SNIA DEVELOPER CONFERENCE
SDC 25
By Developers FOR Developers

Hyatt Regency Santa Clara, CA
September 15-17, 2025

Zoned XFS: Transparent and Efficient Zoned Storage Support For Scalable Storage Systems

Christoph Hellwig, Kernel Hacker
Damien Le Moal, Western Digital Research

www.sniadeveloper.org

Outline

- Introduction
  - Zoned Storage and XFS
- Zoned XFS
  - Storage space management
  - Writing files and managing file data mapping
  - Garbage collection
- Performance evaluation
  - SMR HDDs and ZNS SSDs micro benchmarks
  - RocksDB benchmark
- Conclusion

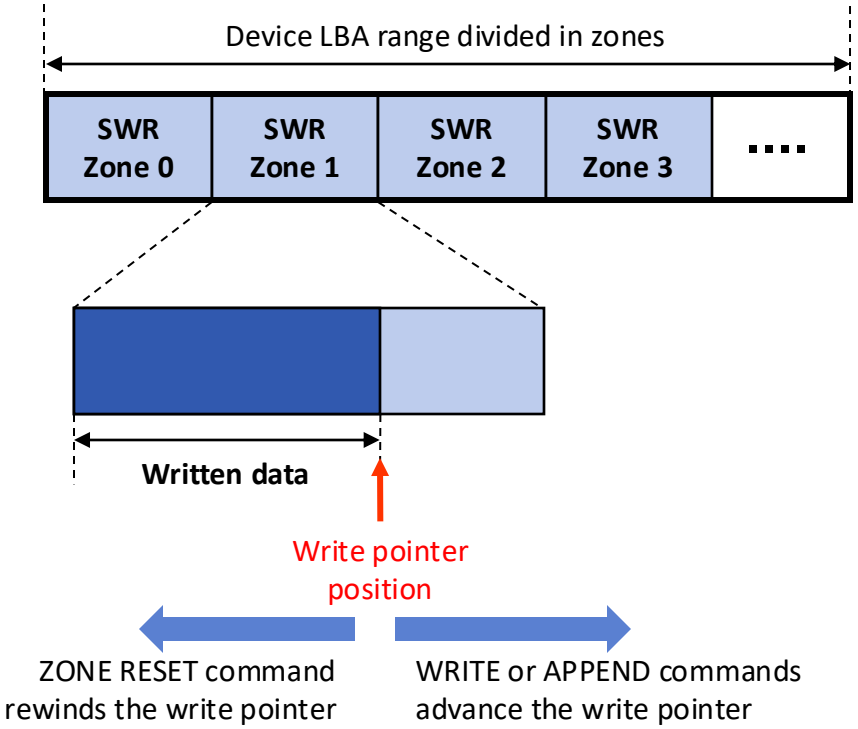
Introduction

What is Zoned Storage and Why Should You Care?

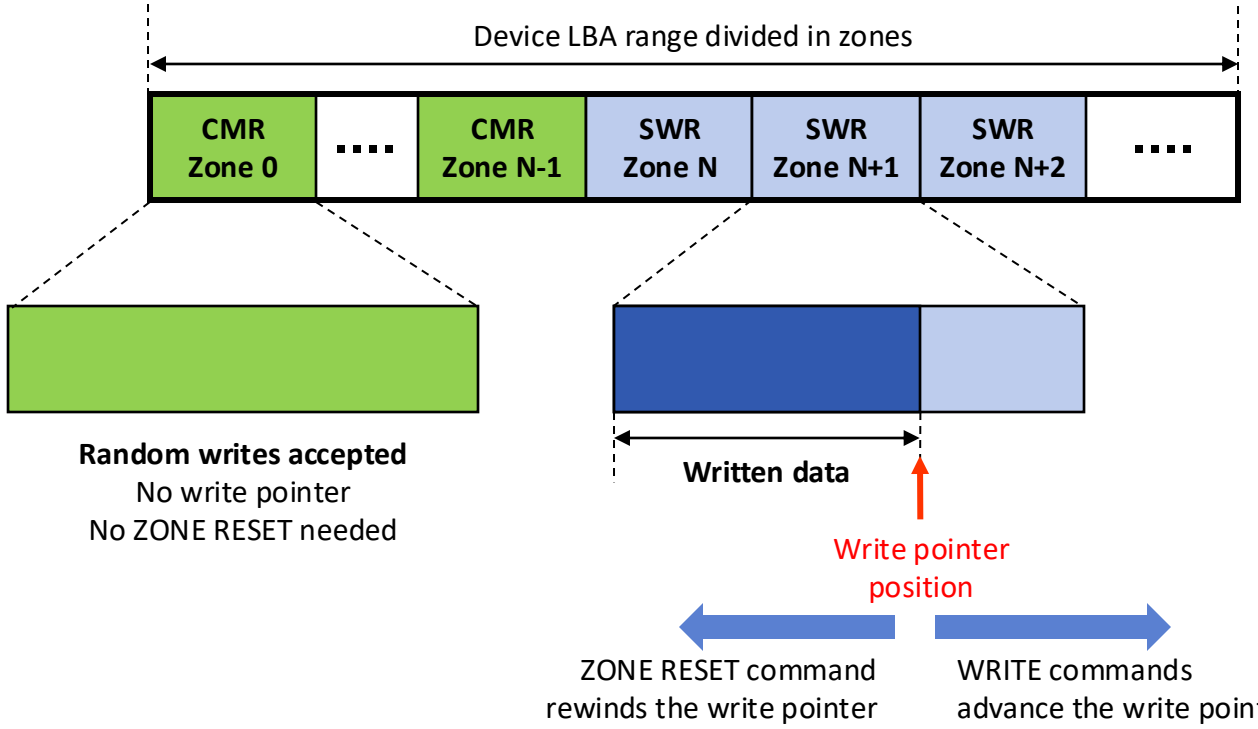
- Device storage address space is divided into regions called “zones”
  - Typically 265MB for HDDs, larger for enterprise SSDs
  - Zones must be written sequentially (SWR zone)
    - Optionally, the device can have conventional zones (CMR zone) that can be written randomly
  - Zones must be reset to allow writing again
    - All data still valid in a zone must be moved out first before reusing a zone
- Benefits: capacity and cost
  - SMR HDDs delivers higher capacities (20% to 25% higher than regular HDDs)
  - ZNS SSDs require less over-provisioning and less DRAM for the FTL
    - Removes most SSD internal activity, improving I/Os tail latency

What is Zoned Storage and Why Should You Care?

Zoned Namespace (ZNS) SSD



Host-Managed SMR HDD



What is XFS and Why Should You Care?

- Most scalable Linux file system
  - 30 years old now
    - Started on SGI/IRIX in 1993 and got ported to Linux in 2001
  - Heavily uses b+trees and variable sized extents
- Space managed using allocation groups (AGs)
  - Equally sized linear regions within the file system
  - Each allocation group manages its own inodes and free space separately
    - Enables excellent (scalable) performance with concurrent I/O threads

Some XFS Features

- Reverse mapping tree
  - Mapping from physical blocks to logical blocks using variable length extents indexed with a per-AG b+tree
- File data Copy-on-Write with *reflink* feature
  - Out-of-place file data block updates
    - Metadata is always updated in-place
- "Realtime" subvolume
  - Separate device used for (some) file data
    - Metadata remains on the "main" device
    - Originally used for HD Video recording / encoding on HDDs back in the late 1990s
    - Recently most often used for separating metadata on SSD and file data on HDD
      - e.g. Meta's "Hybrid XFS" used for Tectonic

Zoned XFS

What is Zoned XFS?

