SSD.

6.1.1 ***Other Database Systems:*** There are other ways that Zone Append could be applied to existing or novel database systems. Each offers replication capabilities at various levels, which we do not discuss in detail. Within a data center, striping and mirroring may be offered in hardware, and a controller could replicate by communicating directly with other controllers.

Hyder [16] was designed for SSDs and works best on raw SSDs, which eliminates log stacking. Hyder uses an append-only [52] capability that enables multiple loggers to access the same log efficiently. Zone Append provides that capability directly, simplifying design. A version of Hyder was also implemented based on the CORFU/Tango shared log that is discussed in §7.

Silo [58] and other "multicore optimistic epoch group" systems submit log records from different workers and might benefit from a Group Append capability.

SQLite [4] (in WAL mode, not journal mode) seems like a great fit for Zone Append, with changes logged and the original DB unchanged. SQLite checkpoints would create a new zone version of the database based on roll forward, whereupon the old zone version could be erased. The database could be sharded based on update frequency (if that is predictable or otherwise determinable), with data placed in each shard based on expected lifetime. Since SQLite is widely used in mobile phones, embedded systems, and applications, improving cost and performance for SQLite would be important, if ZNS Storage is produced for embedded devices [3].

Relational Databases, such as mySQL, can integrate zoned storage utilizing a storage backend such as MyRocks/RocksDB [48], which translates in-place updates to being log-structured updates.

6.2 Research Directions

- **Managing Table Deltas:** If differential changes to a table are appended to an append-only table delta [29] that is managed using Group Append, then there is no need to log those table changes in the usual database log. (Log management using non-volatile memory employs a similar principle [8].) The table delta (which would both be maintained in memory and persisted) would include the ID of the transaction that made the change, which might commit or roll back.
  The standard DB log would contain Commit records, which would include the transaction ID and the logical timestamp of the Commit; it would also contain Rollback records for transactions that roll back. If a transaction commits, an in-memory data structure would identify the mapping between transaction ID and the logical timestamp of the Commit; that mapping would not have to be maintained forever. Periodic merging of table and table delta (once that delta becomes immutable after a new delta is created) could be handled using disaggregated storage, as in Amazon Aurora [59]. After merging deltas, data from rolled back transactions would be eliminated, and data from committed transactions would contain the logical timestamp of the commit.
- **Metadata and Utilities:** Databases manage a wide range of metadata. Some of that metadata can be treated as if it was ordinary data, but configuration data and checkpoint anchors require special handling. Also, databases have a range of utilities (e.g., load, backup, indexing, archive). ZNS commands seem like great fits for some utilities (e.g., device-side copy for backup), but as usual, there are fuzzy subtleties when doing backups in a running system.
- **Replication:** We discuss general replication techniques later in this paper. But there's one tempting replication technique that would not work if used simplistically. If log writes are logically replicated from loggers (or from devices) to other devices (locally or remotely), then transaction commit order would not necessarily agree across those devices, so replication coherence of logs would be violated. However,

more sophisticated techniques (based on quorums, Paxos, or ordered epochs) could be employed successfully.

7 EVENT LOGS AND SHARED LOGS

Event Logs: Event logs track application and system actions. Logs can be analyzed to understand system performance, trends, and anomalies. A log can also be treated as a message bus, publishing events (or summaries of events) to subscribers. But if there are failures, events immediately prior to the failure may be lost, events which might help explain the root cause of the failure. If events are pushed down to the controller using Group Append, then durable logging is faster, and recent log records are less likely to be lost.

Shared Logs: Shared logs are shared Event Logs; Scalog [25] describes them as "a sequence of ordered records that can be accessed and appended to by multiple clients." There are many highly-scalable distributed Shared Logs which enforce ordering guarantees, including Kafka [39], CORFU [11], Tango [12], ZLog [56], and Scalog [25].

7.1 Evolution to ZNS SSDs

If there is a failure, events that occurred immediately prior to that failure may not appear in an Event Log. Those missing events might help explain the root cause of the failure. But if events are pushed down to the controller quickly using Group Append and Zone Append, then recent log records are less likely to be lost.

