Western Digital.

Zoned XFS

Zoned RT devices support

https://en.wikipedia.org/wiki/File:Under_Construction.jpeg

Hans Holmberg, WDC Research

Project overview

- Zoned XFS has been discussed on and off for a few years
- Project kicked off at ALPSS '24
- Current state: experimental
  - Supports SMR HDDs, ZNS SSDs, Zoned mobile flash and conventional block devices
  - Can handle large benchmark runs
  - Passing 99% of applicable xfstests
- Code

kernel: https://git.infradead.org/?p=users/hch/xfs.git;a=shortlog;h=refs/heads/xfs-zoned

xfs-progs: https://git.infradead.org/?p=users/hch/xfsprogs.git;a=shortlog;h=refs/heads/xfs-zoned

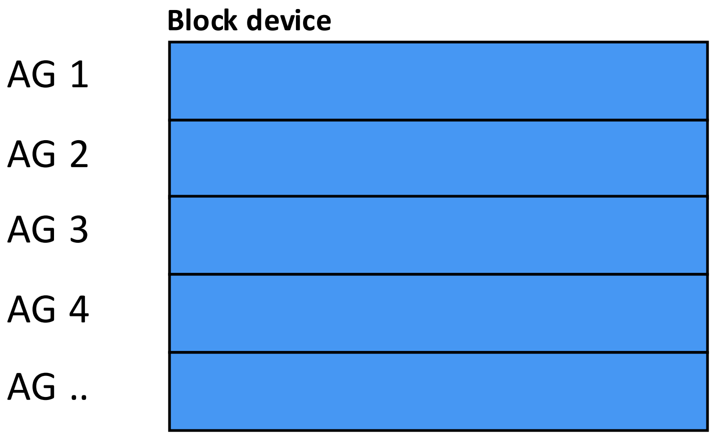
xfstests: https://git.infradead.org/?p=users/hch/xfstests-dev.git;a=shortlog;h=refs/heads/xfs-zoned

Acknowledgements

- **Darrick J. Wong** - XFS Real time dependencies
- **Christoph Hellwig** - Most of the work
- **Damien Le Moal** – Experimentation and tuning for SMR
- **Shinichiro Kawasaki** - Testing
- **Hans Holmberg** – Data placement, work on space accounting, garbage collection

XFS On Disk Data Structures

- **Data**
  - XFS allocates data into Allocation Groups (AGs)
  - Virtual storage regions of fixed size
    - Each AG manages its own set of files and manages its own backing storage
  - Provides both scalability and parallelism
- **Metadata** is stored in two B+ trees
  - One for file attributes and one for storing file extent metadata (file -> data blocks mapping)
  - **Problem: B+ trees is not a good fit for log-structured-writes**
    - Requires in-place updates
    - Incompatible with append-only zoned writes
    - What do we do?



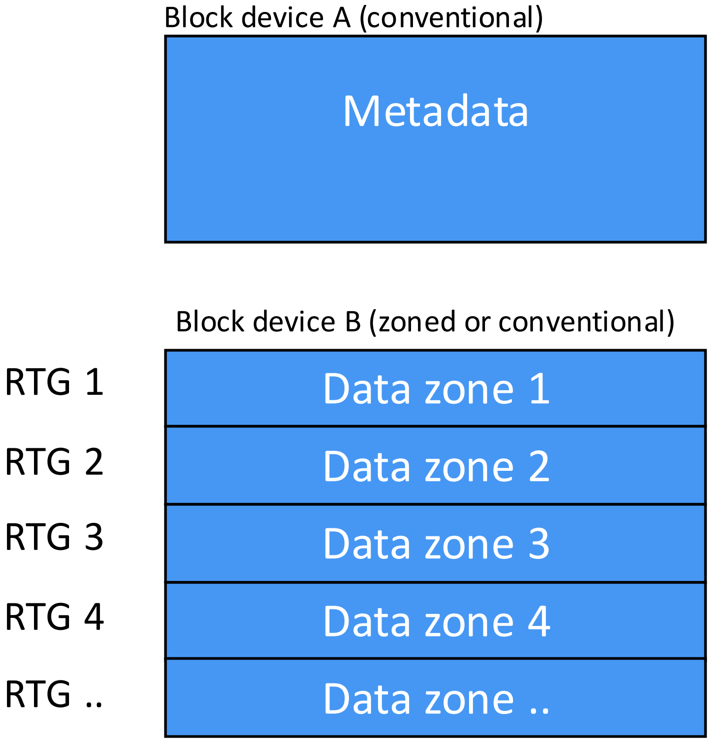
XFS Realtime feature - CONFIG_XFS_RT

- Allows data and meta data on different devices
  - We can keep the B+ trees and focus on data on zoned storage. For now.