- Extension of XFS to support zoned block devices
  - Allows storing file data on zoned devices
    - File data is written sequentially in SWR zones
  - Metadata still requires random-write storage space
    - Conventional zones on a host-managed SMR HDD
    - A regular block device as the main device
      - Zoned device used as the realtime device for file data
- Support merged in Linux kernel v6.15
  - User space tools (xfsprogs) v6.15
  - Slowly trickling into distributions

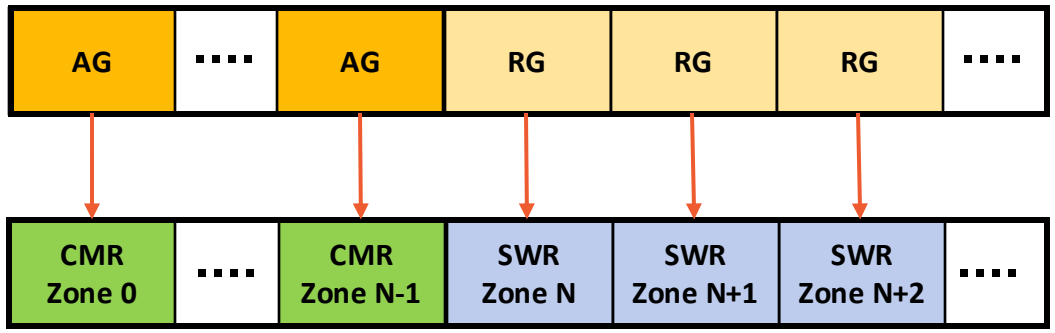
Design Overview

- Relies on the realtime subsystem to separate data and metadata
  - A realtime (data) groups always map to an SWR zones
  - An allocation (metadata) group maps to a CMR zone
- The reverse mapping tree is used for garbage collection decisions
  - When space is reclaimed to create free zones

Design Overview

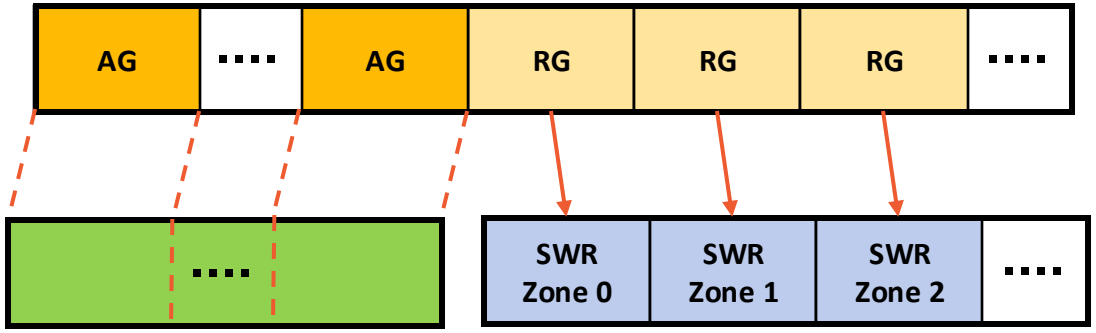
- For SMR HDDs which have CMR zones, the main and realtime devices are emulated on the same Linux block device
  - i.e. a single SMR HDD can be formatted without any additional device
- Multi-device setup can support ZNS SSDs and SMR HDDs without CMR zones
  - SMR HDDs with CMR zones can use this setup to offload metadata accesses to an SSD

Single Device (SMR) Allocation Groups Mapping



Host-Managed SMR HDD
Main and Realtime Device

Multi-Device (ZNS or hybrid) Allocation Groups Mapping



Regular Block Device
Main Device

Zoned device
Realtime device

Space Allocation

- Metadata reuses the regular space allocator
- For file data, space allocation is reduced to choosing an SWR zone
  - Zoned devices limit the number of zones that can be partially written (open zones)
    - The limit can be very small for SSDs, but can be hundreds of zones for SMR HDDs
  - Zoned XFS can use all available open zones, within the device limits
  - One open zone is set aside for garbage collection
    - The rest is available to user data writes
- File data is spread out to different open zones on a per inode (file) basis by default
  - With a few heuristic for grouping data from multiple files into the same zone instead
  - Separating files into different zones reduces write amplification

Space Allocation

- Untar optimization
  - Unpacking of archives files, software installation or similar workloads can create many small files that are written once and rarely if ever updated
  - In this case, try to reuse the same zone to tightly store files when written back by the VM after they have been closed
- Data temperatures
  - The F_SET_RW_HINT *fcntl()* system call allows setting a data temperature for a file (inode)
  - Zoned XFS tries to collocate file data with the same lifetime in the same zones if we cannot spread files out in different zones
    - All hot data is always co-located in the same zones as we do not expect it to stay around for a long time

Write I/Os

- When using regular write operations, a zone of a zone device must be written at its write pointer
  - limits the maximum write command queue depth to 1 per zone
- Zoned XFS uses the Zone append primitive, which is directed to a zone and not an exact block
  - The device then assigns the written block and returns it
  - For ZNS SSDs, this allows high queue depth operation
  - For HDDs, emulates by the block layer using regular write commands
    - Does not remove the QD=1 per zone write limit, but very fast turnaround because zone write plugging is implemented close to the hardware

File Data Mapping Metadata

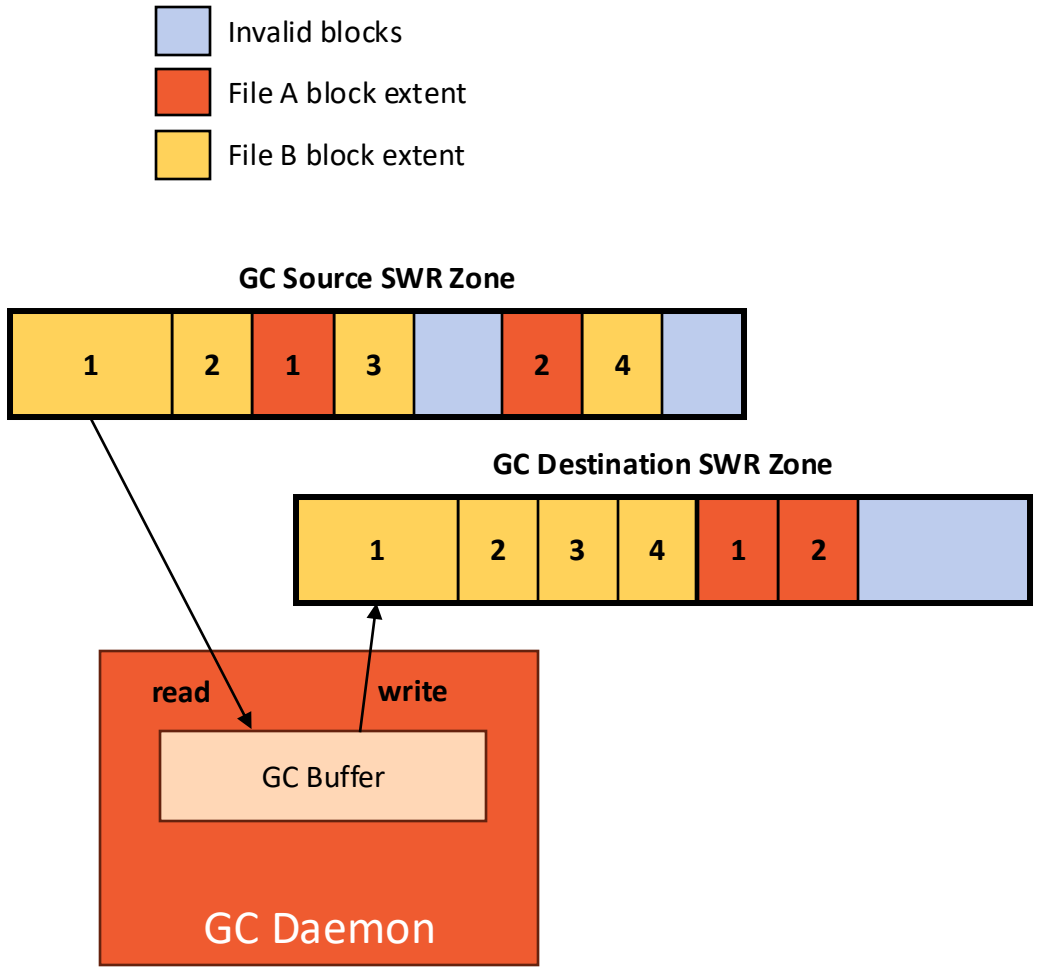
- Zone Append commands only specify the target zone where a file data must be written
  - The device indicates the first block written in the zone on completion of the zone append command
- The I/O completion handler records the logical-to-physical and physical-to-logical mappings of the written file data
  - There is no chance to have invalid data in the mapping, and thus no need to allocate unwritten extents that cannot be read before the I/O completes
- File data blocks that are removed or overwritten become invalid
  - These can only be reused after a zone reset
  - Records of removed blocks are deleted from the reverse mapping btree, and the used blocks counter for the RG is decremented