Let's examine how one Shared Log system, Scalog, would work with ZNS SSDs. Scalog defines append and appendToShard operations which map directly to Zone Append. FIFO replication of log segments corresponds to sending a collection of blocks, perhaps controller-to-controller, with fast acknowledgment by the receiving controller. Matching block boundaries, blocks after the last block previously transmitted can be appended using Zone Append.

7.2 Research Directions

Scalable Shared Logs with elastic scalability are vital to modern data management. Here are a few of the intriguing Event and Shared Log Research challenges for ZNS:

- **Multiple Writers:** For Event Logs, is it better to have a single writer handling all events, or multiple writers that can append to the tail of the log?
- **Computational Storage:** Should events be sent to computational controllers as soon as they are generated, even before event ordering is determined? Should replication and ordering for Shared Logs be managed using cooperative communication across computational storage whenever possible, minimizing host involvement?
- **Zone Discrimination:** Can/should there be discrimination (based on data access patterns) of the zones in which events are stored, either when events are initially stored or when Shared Logs are reorganized? Could this support efficient garbage collection of zones that contain less important data, while still supporting order requirements?

8 GENERAL RESEARCH DIRECTIONS

ZNS SSDs open up co-design possibilities corresponding to the following directions, which apply to the Data Management areas discussed in this paper.

8.1 Placement

ZNS SSDs provide greater control over the physical data layout on the underlying media. This enables the host or the application to control spatial data layout. Because Group Append is space-efficient, sharding (and reorganizing) logs at the controller level into multiple zones (based on expected data life-cycle) might be advantageous, as long as log semantics are preserved and log operations are efficient. Data placement could be further tuned with algorithms that learn access characteristics of data and workloads over time [13].

8.2 Computational and Disaggregated Storage

Group Append can be used by reorganization utilities (e.g., RocksDB compaction) which perform operations in a Disaggregated computational storage device. Coordination among different compute units is trivial when the data they share is immutable (like full zones). Furthermore, Computational Storage [60] could support operations combining the in-memory buffer, controller buffer, and persisted data, which would offer promising new opportunities for application offload. Data-centric operating systems [17] open up the possibility of allowing compute adjacent to data, which would enable better integration of computational storage with ZNS.

8.3 Price/Performance

ZNS SSDs that support Group Append can further improve the overall write amplification for sub-logical block writes, decreasing the overall writes to the SSD, further enabling QLC and PLC flash-based SSDs.

Zoned storage offers efficiency advantages for these research areas (and for other research areas described in individual sections of this paper), and Group Append and Zone Append reinforce these advantages. We intend to explore these opportunities using both actual and simulated hardware implementations, and we hope that other researchers will pursue similar opportunities.

8.4 Metadata

We have made proposals about metadata management in earlier sections of this paper. Metadata can sometimes be placed at the beginning or at the end of the zone holding the data it describes. But there are other alternatives, including

(1) keeping metadata in dedicated metadata zones would simplify finding and managing metadata,
(2) placing metadata in the small random-write and in-place update friendly area that is associated with each zone,
(3) keeping it in NVM associated with the controller, and
(4) persisting metadata changes on a device that is not zoned and also and keeping it in memory. The trade-offs among these approaches in different circumstances are not simple and warrant research study.

There also exist multiple ways to maintain the consistency of metadata and data when using ZNS. The log is a circular sequence

of zones, with timely background truncation (using Zone Erase) of the head zone of the log, so that a "next zone" is always available.

Anchor data describing the zones comprising the log can be kept in a well-known location, perhaps on a standard SSD. The most recent zone can also be stored as anchor data, or it can be determined based on (circularly) increasing sequence numbers kept at the beginning of each zone. This requires care, especially since a log-append might require multiple zones, e.g., for large SSTables. So a multi-zone append cannot be regarded as complete until its entire contents have been written to the zone. There are several ways that an append library can choreograph this; making zone transitions transparent to the host may be the best approach.