- Separates data with realtime access requirements from other data
- All data will be automatically placed on the realtime device with rtinherit=1 mount option

Zoned RT Devices support

- **Depends on conventional block storage for metadata.**
  - Limitation: Space can run out on the metadata block device before the data device and vice versa
  - Conventional zones or conventional name spaces can be used for the metadata, so a separate device is not required
- Maps Real Time Allocation Groups (RTGs) to Zones 1:1
- Utilizes Copy On Write (CoW) to avoid in-place updates for data
- Implements a new data allocator
- Treats zones as buckets and fills them up using zone appends
- **How should we allocate data? What is the best way to do this?**



Data allocation

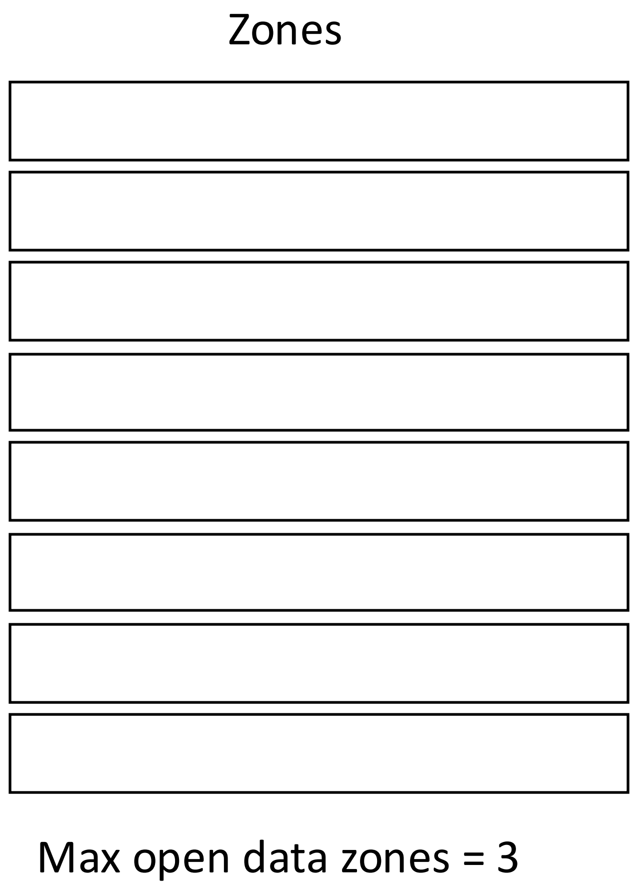
- Zoned storage allows the host to make active choices data placement on the media

- **Opportunity:** reduce write amplification if we get this right
  - Improved performance - less background writes from garbage collection, higher user throughput
  - Reduced media wear , longer media life time - reduced C02 emissions

- **How?**
  - **Minimize fragmentation by separating file data into different zones**
  - When a file is deleted, all of its data will be invalidated on disk..