Garbage Collection

- Garbage collection (GC) is required to reclaim space occupied by unused blocks
  - Zones that do not contain any valid block anymore are reset immediately and made available for writing new user data
- A single thread GC daemon is used to copy (evacuate) valid blocks of a zone to a GC reserved zone
  - The GC daemon is woken up when the number of free zones falls below a predefined (configurable) threshold
  - GC Candidate zones are sorted in buckets based on their number of valid blocks
    - The first zone with the least number of valid blocks is always selected first

Garbage Collection

- Valid data is identified using the reverse mapping btree
  - Valid data is always copied to a GC-reserved zone that is unused for writing incoming user data
    - Garbage collected data can be assumed to be longer lived, so this provides a simple hot/cold separation heuristic, reducing write amplification
  - File data blocks are sorted before copying providing defragmentation "for free"
- The migration process is pipelined and adds minimal locking overhead



Performance Evaluation

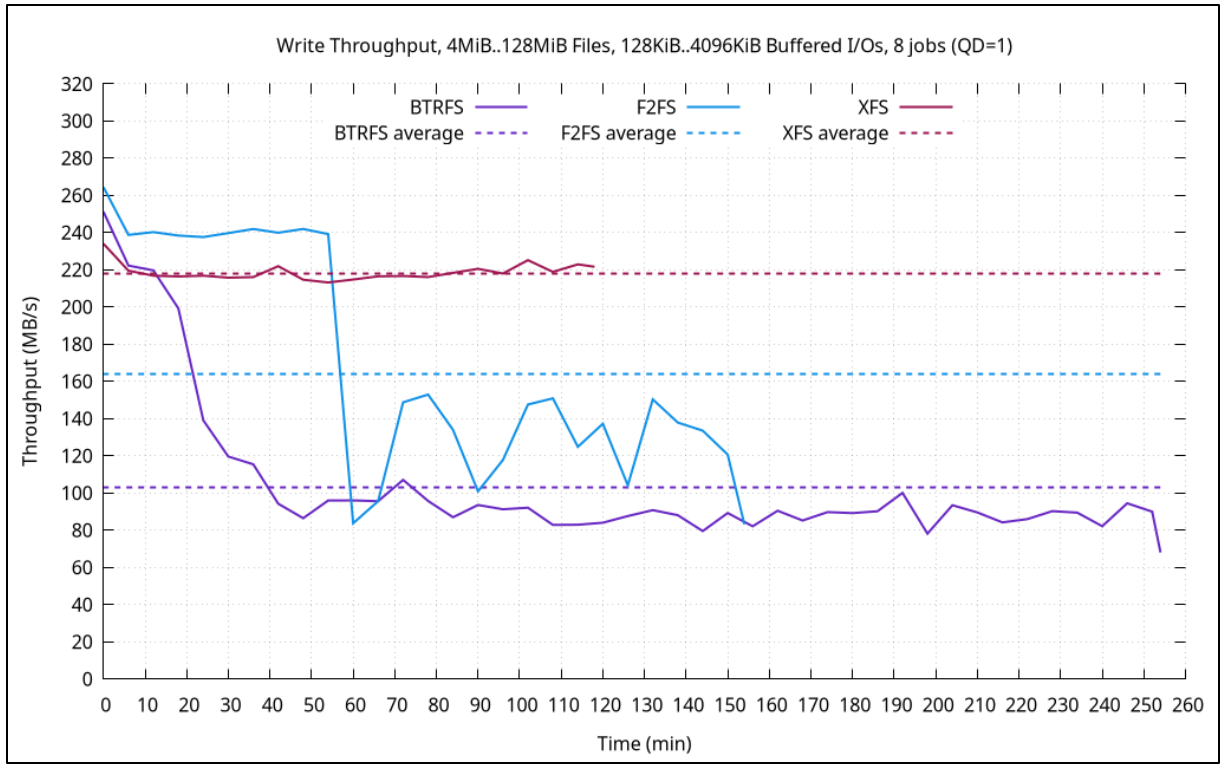
fio micro benchmarks and RocksDB benchmark

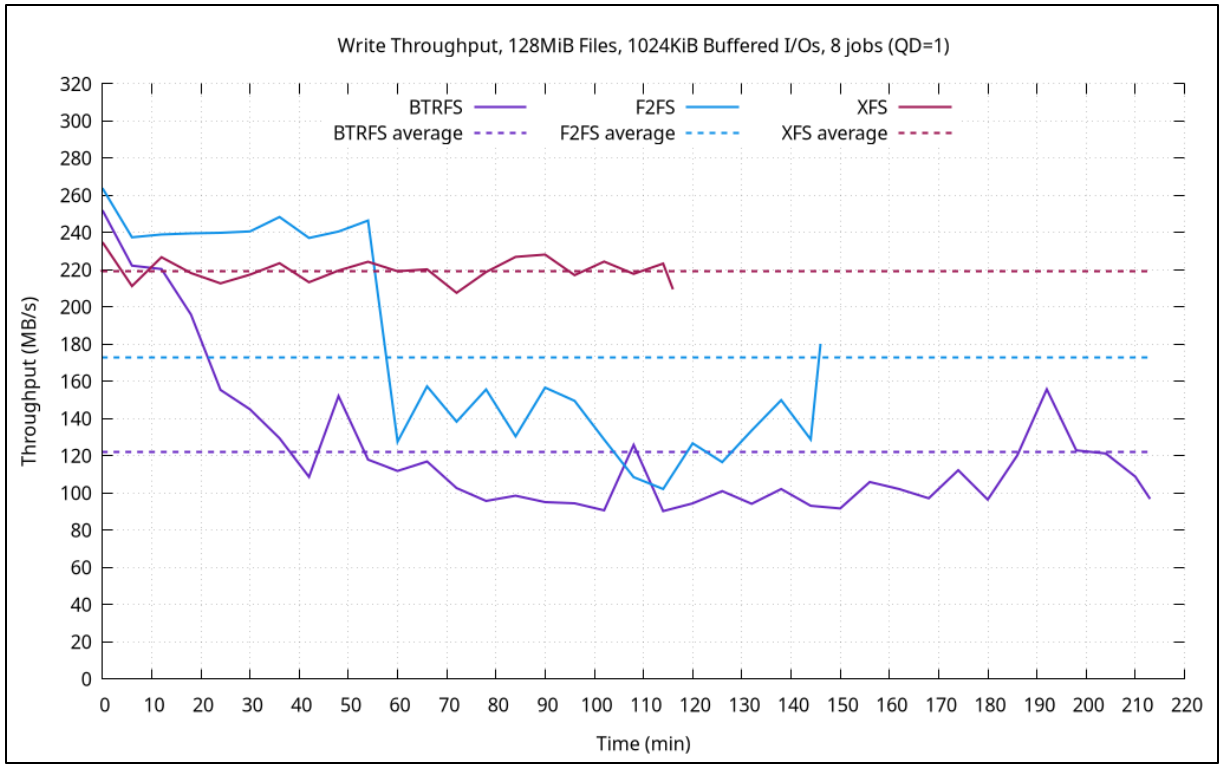
fio Micro Benchmark

- Write, read and overwrite files in 3 phases
  - Write: 8 fio jobs write files until the file system is 75% full
  - Read: 8 fio jobs randomly pick files and sequentially read them
  - Overwrite: 8 fio jobs randomly pick files created in the write phase and overwrite them
    - Triggers garbage collection as otherwise more than 100 % of the volume capacity would be needed
- Workloads
  - Random file size (4 MiB to 128 MiB) and random buffered I/O size (128 KiB to 4 MiB)
  - Fixed file size (128 MiB) and fixed buffered I/O size (1 MiB)
- Zoned XFS is compared to BTRFS and F2FS
  - Note that F2FS is limited to 16 TiB capacity and cannot support the full 30 TB capacity of the HDD