8.5 Replication

Storage devices and systems sometimes fail [20], or degrade the performance of systems to exhibit unacceptable delays (jitter) [34]. There are many well-known techniques for replicating data (including logs) locally and remotely [7], including RAID (with mirroring and/or striping), log-shipping quorums, Paxos/Raft, quorum protocols (at various levels), and coherent logical or physical copying to Backup(s). There are complex tradeoffs among these approaches across Latency, Consistency, Availability, Partition-tolerance, etc. [5]. Device-to-device protocols (even to remote devices) may be particularly attractive for logs, but one size won't fit all. Replicated logs don't have to be equal, but they do need to be equivalent, or (more generally) have a coherent interpretation. [46]. Performing log appends independently at multiple sites is tempting, but it's not coherent unless we can deliver high-level semantics layered on physically different, not necessarily equivalent logs. Equivalence for some operations is simple; RocksDB compression can be performed independently at different replicas at different times, since replicas deliver equivalent key-value store semantics, regardless of when (or how) compression is performed. The trade-offs among replication techniques meeting requirements of specific log-structured systems are complex, warranting detailed investigation, analysis, and evaluation.

8.6 Zone Random Write Area

Zone Append is an optional feature as a part of the ZNS Specification. Alternative ways to address the issue of write pointer contention include a proposal for Zone Random Write Area (ZRWA) [31]. This approach exposes the write buffer for the particular zone and allows random writes as well as in-place updates to this buffer. This buffer can then be explicitly committed with a command or can be committed as it fills up. The main benefit of this approach is that the host interface does not require any changes as random writes are allowed. Persistence within the zone is guaranteed by the device. Group Append can be implemented with a slightly modified interface, as this approach utilizes higher queue depths, but limits write granularity to block sizes. Allowing random-write and in-place updates before commits open up new designs for managing metadata and frequently updated data.

9 CONCLUSION

When innovative hardware and interfaces to hardware emerge, software systems should change to leverage those innovations. Computational storage, disaggregated storage, and zoned storage with ZNS SSDs are such innovations. We're particularly intrigued by opportunities for data management systems that exploit ZNS SSDs, with Group Append pushed down to the controller. Fortunately, this exploitation can sometimes be addressed by evolving the data management system's logging component using techniques detailed in this paper.

This paper identified a broad range of hardware and software co-design research issues that require experimentation and analysis to resolve. We intend to do such work ourselves, working with partners, and we also hope that other researchers in industry and academia will also address these problems, perhaps collaborating with us on them. Although initial solutions to some of these problems using ZNS SSDs may be straightforward, symbiotic co-design (across applications, storage systems, programmable devices, and workloads) offers rich research opportunities that may transform many of these data management systems in unexpected ways.

Acknowledgements

This work was supported in part by the National Science Foundation (NSF Grants IIP-1266400, IIP-1841545, CCF-1942754) and the industry members of Center for Research in Storage Systems (CRSS) at University of California, Santa Cruz.

We are grateful to our collaborators whose advice, insight, and feedback helped guide this paper. Matias Bjørling from Western Digital helped us understand ZNS and improved this paper significantly through several rounds of feedback. We had extensive discussions with Simon Lund, Adam Manzanares, Bala Ganeshan, and Javier Gonzalez from Samsung about the nature of ZNS, and received valuable feedback on this work. Siying Dong from Meta provided us with insights into RocksDB, which helped improve our discussion on LSM Trees in § 5. We thank Pat Helland from Salesforce for the feedback we received from him. We are grateful for the feedback and support we received from our collaborators at the CRSS, Ethan L. Miller, Darrell D. E. Long, Daniel Bittman, and Peter Alvaro.