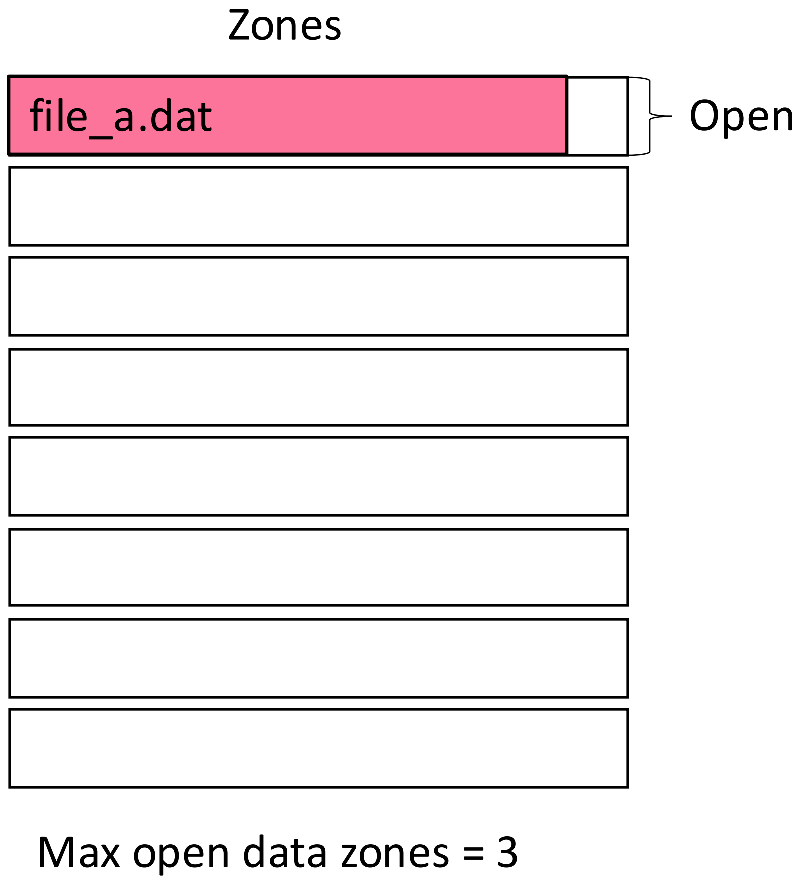
Data placement by file

- Try to minimize fragmentation
- Current (generic) algorithm
  - Keep max N zones open for writes
  - Separate data from different files into different zones as far as possible
  - Use the Least Recently Used zone if an empty zone can not be assigned



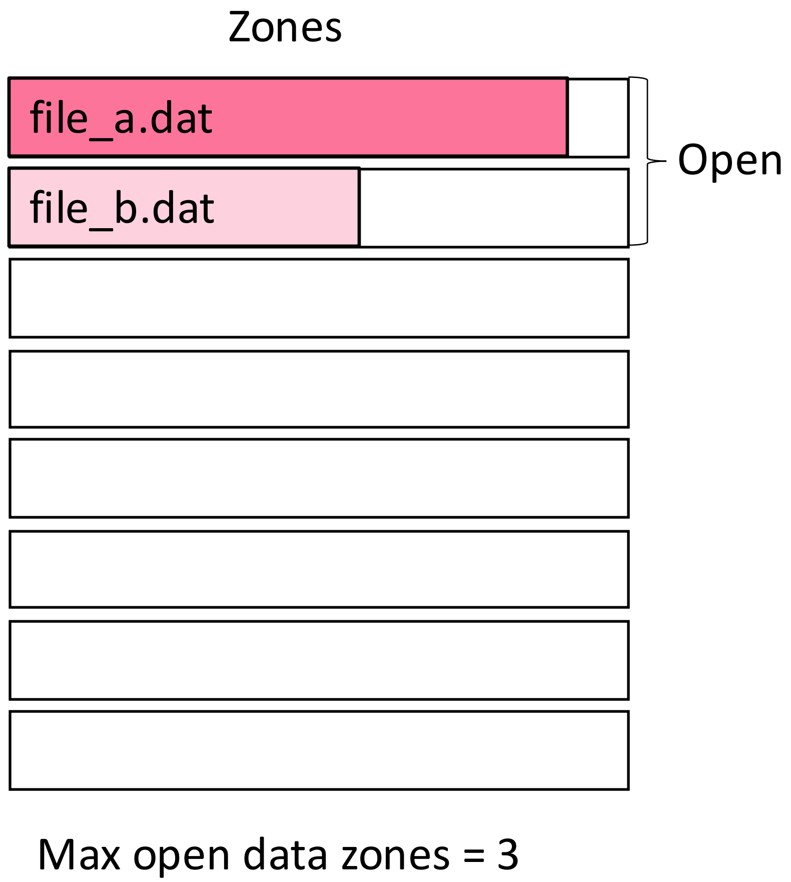
Data placement by file

- Try to minimize fragmentation
- Current (generic) algorithm
  - Keep max N zones open for writes
  - Separate data from different files into different zones as far as possible
  - Use the Least Recently Used zone if an empty zone can not be assigned