SMR HDD Write

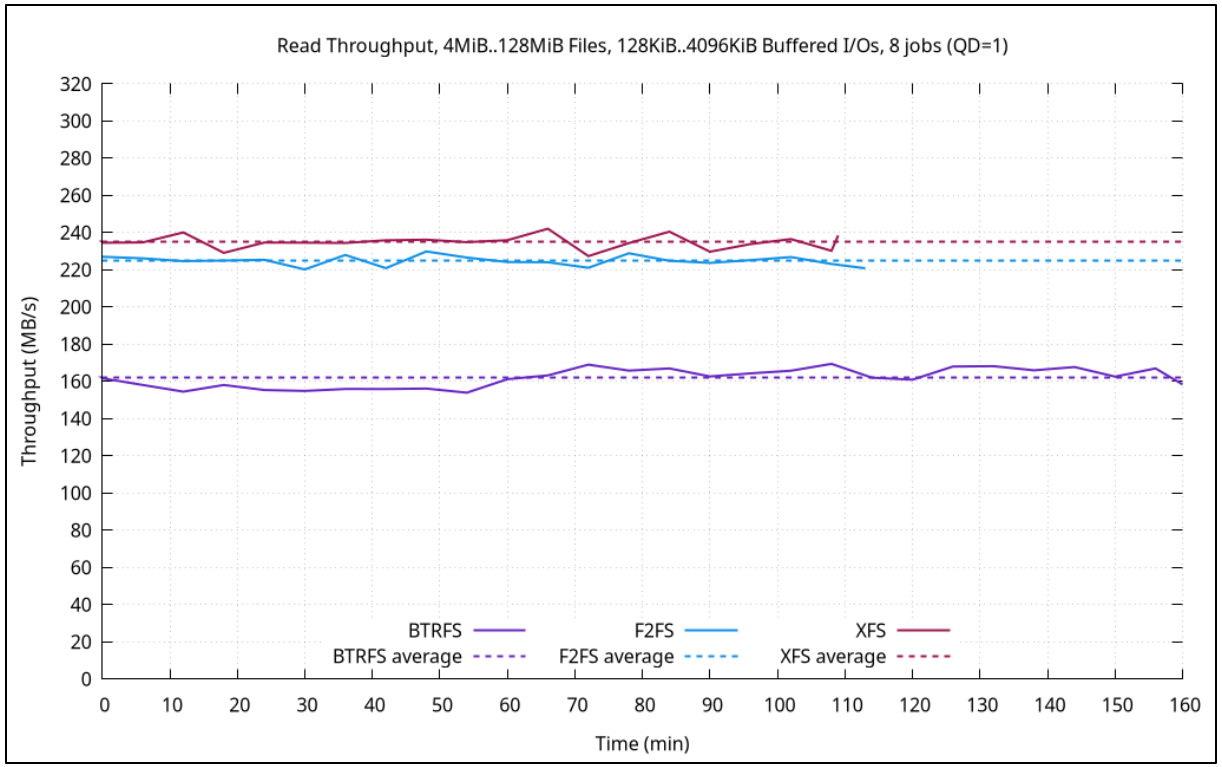
- Zoned XFS provides stable throughput with the highest average
  - F2FS performance sharply drops as it starts GC early (less than 50% usage)
  - BTRFS performance is low due to smaller average size of zone append operations

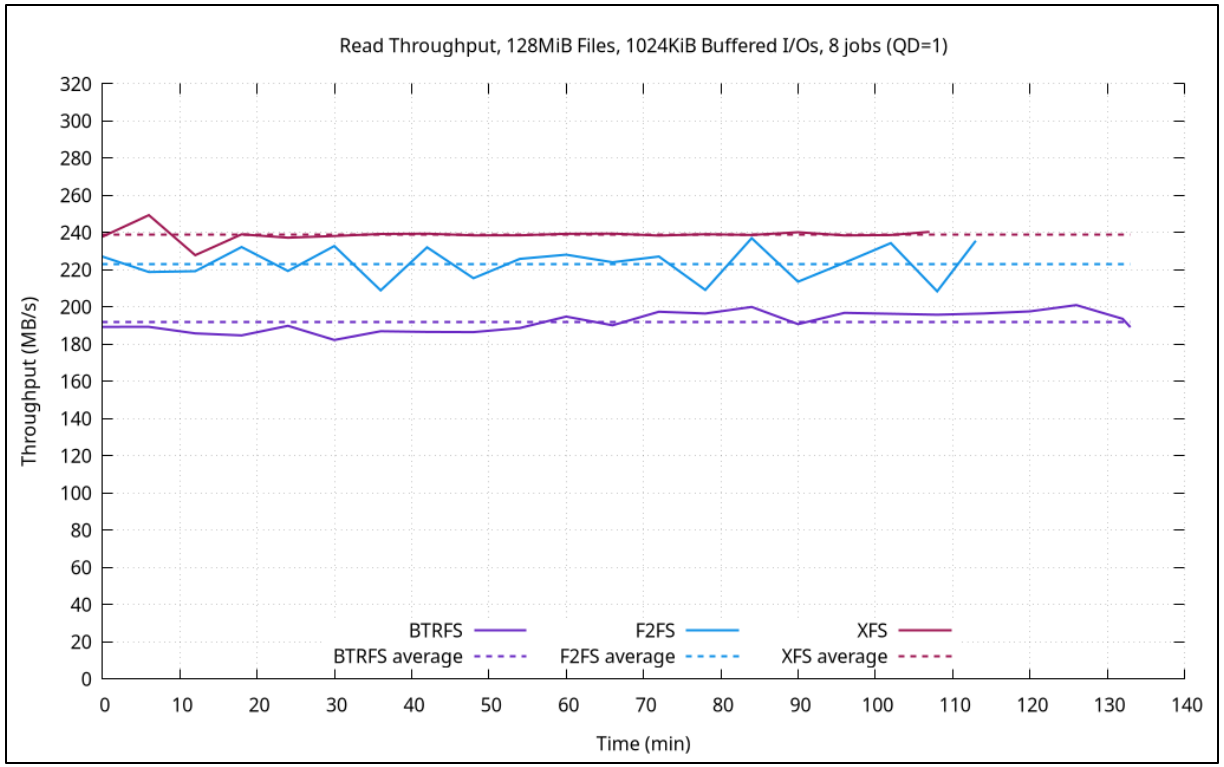




SMR HDD Read

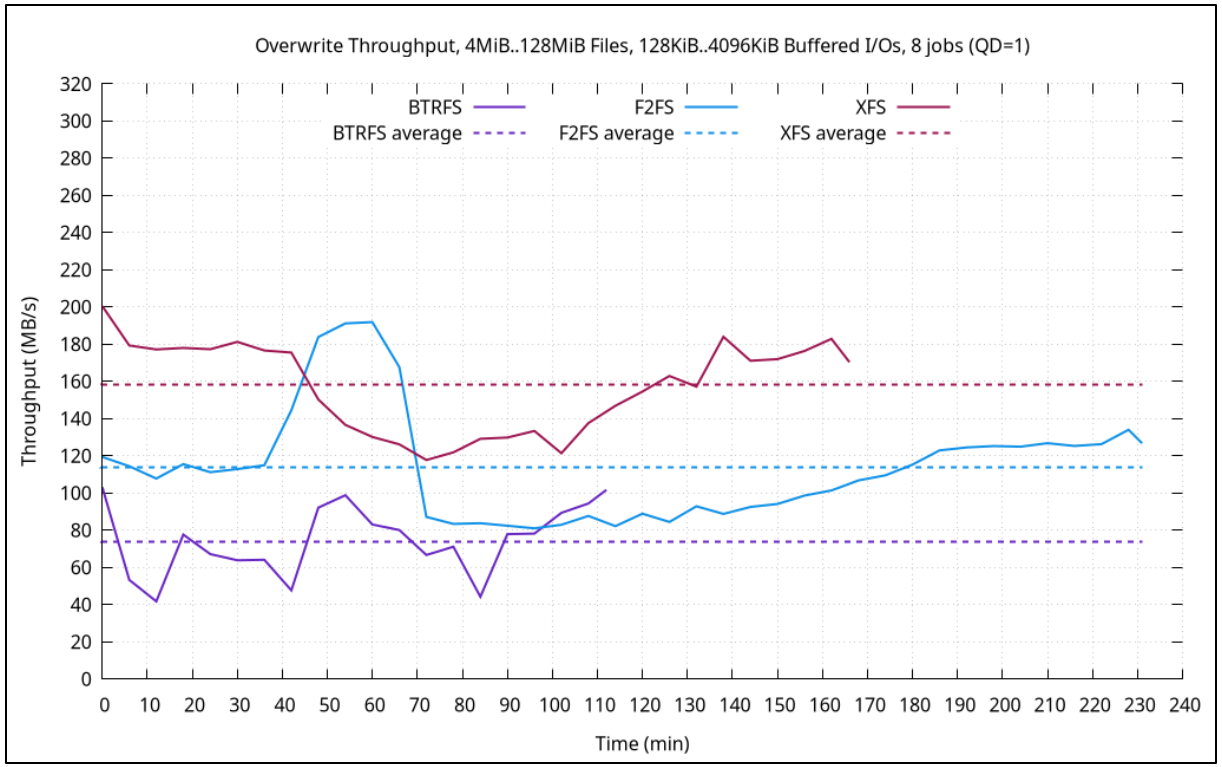
- XFS and F2FS deliver good performance
  - BTRFS has lower throughput due to higher file fragmentation and lack of large folios (large read operations) support

