OpenZFS Performance Improvements

LUG Developer Day 2015

April 16, 2015

Brian, Behlendorf

Lawrence Livermore National Laboratory

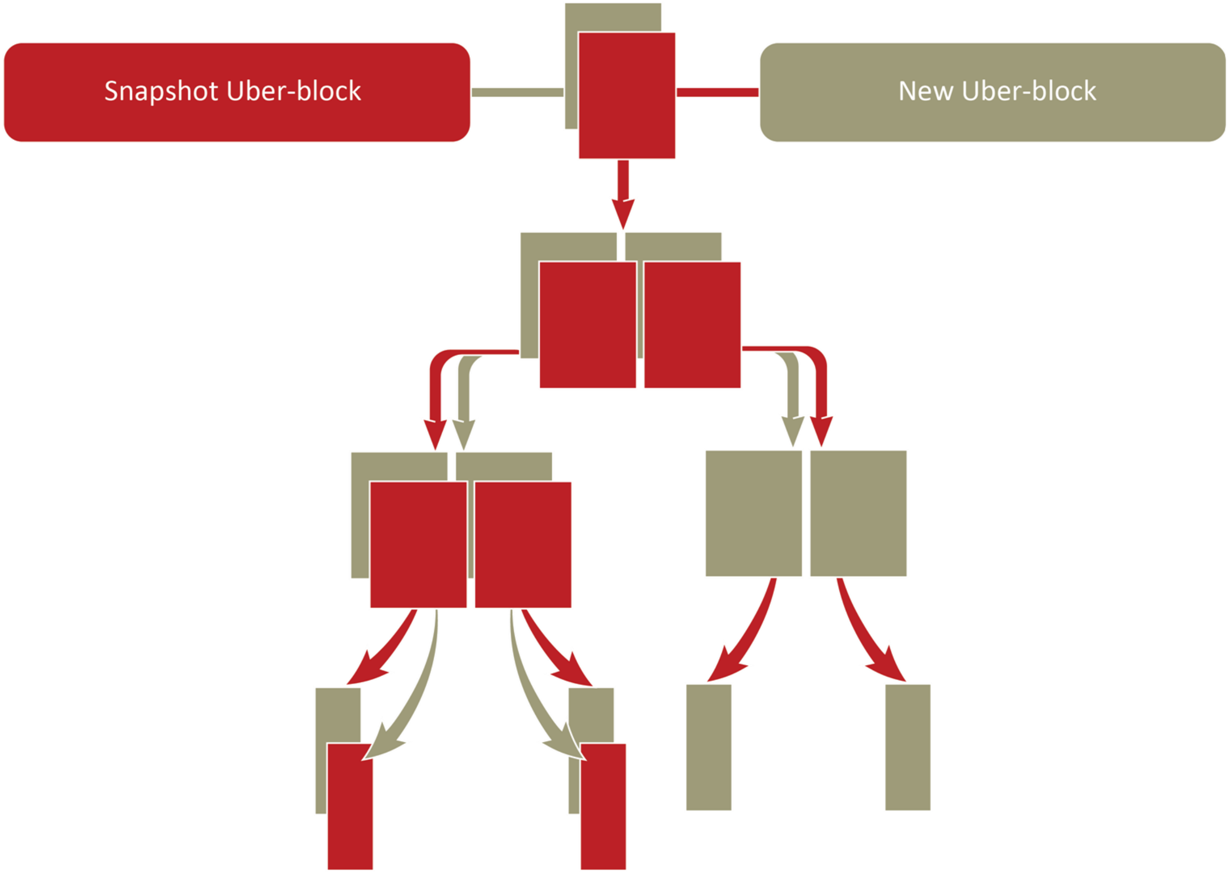
LLNL-PRES-669602

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Agenda

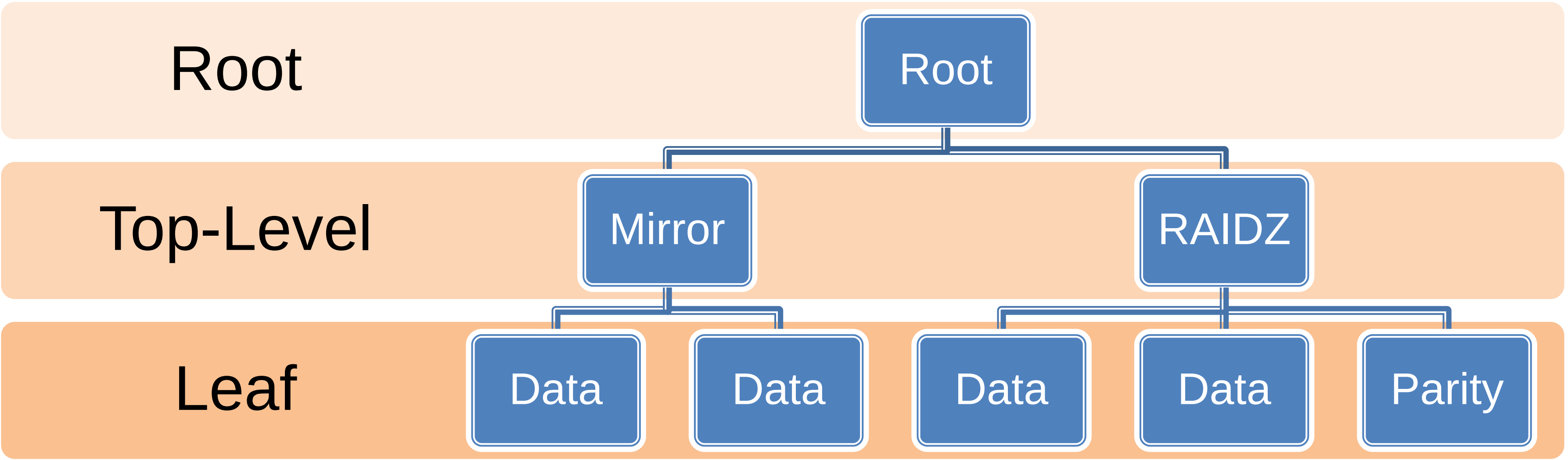
- IO Performance
  - Large Block Feature
- Meta Data Performance
  - Large DNode Feature
- Planned Performance Investigations

IO Performance – Large Blocks



All objects in ZFS are divided in to blocks

Virtual Devices (VDEVs)



Where blocks are stored is controlled by the VDEV topology

Why is 128K the Max Block Size?

- Historical Reasons
  - Memory and disk capacities were smaller 10 years ago
  - 128K blocks were enough to saturate the hardware
- Good for a wide variety of workloads
- Good for a range of configurations
- Performed well enough for most users

