Evolving XFS with Zoned Storage and Intelligent Data Placement

Hans Holmberg
Western Digital Research
hans.holmberg@wdc.com

Christoph Hellwig
Western Digital Research
christoph.hellwig@wdc.com

1 Introduction

We introduce Zoned XFS, which aims to deliver the high capacity, low wear, and high performance of zoned storage devices through the XFS enterprise filesystem [3; 7]. Zoned storage devices require host writes to be sequential within zones and force updates to be made out of place. XFS was originally designed for in-place overwrites on conventional block devices, but was later enhanced to support out of place writes.

This extended abstract presents our efforts to enable zoned storage with XFS and how it expands the existing zoned storage work found in other enterprise filesystems [6] in two ways. Firstly, we utilize the zone append primitive [2], which eliminates the need to serialize the writes to the storage device. Secondly, we reduce write amplification and improve garbage collection performance by co-locating user data with similar lifetimes into zones based on file locality and application hints.

To achieve this, we enhance XFS with a new zoned allocator and a defragmenting garbage collection algorithm and evaluate the performance benefits on in-production storage devices using RocksDB [5].

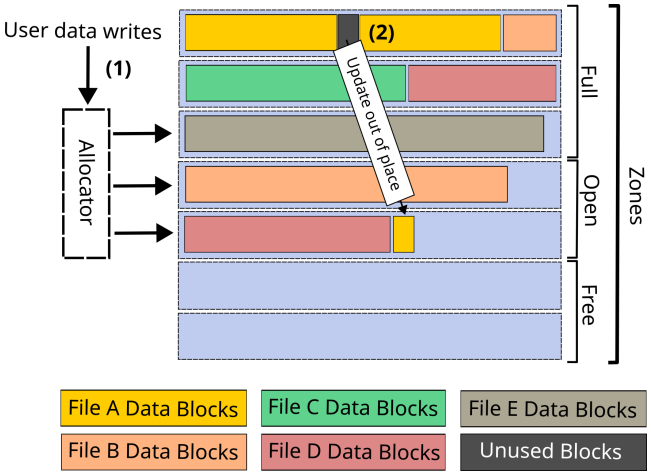

Figure 1. The zoned allocator writes file user data to a limited number of open zones (1). Updates to existing data is done out of place (2), requiring a new allocation. Once the new data blocks have been written, the old blocks are marked as unused.

2 Our approach

We implement zoned storage support based on the existing XFS real-time device feature [1], which enables using separate space allocators for metadata and file data. Metadata continues to use the existing B+ tree-based format requiring in-place updates on conventional storage. File data is allocated on zoned storage using our new zoned allocator. For file overwrites, the new data is redirected to newly allocated blocks using the existing mechanisms to support unsharing reference counted blocks (reflink)[8].

For file data placement, the new zoned allocator places data into a limited number of open zones utilizing the per-zone hardware write pointer to track available space, removing the need for persistent allocator data structures. Device writes are executed using the zone append [2] primitive which avoids the serialization that would be required when submitting regular write commands at the write pointer. See Figure 1. Because the disk location of writes is only recorded at I/O completion time, the number of transactions to perform an allocating write is halved compared to the conventional XFS block allocator.

The zoned allocator separates different file data into different zones when possible and co-locates file data using application provided write lifetime hints when available. Write lifetime hints can be set on a per file basis using an existing Linux kernel interface by applications with knowledge of the expected lifetimes of their on-disk data structures. Because zones are filled with data of similar lifetime, we reduce the amount of garbage collection that has to be performed.

