OpenZFS

Sequential Reconstruction

Mark Maybee

Hewlett Packard Enterprise

Topics

OpenZFS

- History
- Resilver Taxonomy
- Healing Resilver
- Sequential Resilver
- Why Sequential Resilver?
- So why Healing Resilver?

History

OpenZFS

This feature was originally developed by Isaac Huang as part of his work on dRAID. It was later extended, hardened, and integrated by Brian Behlendorf (with some help by Mark Maybee).

Taxonomy

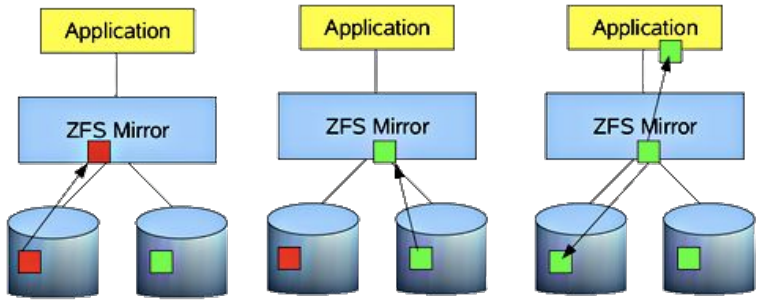
OpenZFS

- Traditionally, data reconstruction in ZFS uses the term Resilver
  - First introduced for Mirrors, but used for RAIDZ as well
  - Driven by a traversal of the pool block pointer tree
- Sequential reconstruction was introduced with dRAID and called Rebuild
  - Common term used in the declustered RAID community
  - Driven by a traversal of space allocation maps
- Reconstruction driven by the block pointer tree is now called Healing Resilver
  - Reconstruction data reads are verified by block pointer checksums
- Reconstruction driven by the space maps is now called Sequential Resilver
  - Reconstruction progresses sequentially through the space allocation maps

Healing Resilver

OpenZFS

- Named for the “self healing” read technology embedded in ZFS
- Works on all data layouts (mirror and raidz)
- Traverses block tree to find data to reconstruct
  - Can trim traversal using block time stamps
  - Very efficient for small outages (e.g., drive offline for a few minutes)
- Problem: can be slow on aged pools
  - Small random IO is expensive
- Enhanced to “batch” IO’s together
  - Batches processed in sequential order
  - Batch size limited by memory size
- Better, but resilver can still be slow



Sequential Resilver

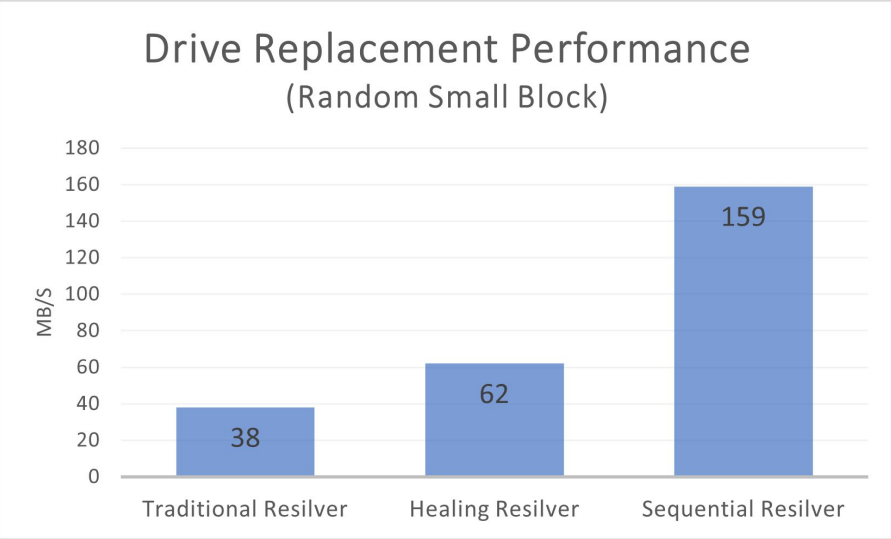
OpenZFS

- Sequentially walks space allocation maps to find the “used” parts of the drive
  - Breaks allocation ranges into sequential blocks
  - Issues Reads/Writes in large blocks
- Must be able to reconstruct without block pointer data
  - Reconstructs based off of knowledge of data layout
  - E.g., Mirror reconstruction is just a copy
- Problem: no block pointer data
  - Does not verify reads (has no checksums)
  - Does not work with RAIDz layout (no block boundaries to locate parity)

Why use Sequential Resilver?

OpenZFS

- If recovery speed is critical
  - It always is -- reduce window of vulnerability
  - Sequential resilver can be significantly faster than healing resilver
- Always a good idea with 2-way mirror
  - Verifying the read data does not change the results
- Need a scrub
  - Detect/correct data errors
  - Triggered automatically at end of resilver
  - You should always scrub regularly anyway



So why use Healing Resilver?

OpenZFS

- Short drive outages can be rapidly recovered
  - Most of the drive data content is still correct
  - Just need to fill in missed writes
  - Block tree traversal is very efficient way to locate data associated with a transaction range
- No need to scrub following reconstruction
- RAIDZ
  - Block pointer content is required to be able to map data to parity
  - Parity is required to reconstruct missing data
- What about dRAID?

QUESTIONS?