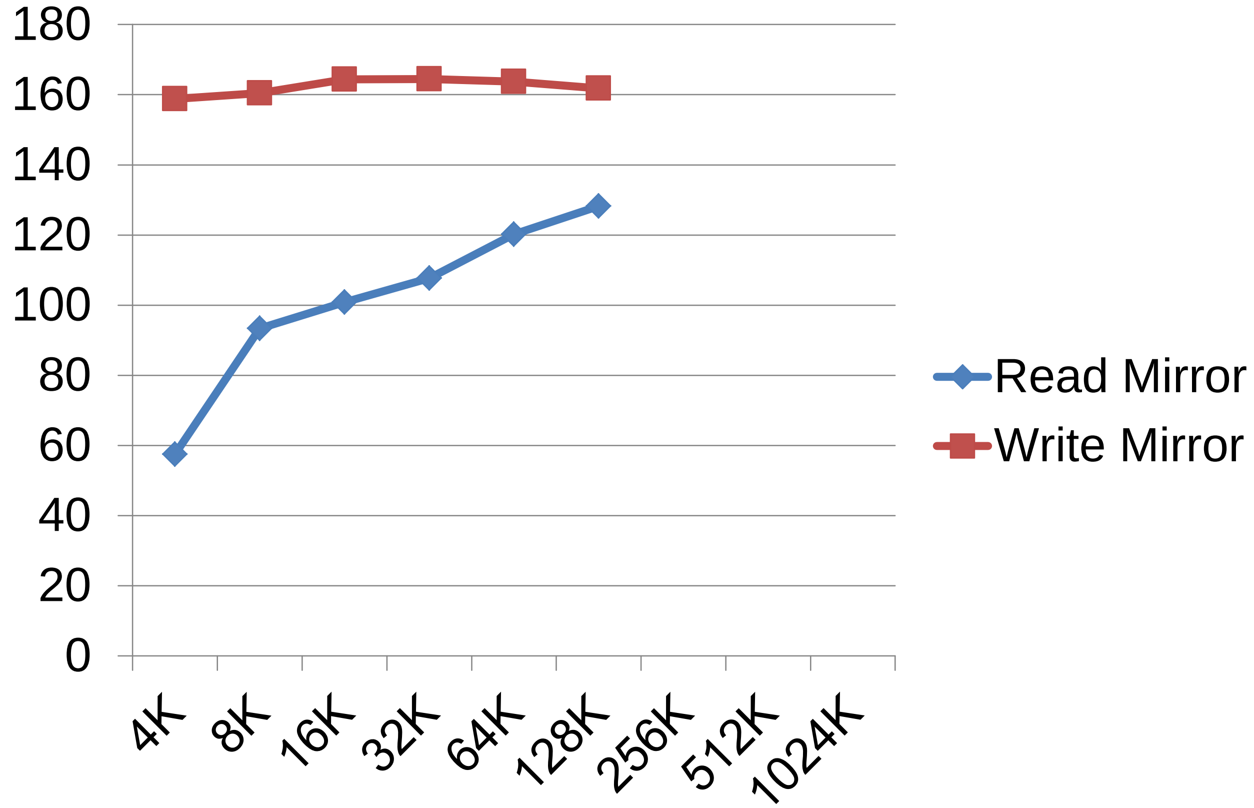
But times have changed and it's now limiting performance

Simulated Lustre OSS Workload

- fio – Flexible I/O Tester
  - 64 Threads
  - 1M IO Size
  - 4G Files
  - Random read/write workload