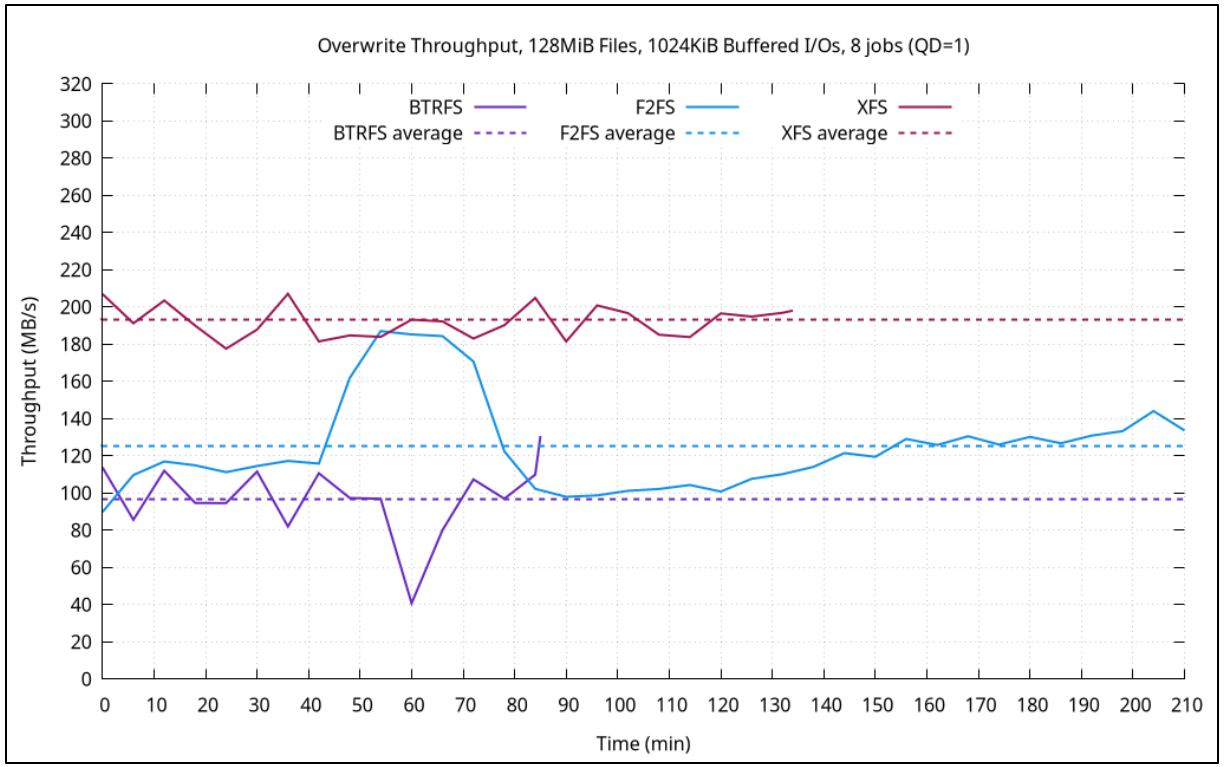




SMR HDD Overwrite

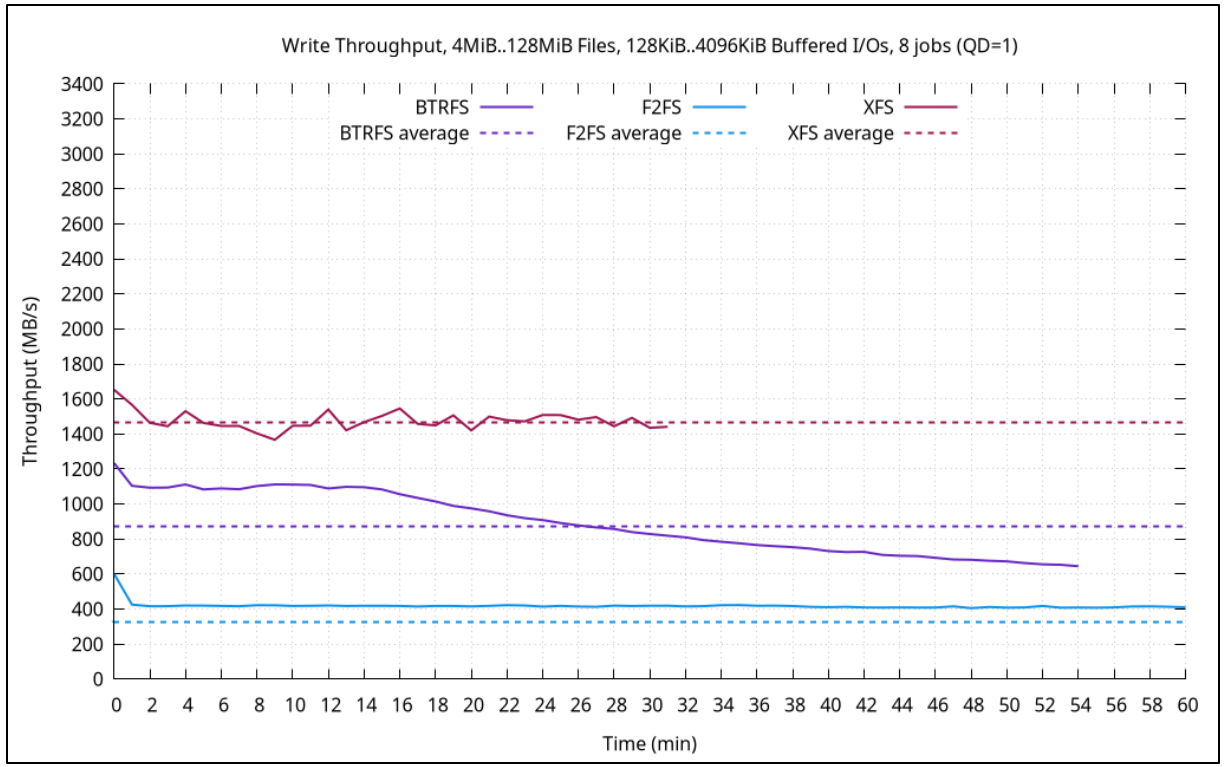
- Zoned-XFS on-demand GC has the smallest overhead, resulting in the highest average throughput
  - Minimal impact for large files workload
- BTRFS fails to overwrite all files due to block allocation and GC synchronization issues