REFERENCES

[1] 2020. RocksDB Wiki: WAL Performance. https://github.com/facebook/rocksdb/wiki/WAL-Performance
[2] 2021. *RocksDB | A persistent key-value store.* http://rocksdb.org/
[3] 2021. *Well-Known Users Of SQLite.* https://www.sqlite.org/famous.html
[4] 2021. *Write-Ahead Logging.* https://sqlite.org/wal.html
[5] Daniel Abadi. 2012. Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *Computer* 45, 2 (2012), 37–42. https://doi.org/10.1109/MC.2012.33
[6] Ian F Adams, Neha Agrawal, and Michael P Mesnier. 2021. Enabling Near-Data Processing In Distributed Object Storage Systems. 7.
[7] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. 2019. Dissecting the Performance of Strongly-Consistent Replication Protocols. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1696–1710. https://doi.org/10.1145/3299869.3319893
[8] Joy Arulraj, Matthew Perron, and Andrew Pavlo. 2016. Write-behind Logging. *Proc. VLDB Endow.* 10, 4 (nov 2016), 337–348. https://doi.org/10.14778/3025111.3025116
[9] Manos Athanassoulis, Michael S Kester, Lukas M Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki, and Mark Callaghan. 2016. Designing Access Methods: The RUM Conjecture.. In *EDBT*, Vol. 2016. 461–466.
[10] Jen Axboe. 2017. Add support for write life time hints [LWN.net]. https://lwn.net/Articles/726477/
[11] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobbler, Michael Wei, and John D. Davis. 2012. CORFU: A Shared Log Design for Flash

Clusters. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, San Jose, CA, 1–14. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/balakrishnan

[12] Mahesh Balakrishnan, Dahlia Malkhi, Ted Wobber, Ming Wu, Vijayan Prabhakaran, Michael Wei, John D. Davis, Sriram Rao, Tao Zou, and Aviad Zuck. 2013. Tango: distributed data structures over a shared log. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. 325–340. https://doi.org/10.1145/2517349.2522732

[13] Oceane Bel, Kenneth Chang, Nathan Tallent, Dirk Duellman, Ethan L. Miller, Faisal Nawab, and Darrell D. E. Long. 2020. Geomancy: Automated Performance Enhancement through Data Layout Optimization. In *Proceeding of the Conference on Mass Storage Systems and Technologies (MSST '20)*.

[14] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. 1995. A critique of ANSI SQL isolation levels. *ACM SIGMOD Record* 24, 2 (1995), 1–10.

[15] Philip A. Bernstein, Sudipto Das, Bailu Ding, and Markus Pilman. 2015. Optimizing Optimistic Concurrency Control for Tree-Structured, Log-Structured Databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1295–1309. https://doi.org/10.1145/2723372.2737788

[16] Philip A Bernstein, Colin W Reid, and Sudipto Das. 2011. Hyder-A Transactional Record Manager for Shared Flash.. In *CIDR*, Vol. 11. 9–20.

[17] Daniel Bittman, Peter Alvaro, Pankaj Mehra, Darrell D. E. Long, and Ethan L. Miller. 2020. Twizzler: a Data-Centric OS for Non-Volatile Memory. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*.

[18] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. 2021. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 689–703. https://www.usenix.org/conference/atc21/presentation/bjorling

[19] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. 2017. LightNVM: The Linux Open-Channel SSD Subsystem. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*. USENIX Association, Santa Clara, CA, 359–374. https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling

[20] Chandranil Chakraborttii and Heiner Litz. 2020. Improving the accuracy, adaptability, and interpretability of SSD failure prediction models. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 120–133.

[21] Chandranil Chakraborttii and Heiner Litz. 2021. Reducing write amplification in flash by death-time prediction of logical block addresses. In *Proceedings of the 14th ACM International Conference on Systems and Storage*. 1–12.

[22] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. 2008. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* 26, 2 (2008), 1–26.

[23] David J DeWitt, Randy H Katz, Frank Olken, Leonard D Shapiro, Michael R Stonebraker, and David A. Wood. 1984. Implementation Techniques for Main Memory Database Systems. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data* (Boston, Massachusetts) *(SIGMOD '84)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/602259.602261

[24] Peter C. Dillinger and Stefan Walzer. 2021. Ribbon filter: practically smaller than Bloom and Xor. arXiv:2103.02515 [cs.DS]

[25] Cong Ding, David Chu, Evan Zhao, Xiang Li, Lorenzo Alvisi, and Robbert Van Renesse. 2020. Scalog: Seamless Reconfiguration and Total Order in a Scalable Shared Log. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 325–338. https://www.usenix.org/conference/nsdi20/presentation/ding

[26] Jaeyoung Do, Ivan Luiz Picoli, David Lomet, and Philippe Bonnet. 2021. Better database cost/performance via batched I/O on programmable SSD. *The VLDB Journal* (2021), 1–22.