- All tests run after import to ensure a cold cache
- All default ZFS tunings, except:
  - zfs_prefetch_disabled=1
  - zfs set recordsize={4K..1M} pool/dataset

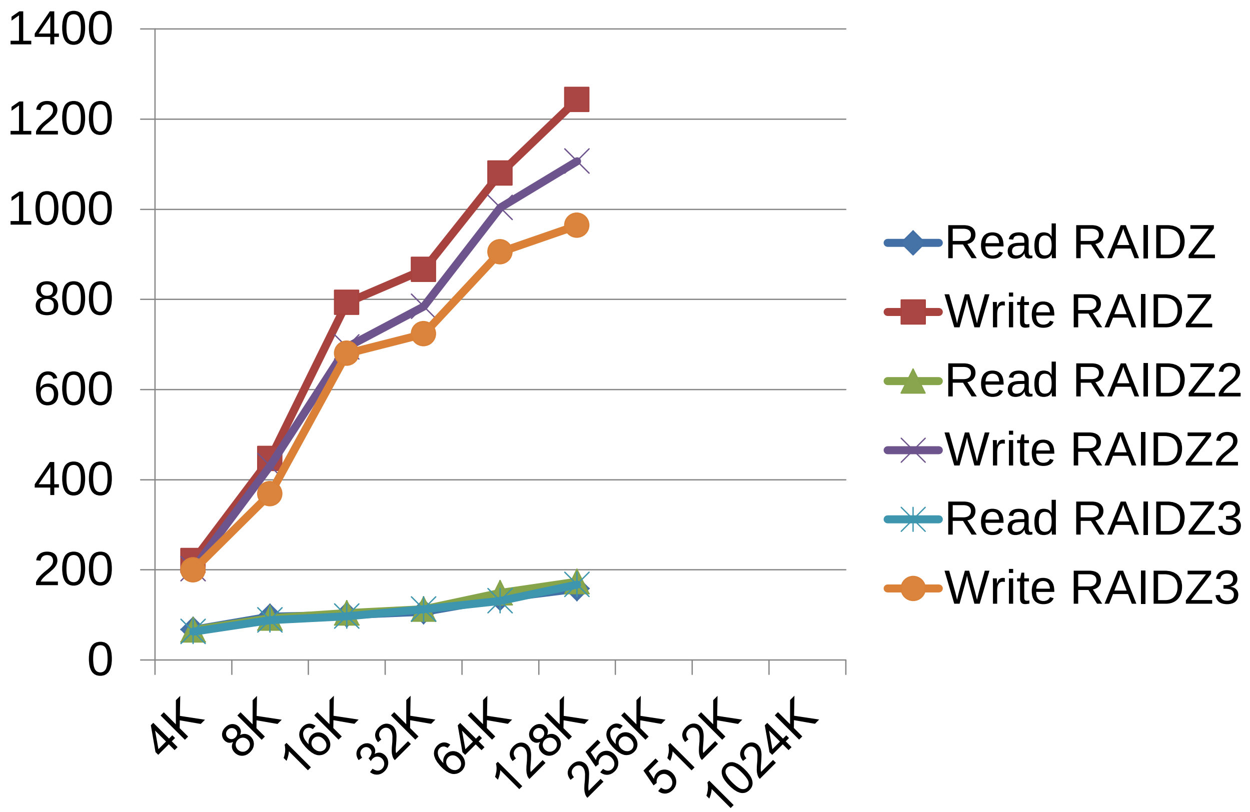
Workload designed to benchmark a Lustre OSS