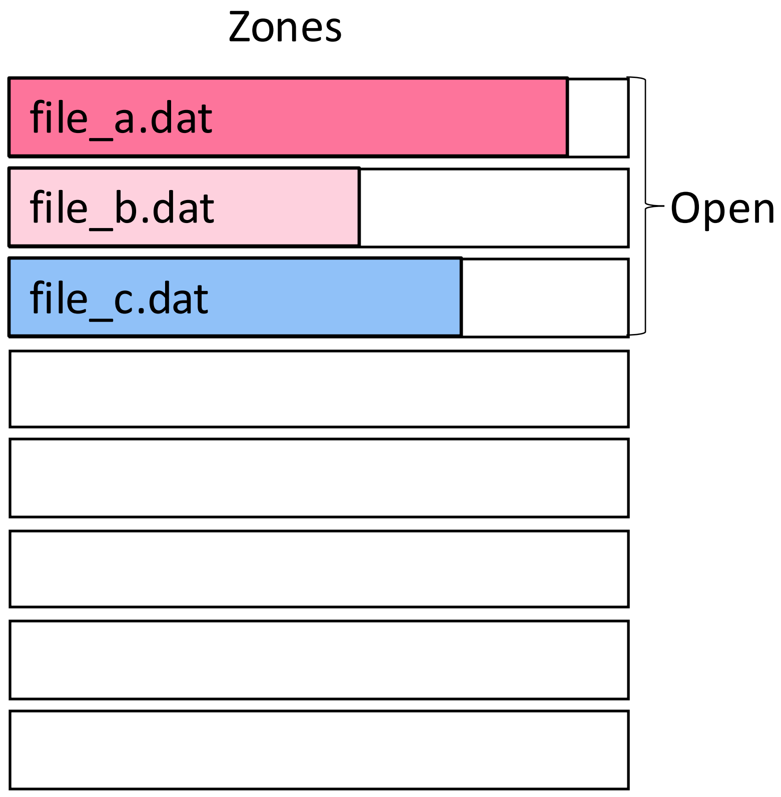
Data placement by file

- Try to minimize fragmentation
- Current (generic) algorithm
  - Keep max N zones open for writes
  - Separate data from different files into different zones as far as possible
  - Use the Least Recently Used zone if an empty zone can not be assigned



Data placement by file

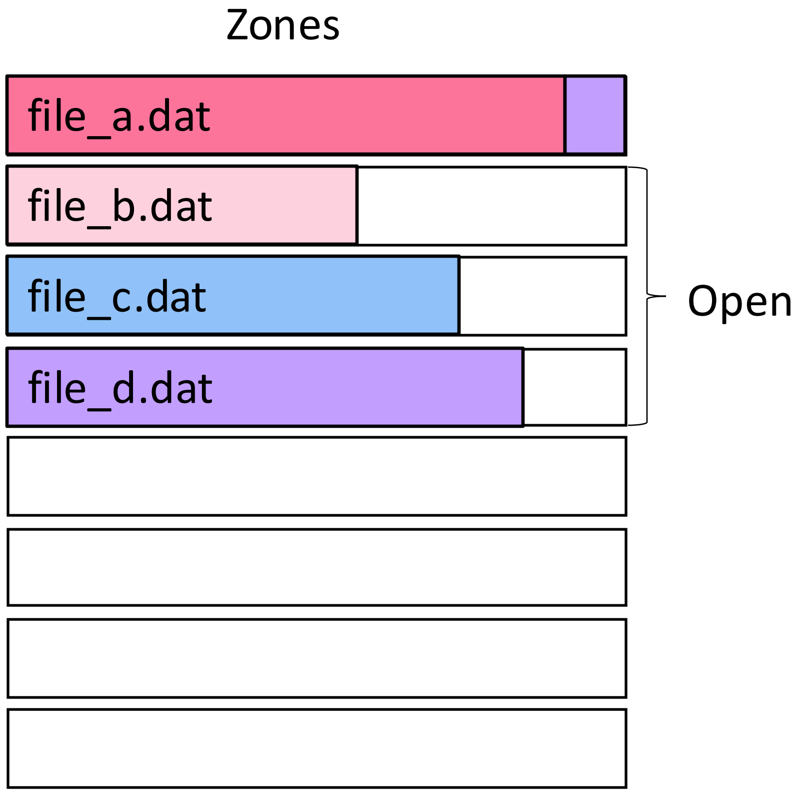
- Try to minimize fragmentation
- Current (generic) algorithm
  - Keep max N zones open for writes
  - Separate data from different files into different zones as far as possible
  - Use the Least Recently Used zone if an empty zone can not be assigned