To reclaim the space occupied by blocks invalidated by file removal or overwrites, garbage collection is implemented. Valid data in partially used zones is located using the physical-to-logical block reverse mapping tree originally added to support online repair [9] and moved to new zones while performing automatic data defragmentation of files. When the number of free zones drops below a predefined threshold, the zone with the least number of valid blocks is chosen for reclaim and all valid blocks are migrated file by file and written in file block order to open zones dedicated to garbage collection data and thus not mixed with freshly written user data. Garbage collected data can be assumed to be more long-lived than incoming user data, so the separation of user data and migrated data provides an additional heuristic to decrease write amplification. Garbage collection is performed in a highly pipelined fashion that allows reading data written to a single target zone from multiple victim zones without interrupting the garbage collection process. No locks are held over the garbage collection I/O, and thus garbage collection does not interfere with application reads and writes to

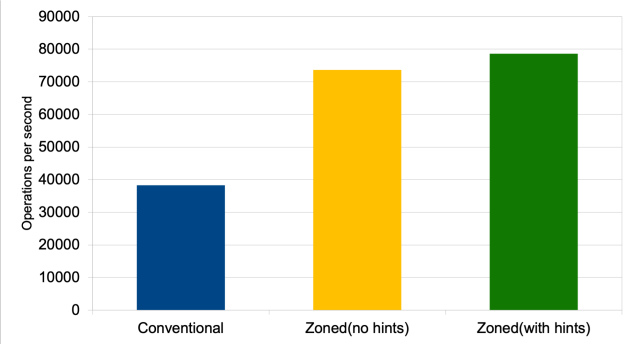

Figure 2. RocksDB db_bench overwrite benchmark, comparing XFS on conventional storage with Zoned XFS with and without write lifetime hint-based allocation.

files that contain data that is garbage collected. This significantly reduces the tail latency impact of garbage collection on application performance.

3 Results

We have evaluated the performance of the Zoned XFS implementation on production-based SSDs, comparing against baseline XFS on conventional SSDs using RocksDB [5], benchmarks and ZNS SSDs using the same hardware with 1TB storage capacity. RocksDB provides file systems with lifetime hints, marking its write ahead log files short lived and sorted string table files(SSTs) with increasing expected lifetime based on their level depth in the SST-tree.

First, we run a fillunique_random workload using the db_bench tool to fill up the file systems to 80% capacity performing several drive overwrites in the process and bringing the file systems into a steady state. Second, we run an overwrite benchmark to measure the effects of intelligent data placement.

We show that we can drastically reduce write amplification, improving the overwrite benchmark throughput by 92% by separating files into different zones. Utilizing write lifetime hints, where files of similar write lifetimes are co-located into the same zone, adds another 7%, see Figure 2.

4 Conclusion and further work

This abstract presents how the Zoned XFS enables a mature, high-performing file system to support zoned storage devices. Through intelligent data placement into zones by file locality and application provided write lifetime hints, we can improve key-value workload throughput by up to 99% by reducing write amplification. Our work is publicly available as open source and is being reviewed by the Linux community [4], opening up for further research in the area. We plan to investigate how to automatically estimate the data lifetime for applications that do not use explicit hints.

References

[1] Chandan Babu. XFS: Tracking space on a realtime device. https://blogs.oracle.com/linux/post/xfs-realtime-device, 2024.

[2] Matias Bjørling. Zone append: A new way of writing to zoned storage. Santa Clara, CA, February 2020. USENIX Association. https://www.usenix.org/conference/vault20/presentation/bjorling.

[3] Christoph Hellwig. XFS: The Big Storage File System for Linux. *USENIX ;login*, (34), 2009. https://www.usenix.org/system/files/login/articles/140-hellwig.pdf.

[4] Christoph Hellwig. Support for zoned devices. https://www.spinics.net/lists/linux-xfs/msg97163.html, 2025.

[5] Meta Platforms, Inc. RocksDB: A persistent key value store for fast storage environments. https://rocksdb.org, 2025.

[6] Marta Rybczyńska. Btrfs on zoned block devices. https://lwn.net/Articles/853308, 2021.

[7] Adam Sweeney. Scalability in the XFS file system. In *USENIX 1996 Annual Technical Conference (USENIX ATC 96)*, San Diego, CA, January 1996. USENIX Association. https://www.usenix.org/legacy/publications/library/proceedings/sd96/full_papers/sweeney.txt.

[8] Darrick Wong. XFS - Data Block Sharing (Reflink). https://blogs.oracle.com/linux/post/xfs-data-block-sharing-reflink, 2020.

[9] Darrick Wong. XFS - Online Filesystem Repair. https://blogs.oracle.com/linux/post/xfs-online-filesystem-repair, 2024.