Random 1M R/W – 2-Disk Mirror MB/s vs ZFS Block Size



Read bandwidth increases with block size

Random 1M R/W – 10-Disk RAIDZ MB/s vs ZFS Block Size



Read and write bandwidth increase with block size

Block Size Tradeoffs

- Bandwidth vs IOPs
  - Larger blocks better bandwidth
  - Smaller blocks better IOPs
- ARC Memory Usage
  - Small and large blocks have the same fixed overhead
  - Larger blocks may result in unwanted cached data
  - Larger block are harder to allocate memory for
- Overwrite Cost
- Checksum Cost

It’s all about tradeoffs

Block Size Tradeoffs

- Disk Fragmentation and Allocation Cost
  - Gang Blocks
- Disk Space Usage
  - Snapshots
- Compression Ratio
  - Larger blocks compress better
- De-Duplication Ratio
  - Smaller blocks de-duplicate better

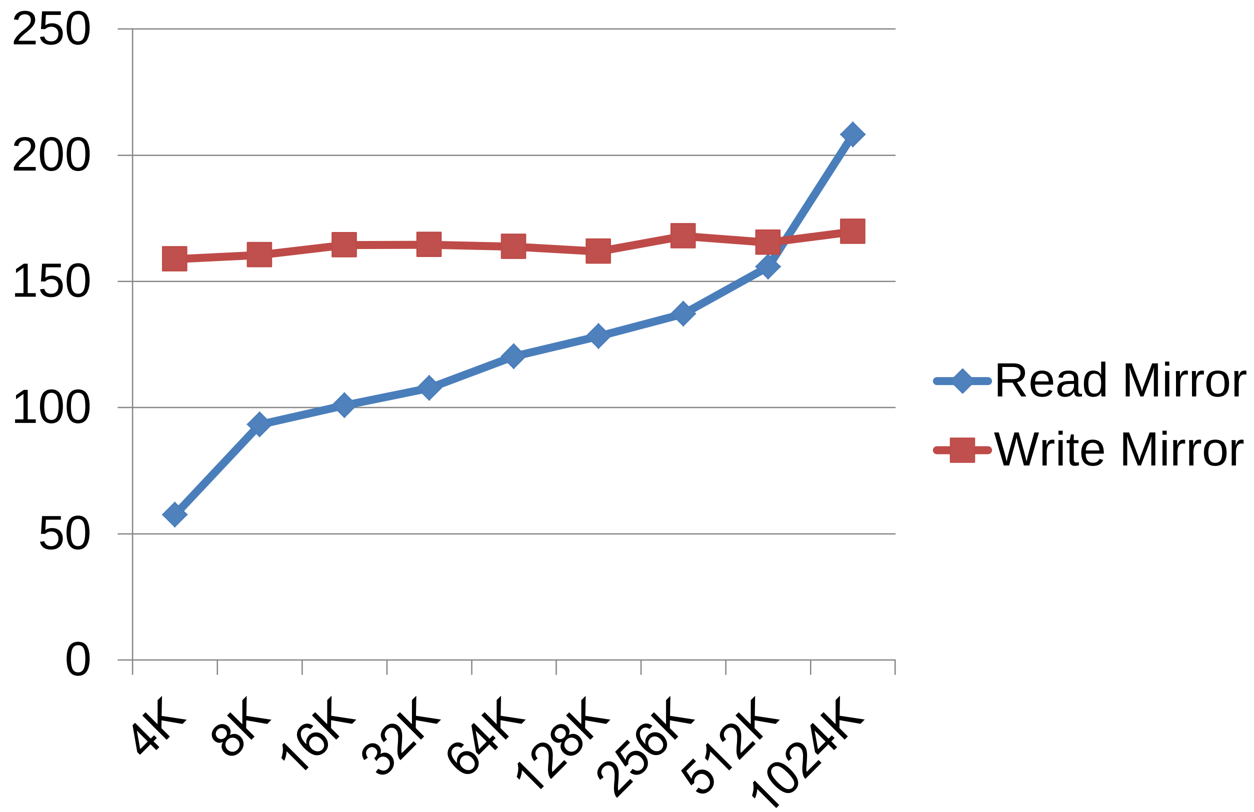
The “right” block size depends on your needs