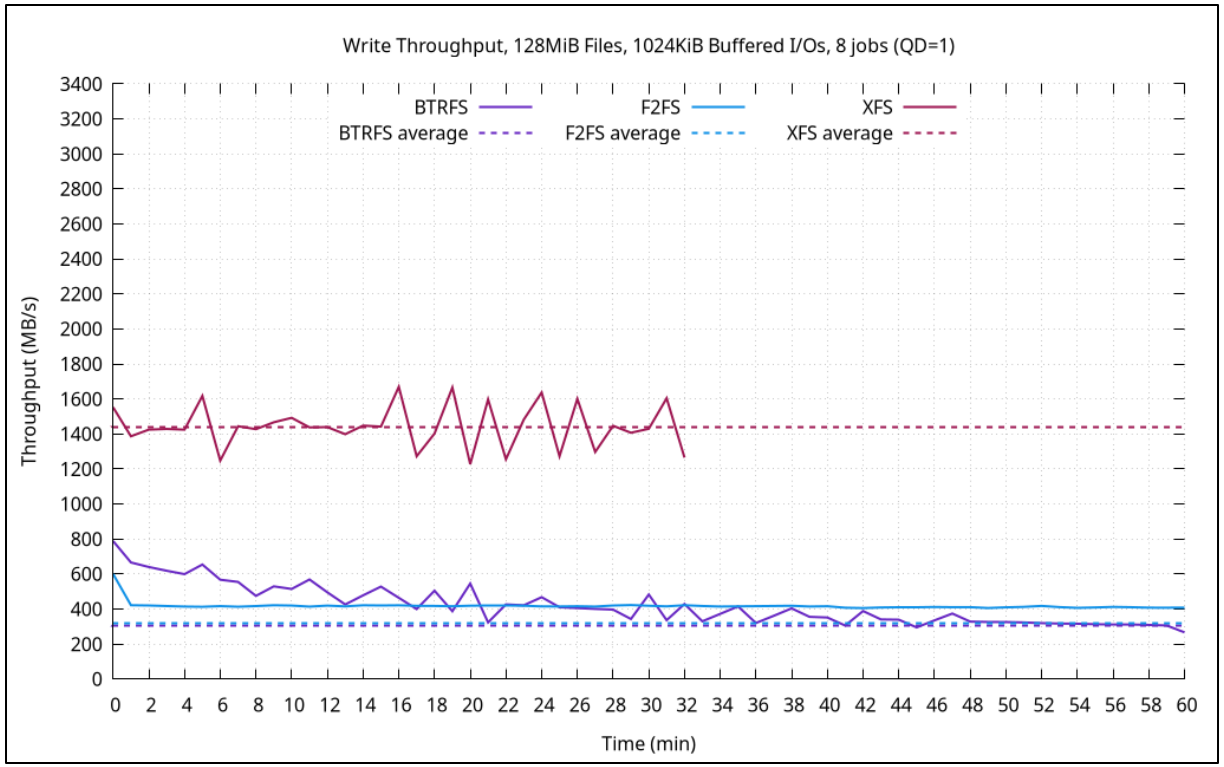




ZNS SSD Write

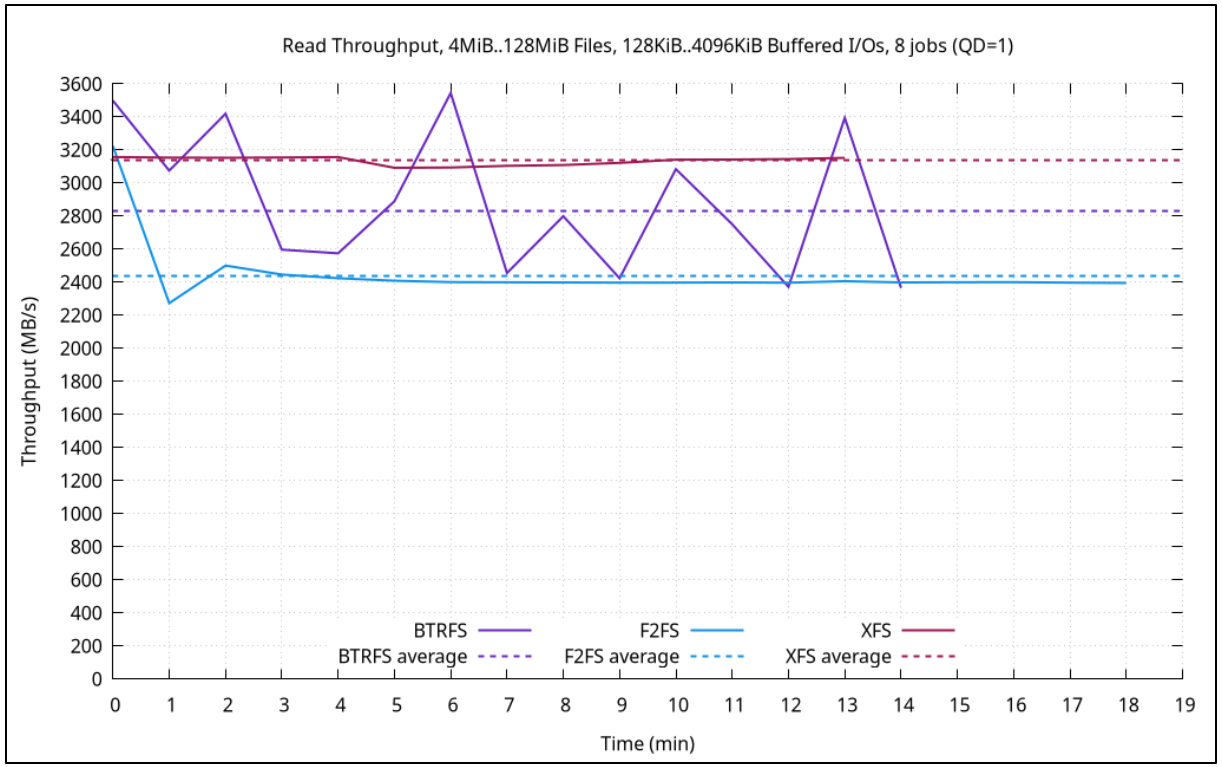
- Zoned XFS provides the highest write throughput
  - BTRFS performance is lower due to smaller average size of zone append operations
  - F2FS performance is the lowest due to the use of regular write operations