Max open data zones = 3

Data placement by file

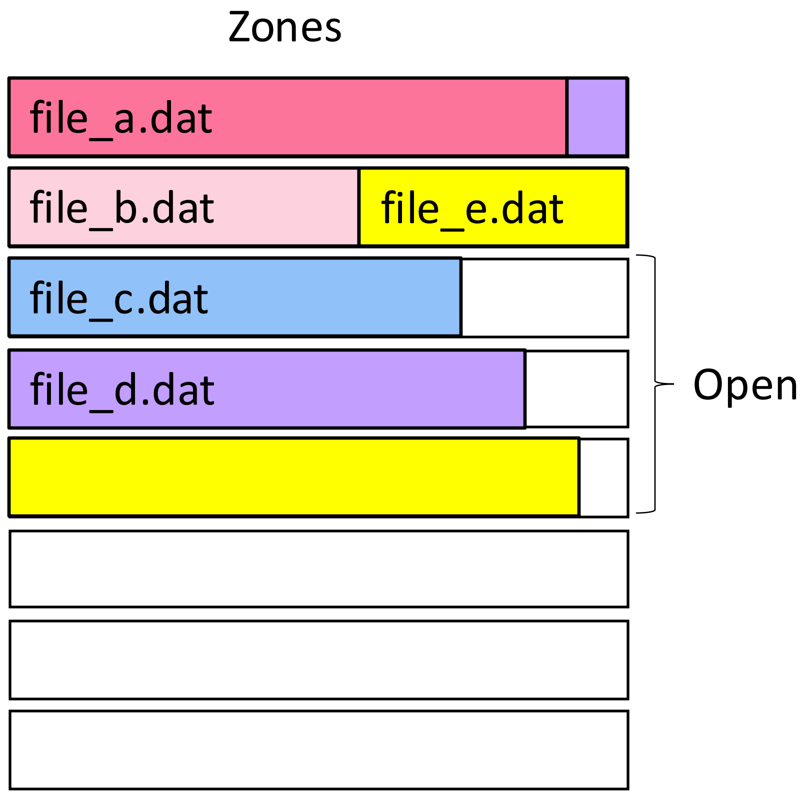
- Try to minimize fragmentation
- Current (generic) algorithm
  - Keep max N zones open for writes
  - Separate data from different files into different zones as far as possible
  - Use the Least Recently Used zone if an empty zone can not be assigned



Max open data zones = 3

Data placement by file

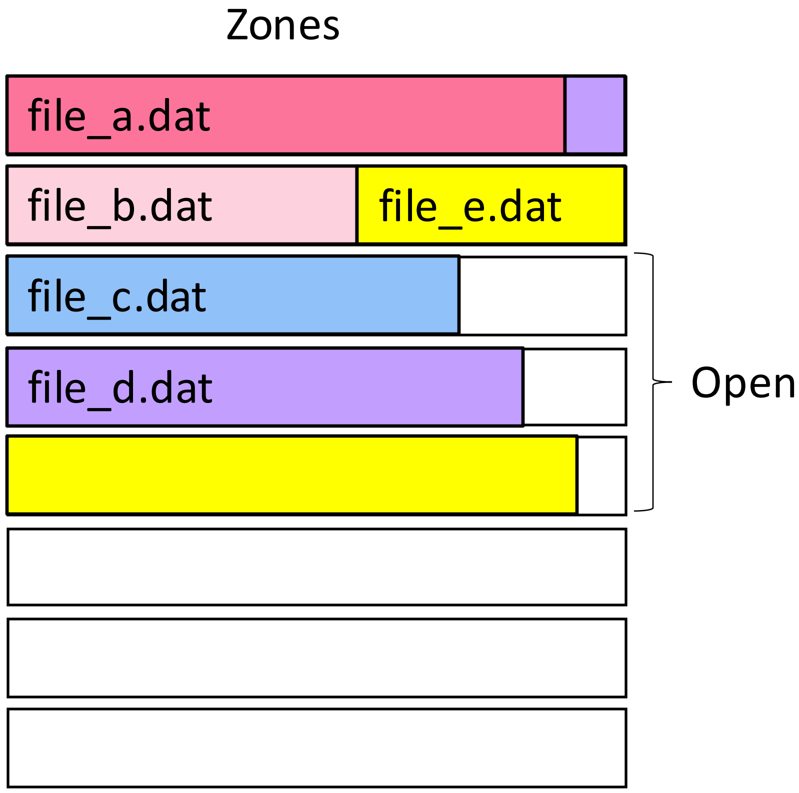
- Try to minimize fragmentation
- Current (generic) algorithm
  - Keep max N zones open for writes
  - Separate data from different files into different zones as far as possible
  - Use the Least Recently Used zone if an empty zone can not be assigned



Max open data zones = 3

Data placement by file

- Try to minimize fragmentation
- Current (generic) algorithm
  - Keep max N zones open for writes
  - Separate data from different files into different zones as far as possible
  - Use the Least Recently Used zone if an empty zone can not be assigned