Let's Increase the Block Size

- The 128K limit
  - Only limited by the original ZFS implementation
  - The on-disk format supports up to 16M blocks
- New "Large Block" Feature Flag
- Easy to implement for prototyping / benchmarking
- Hard to polish for real world production use
  - Cross-platform compatibility
  - 'zfs send/recv' compatibility
  - Requires many subtle changes throughout the code base

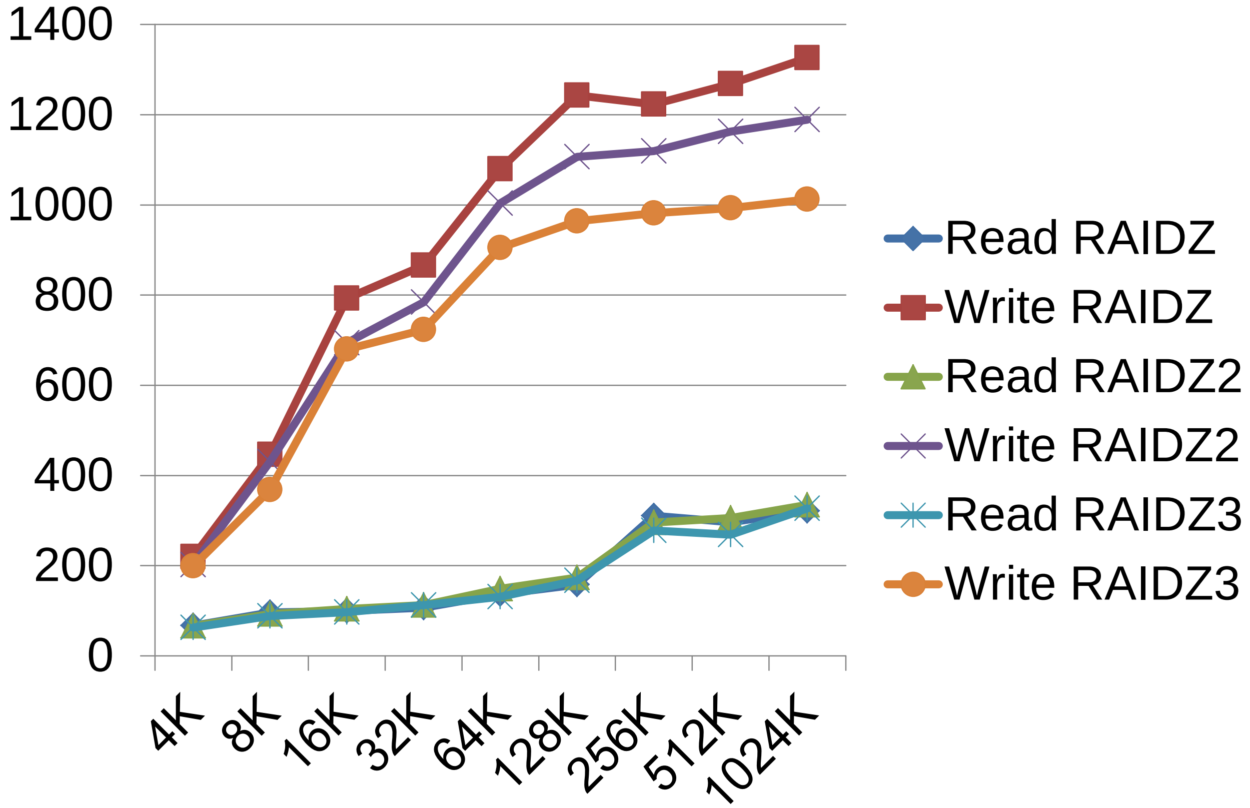
We expect bandwidth to improve…

Random 1M R/W – 2-Disk Mirror MB/s vs ZFS Block Size



63% performance improvement for 1M random reads!

Random 1M R/W – 10-Disk RAIDZ MB/s vs ZFS Block Size