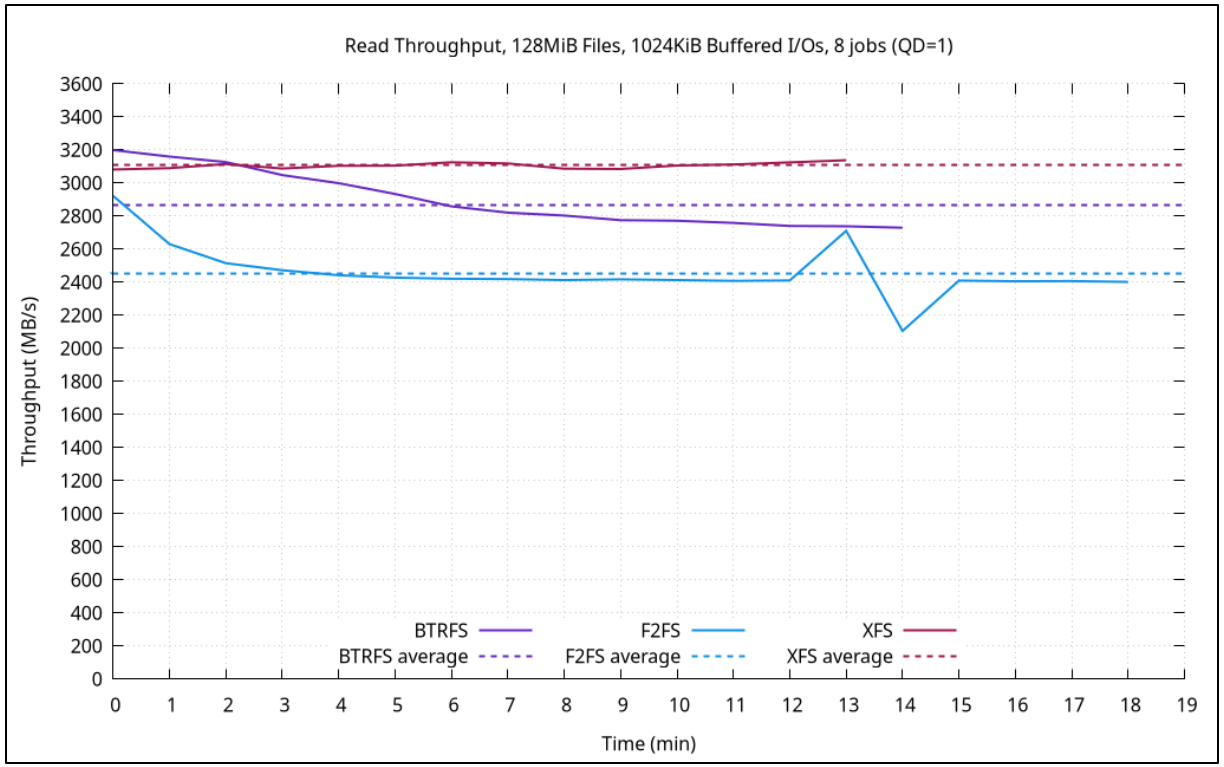




ZNS SSD Read

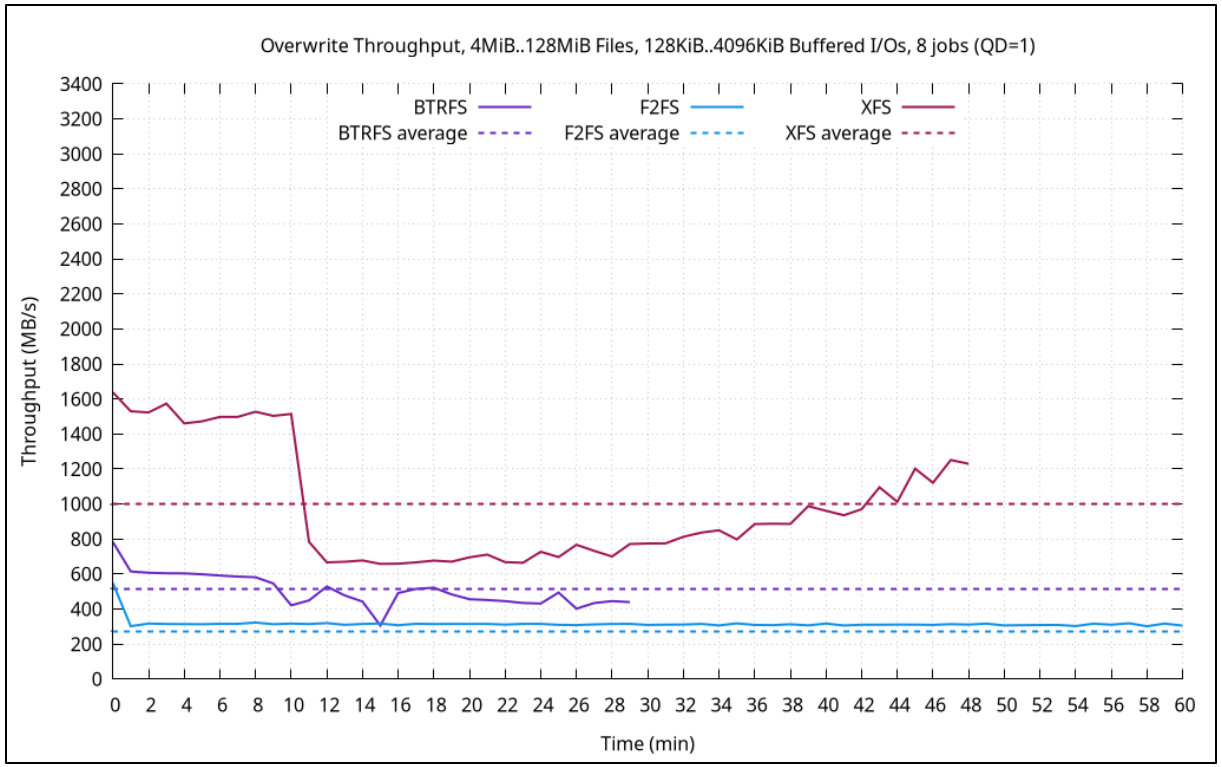
- Good performance with Zoned XFS and BTRFS
  - Zoned XFS performance is more stable and has the highest average
  - F2FS is less efficient and fails to deliver the drive maximum throughpout

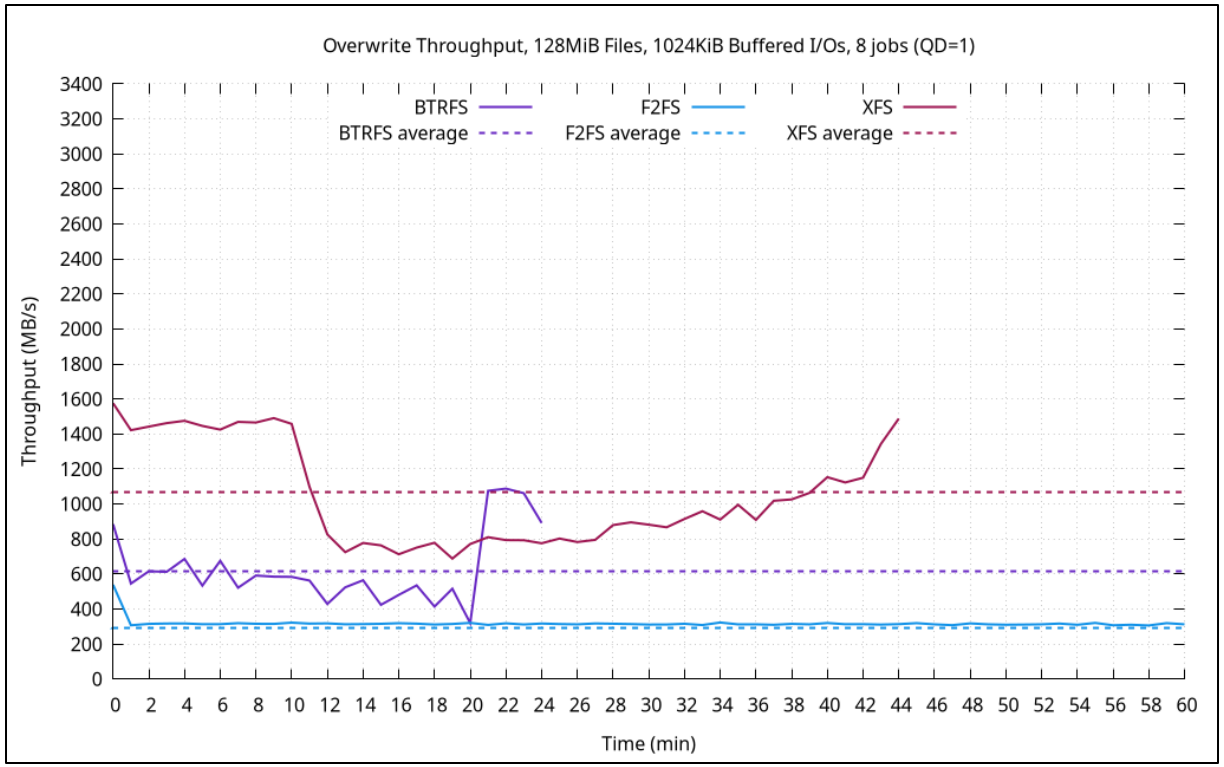




ZNS SSD Overwrite

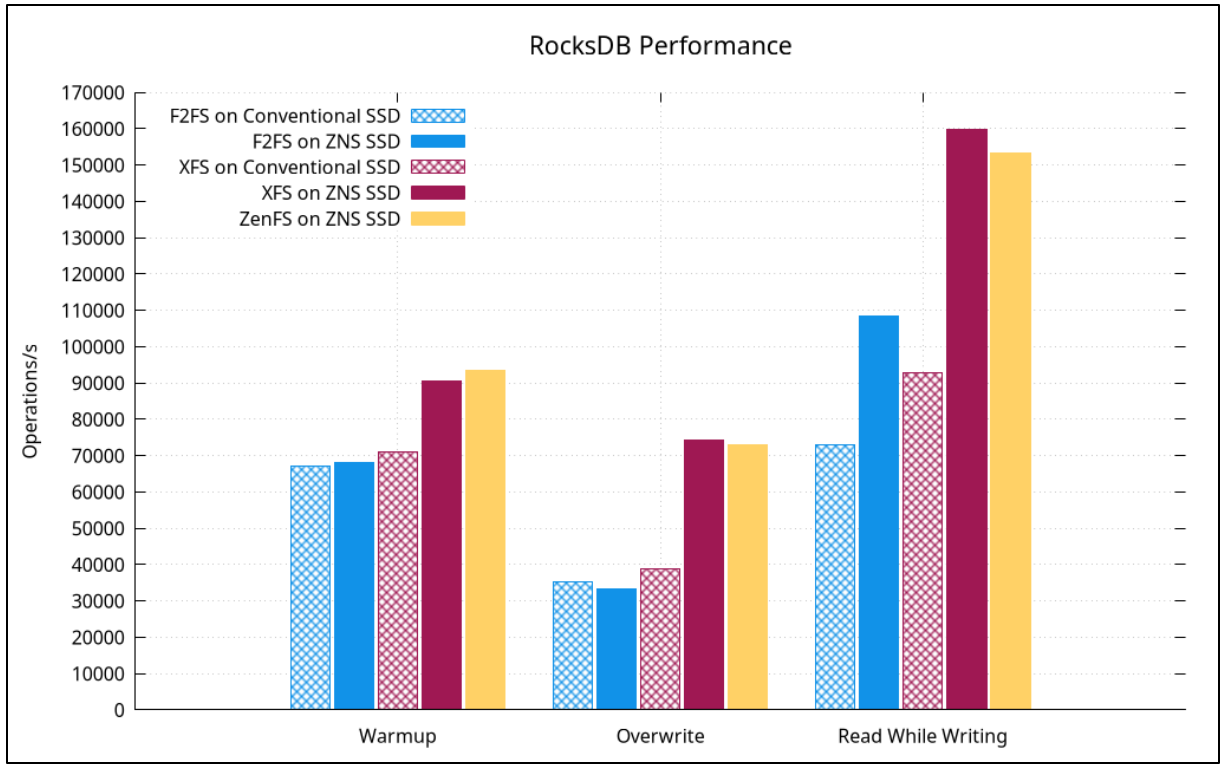
- Zoned-XFS on-demand GC again gives the highest average throughput
  - BTRFS fails to overwrite all files due to ENOSPC errors
  - F2FS use of regular write operations fails to efficiently exercise the ZNS drive