Max open data zones = 3

Write life time hint support

- We utilize write life time hints when passed by the user
    - `fcntl(fd_, F_SET_RW_HINT, &fcntl_hint)`
- Heuristic
    - Colocate file data if there is a good match between expected life time of the data stored in an open zone with incoming file data
    - Separate file data if there is not a good match
    - Based on statistics from RocksDB
- More workloads needs to be evaluated..

```
static bool
xfs_good_hint_match(
        struct xfs_rtgroup      *rtg,
        enum rw_hint            file_hint)
{
        switch (rtg->rtg_write_hint) {
        case WRITE_LIFE_LONG:
        case WRITE_LIFE_EXTREME:
                /* don't colocate cold data */
                break;
        case WRITE_LIFE_MEDIUM:
                /* colocate medium with medium */
                if (file_hint == WRITE_LIFE_MEDIUM)
                        return true;
                break;
        case WRITE_LIFE_SHORT:
        case WRITE_LIFE_NONE:
        case WRITE_LIFE_NOT_SET:
                /* colocate short and none */
                if (file_hint <= WRITE_LIFE_SHORT)
                        return true;
                break;
        }
        return false;
}
```

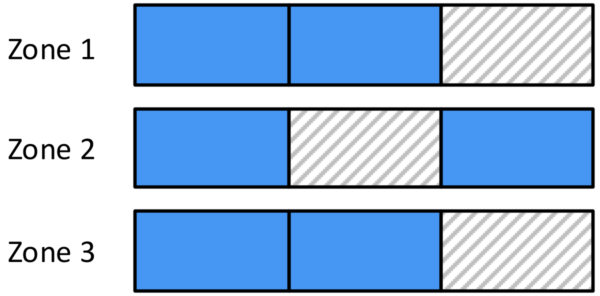
Reclaim

- To reclaim unused written space, garbage collection (GC) is needed
- Current design:
  - **Lazy**
    - Start gc when data separation is impacted
    - Target: keep N open zones open for data placement
  - **Greedy victim selection**
    - The zone with most reclaimable space is picked by the GC daemon
    - Open zones are not garbage collected
  - **Allocate moved data separately from user writes**
    - simple hot/cold separation heuristic

Balancing User and GC writes

- When write pressure and write amplification is high, we need to make sure that reclaim can keep up to avoid running out of free zones
  - We may need to block until free space is available

- **Problem:** the minimum reclaim unit is a zone
  - We may have to run GC over several zones
  - Freeing up a whole zone can take seconds(!) if fragmentation is high
  - Stopping user writes completely during a full-zone reclaim is not OK



Example: Space from three zones needs to be reclaimed to free one zone

- **Solution:** rate-limit user writes
  - Make user writes wait in a queue for free space

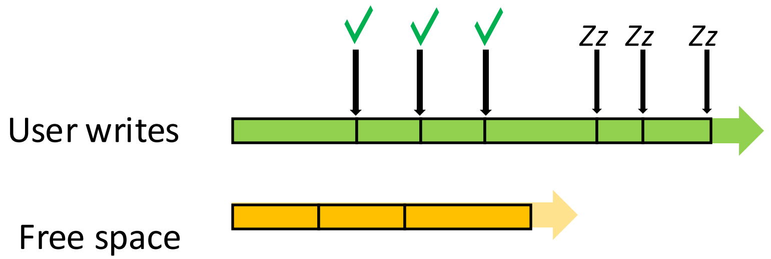
User write throtteling

- When running low on free zones, we awake the GC daemon and make user writes reserve free space before data allocation

- If the reservation cannot be covered by existing free space, the write is put to sleep until enough free space has been reclaimed to cover number of blocks for the incoming write

- Free space is produced by the garbage collection daemon in small chunks as data is being moved of the current zone being reclaimed:
  - reclaimed_space += gc_chunk_size / reclaim_ratio
  - reclaim_ratio is needed to compensate for current level of fragmentation / write amplification