[27] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Evolution of Development Priorities in Key-value Stores Serving Large-scale Applications: The RocksDB Experience. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, 33–49. https://www.usenix.org/conference/fast21/presentation/dong

[28] Jake Edge. 2019. Accessing zoned block devices with zonefs. https://lwn.net/Articles/794364/

[29] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. 2012. The SAP HANA Database–An Architecture Overview. *IEEE Data Eng. Bull.* 35, 1 (2012), 28–33.

[30] Javier González and Matias Bjørling. 2017. Multi-tenant I/O isolation with open-channel SSDs. In *Nonvolatile Memory Workshop (NVMW)*.

[31] Javier González. 2020. Zoned Namespaces: Use Cases, Standard and Linux Ecosystem. In *SNIA Storage Developer's Conference SDC EMEA' 20*.

[32] Jim Gray and Andreas Reuter. 1992. *Transaction Processing: Concepts and Techniques* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[33] Pat Helland. 2015. Immutability Changes Everything: We need it, we can afford it, and the time is now. *Queue* 13, 9 (2015), 101–125.

[34] John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas Anderson, and Arun Venkataramani. 2008. Consensus Routing: The Internet as a Distributed System. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*. USENIX Association, San Francisco, CA. https://www.usenix.org/conference/nsdi-08/consensus-routing-internet-distributed-system

[35] Ryan Johnson, Ippokratis Pandis, Radu Stoica, Manos Athanassoulis, and Anastasia Ailamaki. 2012. Scalability of write-ahead logging on multicore and multi-socket hardware. *The VLDB Journal* 21, 2 (2012), 239–263.

[36] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-Streamed Solid-State Drive. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems* (Philadelphia, PA) *(HotStorage'14)*. USENIX Association, USA, 13.

[37] Taejin Kim, Duwon Hong, Sangwook Shane Hahn, Myoungjun Chun, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. 2019. Fully Automatic Stream Management for Multi-Streamed SSDs Using Program Contexts. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. USENIX Association, Boston, MA, 295–308. https://www.usenix.org/conference/fast19/presentation/kim-taejin

[38] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2017. Reflex: Remote flash=local flash. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 345–359.

[39] Jay Kreps, Neha Narkhede, Jun Rao, and others. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, Vol. 11. 1–7.

[40] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.

[41] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. 2015. F2FS: A New File System for Flash Storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, Santa Clara, CA, 273–286. https://www.usenix.org/conference/fast15/technical-sessions/presentation/lee

[42] Alberto Lerner and Philippe Bonnet. 2021. Not Your Grandpa's SSD: The Era of Co-Designed Storage Devices. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. 2852–2858. https://doi.org/10.1145/3448016.3457540

[43] Heiner Litz, David Cheriton, Amin Firoozshahian, Omid Azizi, and John P Stevenson. 2014. SI-TM: Reducing transactional memory abort rates through snapshot isolation. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. 383–398.

[44] Heiner Litz, Ricardo J Dias, and David R Cheriton. 2015. Efficient correction of anomalies in snapshot isolation transactions. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 4 (2015), 1–24.

[45] Heiner Litz, Javier Gonzalez, Ana Klimovic, and Christos Kozyrakis. 2021. RAIL: Predictable, Low Tail Latency for NVMe Flash. In *Transactions on Storage (ToS)*.

[46] Joshua Lockerman, Jose M. Faleiro, Juno Kim, Soham Sankaran, Daniel J. Abadi, James Aspnes, Siddhartha Sen, and Mahesh Balakrishnan. 2018. The FuzzyLog: A Partially Ordered Shared Log. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 357–372. https://www.usenix.org/conference/osdi18/presentation/lockerman

[47] Umesh Maheshwari. 2021. From Blocks to Rocks: A Natural Extension of Zoned Namespaces. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '21)*. 21–27. https://doi.org/10.1145/3465332.3470870

[48] Yoshinori Matsunobu, Siying Dong, and Herman Lee. 2020. MyRocks: LSM-tree database storage engine serving Facebook's social graph. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3217–3230.

