Lustre* I/O Performance on ZFS

Jinshan Xiong

April, 2016
Agenda

- Lustre* on ZFS
- Lustre performance on ZFS updates
- Review ZFS I/O Performance
  - Follow up ZFS slides from SDSC last year
- Future work
Lustre* on ZFS

- Why ZFS?
  - Superb write performance; writes are always sequential in ZFS
  - Always on-disk persistent
  - Built-in disks management
    - RAIDZ, mirror, etc.
  - Built-in block checksum
  - Built-in data scrub support
  - Metadata are duplicated for redundancy
  - ...
Latest ZFS I/O Performance – Hardware

- Network: FDR IB
- JBOD: 12GB SAS HBA
  - Disk raw throughput: 160MB/s
- OSS RAM: 256GB
Latest ZFS I/O Performance – OBDFilter

- 10 OSTs - 9+2 RAIDZ2
- Single disk raw throughout
  - Write: ~170 MB/s
  - Read: ~190 MB/s
- Community release 2.8
- ZFS-0.6.4-92; record size: 1M
- RHEL 7.2
- Results
  - Write: 90 data disks deliver ~13GB/s

Theoretical Max Write =15300 MB/s
Latest ZFS I/O Performance - Lustre* Clients

- 10 OSTs, 9+2 RAIDZ
  - 110 disks in total, 90 data disks deliver 11GB/s

- ZFS 0.6.4-92
  - 1MB record size
  - 4KB sector size
  - Why? LU-7404

- Lustre 2.8
ZFS Read Problems

- No file aware block allocation
  - Blocks written sequentially may spread around the whole pool
  - Lots of disk seek to read them back
- This is why read is usually slower than write
- Bigger block size would mitigate this problem
Tickets Status Review

- Patches that have been landed into 2.8
  - LU-4820, LU-5278, LU-6038, LU-6152, LU-6155

- In progress: LU-7404
  - Identified commit ‘Illumos 5497 - lock contention on arcs_mtx’ caused I/O timeout problem
  - Still work with upstream developers
  - This is why 2.8 used ZFS-0.6.4.2
Fast Checksum Computation

- Use AVX2 to compute Fletcher-4 checksum
- Compute RAIDZ parity with AVX2 is also in progress

Help is on the way!
- Work started on AVX(2) optimizations for checksums
- Hoping to see this extended to parity

Throughput of Fletcher 4 CPU vs. SSE
Work in Progress

- Development in progress for CORAL project
  - Large block size
  - Parity Declustered RAID - dRAID
  - Separate MD Allocation Class

- All work being upstreamed to ZFS-on-linux project when completed
  - Will likely become available Lustre* ~2.11 Community Release
ZFS 16M Block Size

- ZFS now supports up to 16MB block size
  - Lustre will support 16M RPC size to ensure large block size for ZFS
  - Problems with ZFS memory management
    - Large ARC data buffers are vmalloc() based slabs
    - Use scatter/gather page list to store ARC data
    - Compressed ARC buffer may help a little bit

- Dynamic block OSD-ZFS size is necessary to reduce overhead on small files
  - Avoid the overhead of read-modify-write
  - Application can set block size
  - OSD-ZFS can choose block size by I/O pattern
Why Large Block Size?

- Considering a 8+2 RAIDZ2 again
  - For a 1MB block size, every data drive will store 128KB data
    - Small I/O hurts performance
  - With 16MB block size, we can guarantee 2MB data on each drive
- Deliver better read performance
ZFS dRAID

- Faster rebuild/resilver time
  - Spare blocks are distributed over all disks
  - Short time leads to less risk on data loss
    - 2\textsuperscript{nd} or 3\textsuperscript{rd} disk failure during rebuild time

- Reasonable throughput in degraded mode
  - Lost one disk -> lose 1/N disk bandwidth

- Permutation development based on randomly generated initial permutation
Separate MD Allocation Class

- Metadata blocks are with smaller size, and accessed more frequently
- A dedicated VDEV with high IOPS drives to store metadata
  - SSD or NVRAM
  - Mirrored for redundancy
- Better use of SSD than L2ARC
Why Separate MD Class?

- Loading metadata faster helps deliver better I/O performance
  - Lower read latency
  - Faster scrub/resilver

- Considering a 8+2 RAIDZ2 device
  - Metadata block size varies from 512B to 16KB in ZFS
  - For a 16KB metadata block, 8 data disks will store 2KB on each
  - Small I/O hurts read perf due to 2KB read from each disk for a data buffer