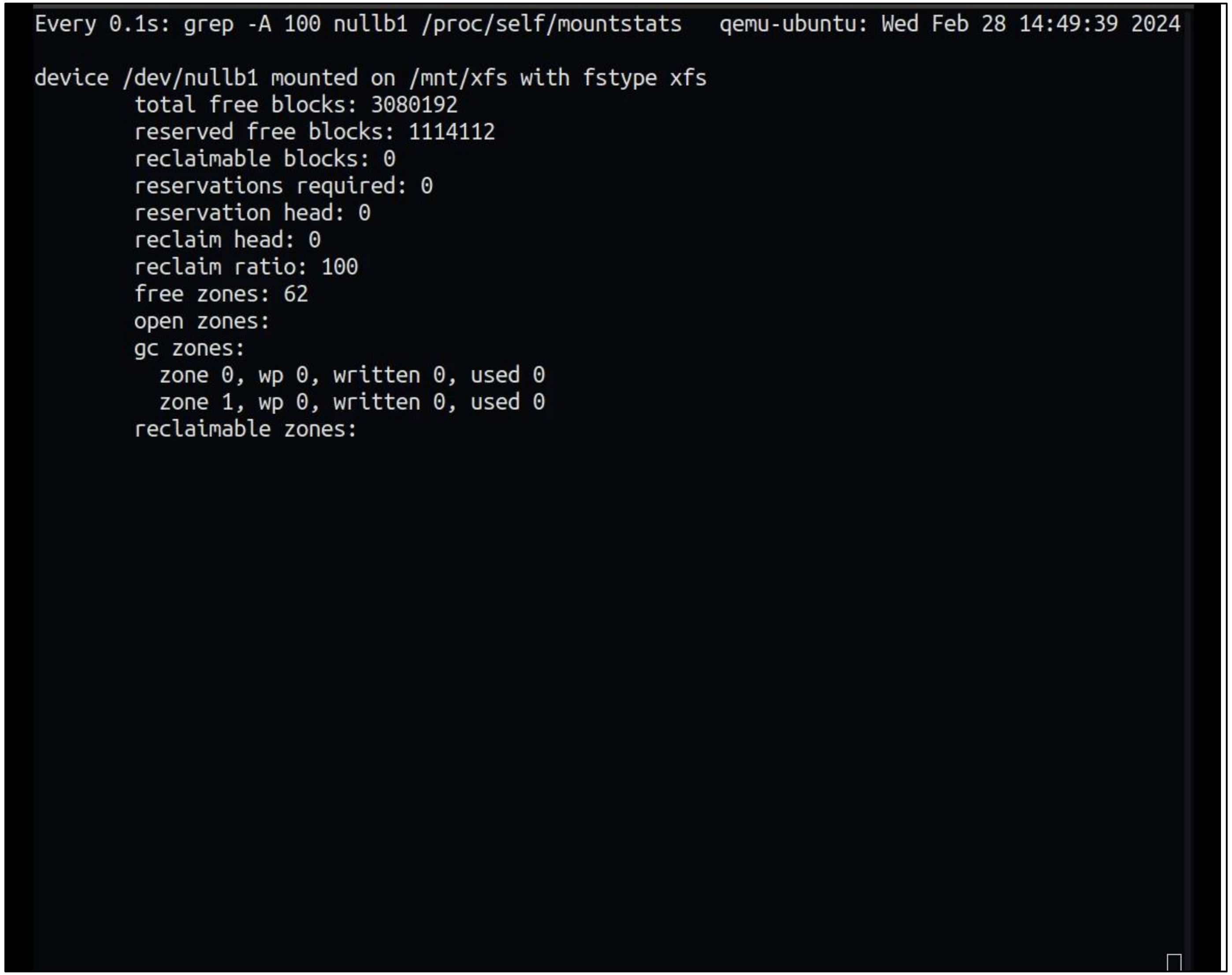
- **Updating the reclaimed space counter in small increments avoids large spikes in write latency**



*Example: The GC daemon needs to move two blocks to free up one block. **Reclaim_ratio = 0.666..***

Demo: gc stress test

- Virtual nullblk device 64 zone, max 16 open
  - 15 data zones
  - one dedicated to GC writes
- **Phase 1: fillup**
  - The workload generates files of random size until the file system is filled to 95%
- **Phase 2: mixed write/delete**
  - New files of random size are being written
  - Random existing files are deleted in parallel to stay at 95% file system utilization
- xfstests-dev/tests/generic/747