[49] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33, 4 (1996), 351–385. Publisher: Springer.

[50] Jong-Hyeok Park, Soyee Choi, Gihwan Oh, and Sang-Won Lee. 2021. SaS: SSD as SQL Database System. In *Proceedings of VLDB Endowment*, Vol. 14. 1481–1488. https://doi.org/10.14778/3461535.3461538

[51] Ivan Luiz Picoli, Niclas Hedam, Philippe Bonnet, and Pinar Tözün. 2020. Open-Channel SSD (What is it Good For). In *CIDR*.

[52] Colin Reid and Phil Bernstein. 2010. Implementing an Append-Only Interface for Semiconductor Storage. *IEEE Data Eng. Bull.* 33 (Jan. 2010), 14–20. https://www.microsoft.com/en-us/research/publication/implementing-an-append-only-interface-for-semiconductor-storage/

[53] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. BTRFS: The Linux B-Tree Filesystem. *ACM Trans. Storage* 9, 3, Article 9 (Aug. 2013), 32 pages. https://doi.org/10.1145/2501620.2501623

[54] Mendel Rosenblum and John K Ousterhout. 1991. The design and implementation of a log-structured file system. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*. 1–15.

[55] Manoj P. Saha, Adnan Maruf, Bryan S. Kim, and Janki Bhimani. 2021. KV-SSD: What Is It Good For?. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 1105–1110. https://doi.org/10.1109/DAC18074.2021.9586111

[56] Michael A. Sevilla, Noah Watkins, Ivo Jimenez, Peter Alvaro, Shel Finkelstein, Jeff LeFevre, and Carlos Maltzahn. 2017. Malacology: A Programmable Storage

System. In *Proceedings of the Twelfth European Conference on Computer Systems* (Belgrade, Serbia) *(EuroSys '17)*. Association for Computing Machinery, New York, NY, USA, 175–190. https://doi.org/10.1145/3064176.3064208

[57] Theano Stavrinos, Daniel S. Berger, Ethan Katz-Bassett, and Wyatt Lloyd. 2021. Don't be a blockhead: zoned namespaces make work on conventional SSDs obsolete. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. Association for Computing Machinery, 144–151. https://doi.org/10.1145/3458336.3465300

[58] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. 18–32.

[59] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, James Corey, Kamal Gupta, Murali Brahmadesam, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvilli, and Xiaofeng Bao. 2018. Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. 789–796. https://doi.org/10.1145/3183713.3196937

[60] Peter Wilcox and Heiner Litz. 2021. Design for Computational Storage Simulation Platform. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems* (Online Event, United Kingdom) *(CHEOPS '21)*. Association for Computing Machinery, New York, NY, USA, Article 5, 8 pages. https://doi.org/10.1145/3439839.3459085

[61] NVM Express Workgroup. 2021. NVM Express Specification 2.0. https://nvmexpress.org/developers/nvme-specification/

[62] NVM Express Workgroup. 2021. NVM Express Zoned Namespaces Command Set 1.1. https://nvmexpress.org/developers/nvme-command-set-specifications/

[63] Sung-Ming Wu, Kai-Hsiang Lin, and Li-Pin Chang. 2018. KVSSD: Close integration of LSM trees and flash translation layer for write-efficient KV store. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 563–568. https://doi.org/10.23919/DATE.2018.8342070

[64] Minghao Xie, Chen Qian, and Heiner Litz. 2020. ReFlex4ARM: Supporting 100GbE Flash Storage Disaggregation on ARM SoC. In *OCP Future Technology Symposium*.

[65] Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman. 2014. Don't Stack Your Log On My Log. In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)*. USENIX Association, Broomfield, CO. https://www.usenix.org/conference/inflow14/workshop-program/presentation/yang