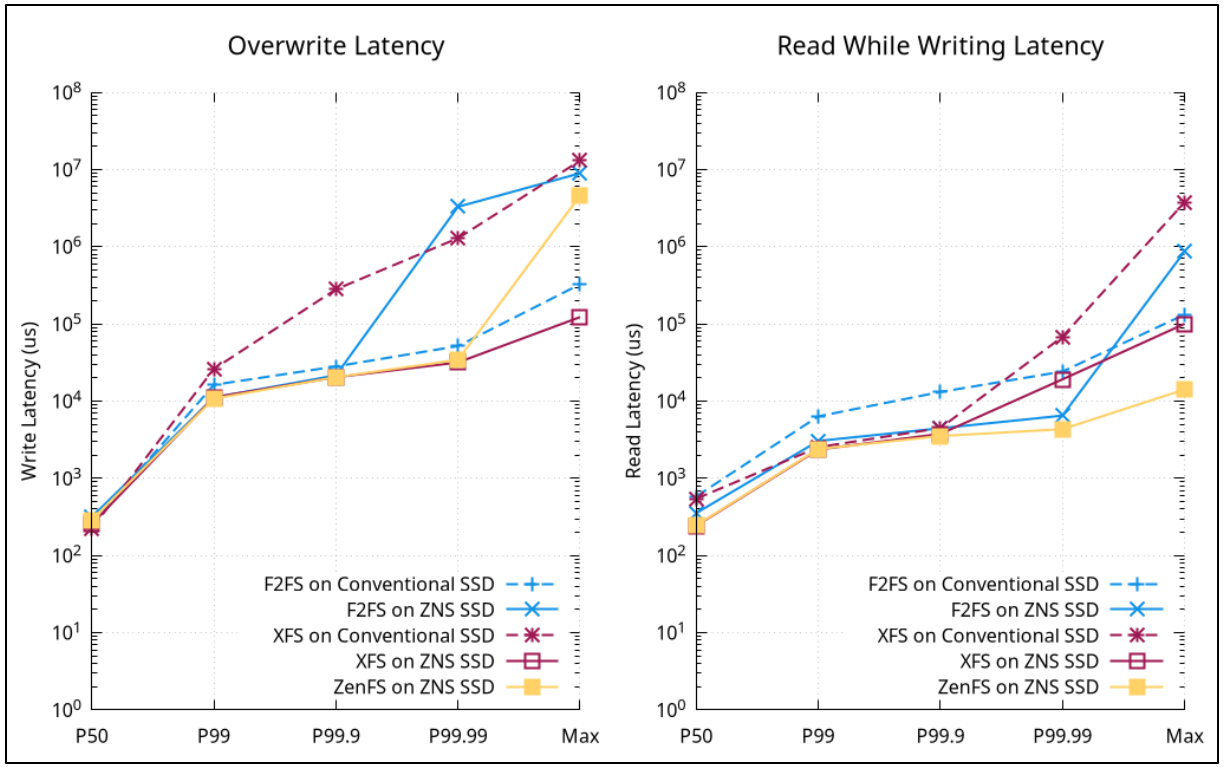




RocksDB Benchmark

- Zoned-XFS achieves performance comparable to the purpose-built and highly optimized RocksDB ZenFS storage backend on ZNS SSDs
  - Competitive throughput and tail latency without application modifications, even compared against a conventional SSD

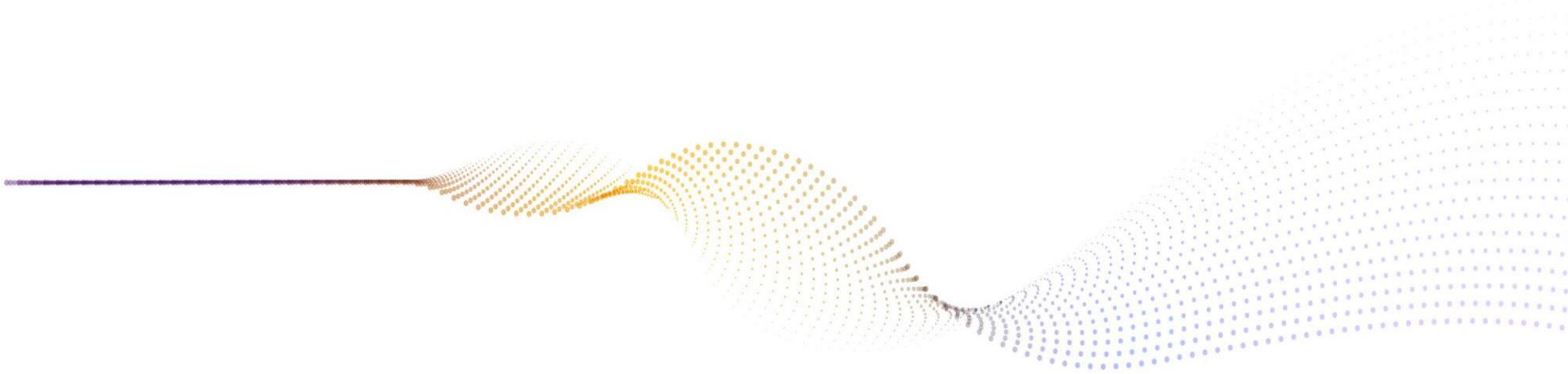




Conclusion

Conclusion

- XFS support for zoned devices is very stable and delivers good initial performance
  - Self-contained support for SMR disks with conventional zones facilitates deployment
  - Performance remains good even under garbage collection conditions
- Areas of improvements have been identified
  - Interaction with the system page cache writeback sometimes causes file fragmentation which impacts GC overhead
  - Improved garbage collection victim selection using a per-zone last-write timestamp
    - Automatic estimation of the lifetime of data within the zone



Thank you for attending!

Please remember to rate this session. You get access the presentations at
http://sniadeveloper.org/conference

Micro Benchmark Test Environment

- Enterprise grade server CPU, 128 GB of DRAM
- Kernel 6.16.0
- 30 TB SATA SMR HDD connected to an AHCI SATA port
  - 111,760 zones of 256 MiB, including 1178 conventional zones from LBA 0
  - 128 open zones limit
  - Only the first 2TB of the disk are used to reduce run times
- 8 TB ZNS U.2 SSD (128 GB M.2 regular SSD as the main device)
  - 3624 zones of 2 GiB (1077 MiB capacity per zone)
  - 14 open zones limit (14 active zones limit)

RocksDB Test Environment

- Dell R7515 server, 16-core AMD Epyc 7302P CPU, 128 GiB of DRAM
- Kernel 6.15-rc7
- 1 TB Western Digital ZN540 NVMe ZNS SSD
  - 453 zones of 2 GiB (1077 MiB capacity per zone)
  - 14 open zones limit (14 active zones limit)
  - Small 4GiB conventional namespace used for metadata
- 1TB Western Digital SN540 conventional SSD for comparison with XFS and F2FS
- RocksDB configuration set to use direct I/Os, 64 threads, and a file size of 1077 MiB,
  - Matching the ZNS SSD zone capacity, to minimize garbage collection
- The warmup workload fills the drives to 75% capacity