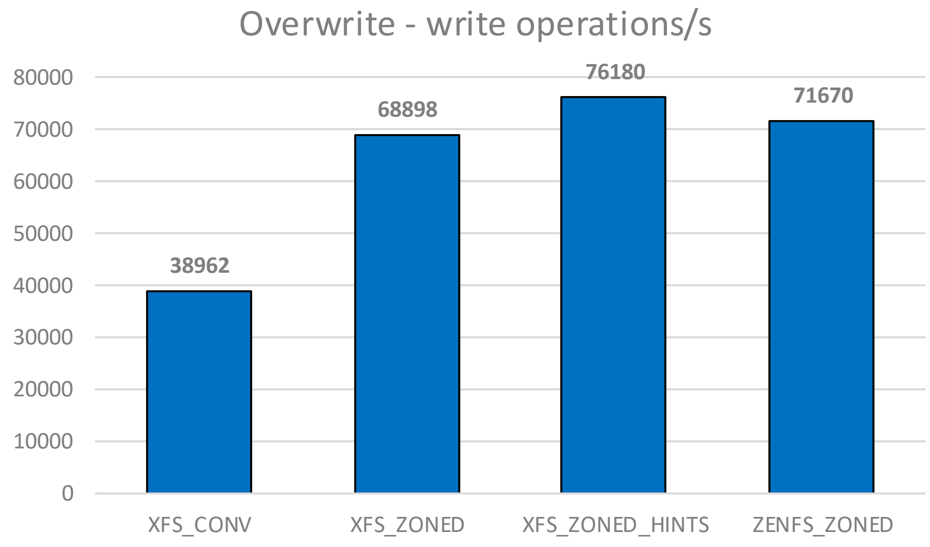


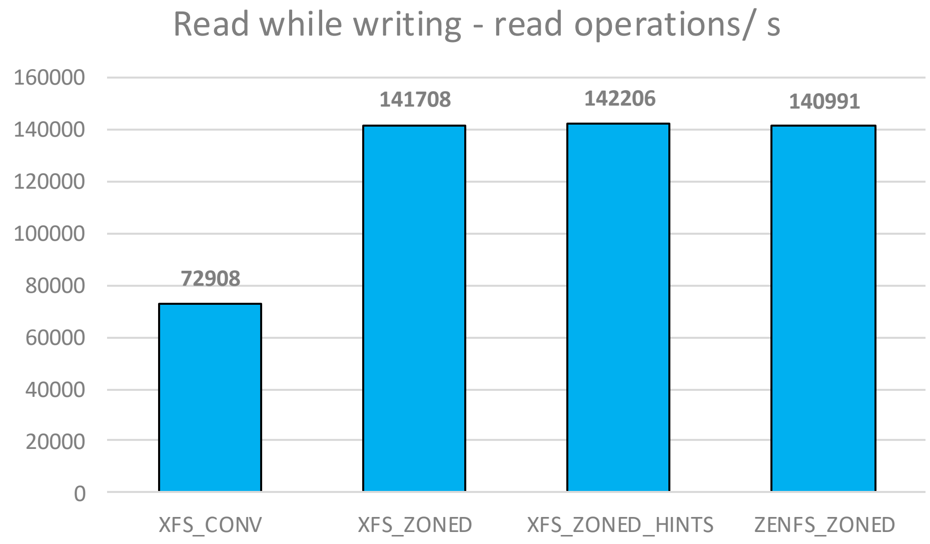
```
Every 0.1s: grep -A 100 nullb1 /proc/self/mountstats   qemu-ubuntu: Wed Feb 28 14:49:39 2024

device /dev/nullb1 mounted on /mnt/xfs with fstype xfs
        total free blocks: 3080192
        reserved free blocks: 1114112
        reclaimable blocks: 0
        reservations required: 0
        reservation head: 0
        reclaim head: 0
        reclaim ratio: 100
        free zones: 62
        open zones:
        gc zones:
          zone 0, wp 0, written 0, used 0
          zone 1, wp 0, written 0, used 0
        reclaimable zones:
```

Early data placement benchmarks

- Setup
  - RocksDB 7.6.3 using direct IO
  - db_bench filluniquerandom, overwrite, readwhilewriting
  - 80% drive utilization
  - 1TB Conventional vs Zoned SSDs (same raw read/write performance)
  - XFS Zoned kernel dev branch (v6.10.0 - edition)

- Performance improvements for write workloads
  - Data-placement-by-file very effective when matching file size with zone capacity
  - Reduced reclaim increases user read and write throughput (2x)
  - Write-life-time-hint support adds ~10%
  - On par, or better than ZenFS (used as a reference – 1.0 write amp.)





What Comes Next?

- Upstream
  - outstanding xfs-realtime dependencies
  - zoned additions in fs/xfs/ (~3k lines added)
  - xfstools (mkfs..)
  - xfstests (infrastructure, new tests)
  - Look out for the RFC!
- More benchmarking
- Start thinking about log-structured metadata..

Western Digital®

Create What's Next