100% performance improvement for 1M random reads!

I/O Performance Summary

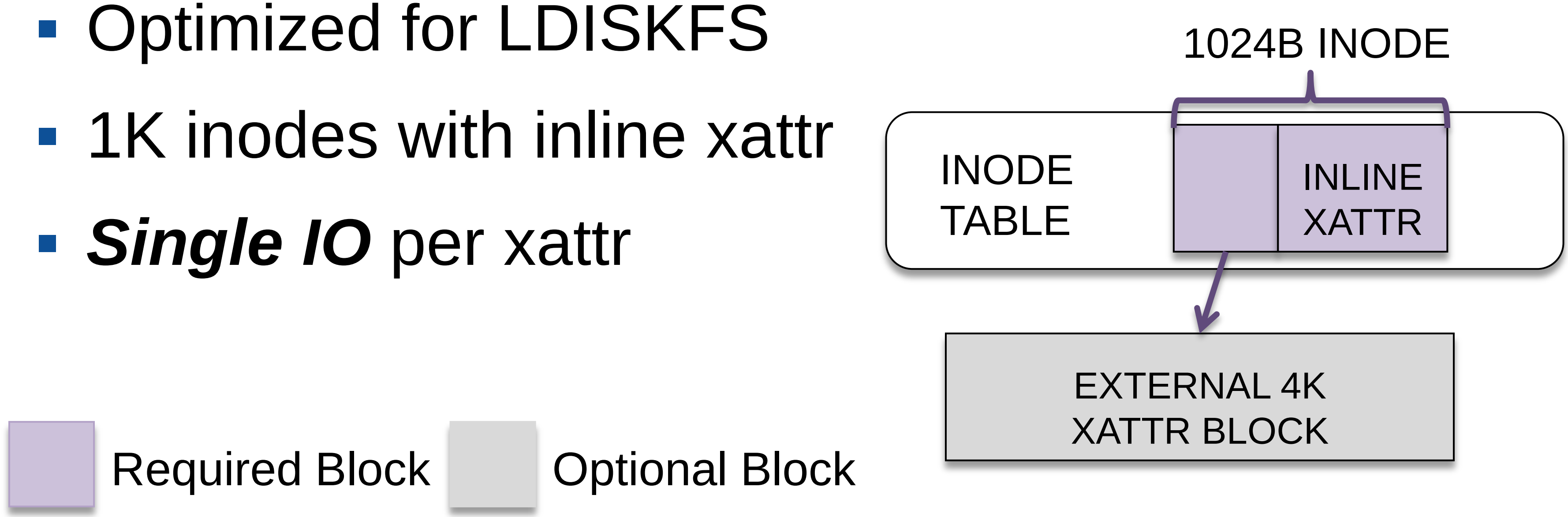
- 1M blocks can improve random read performance by a factor of **2x** for RAIDZ
- Blocks up to 16M are possible
  - Requires ARC improvements (Issue #2129)
  - Early results show 1M blocks may be the sweet spot
  - >1M blocks may help large RAIDZ configurations
- Planned for the next Linux OpenZFS release
- Large blocks already merged upstream

Large blocks can improve performance

Meta Data Performance - Large Dnodes

- MDT stores file striping in an extended attribute

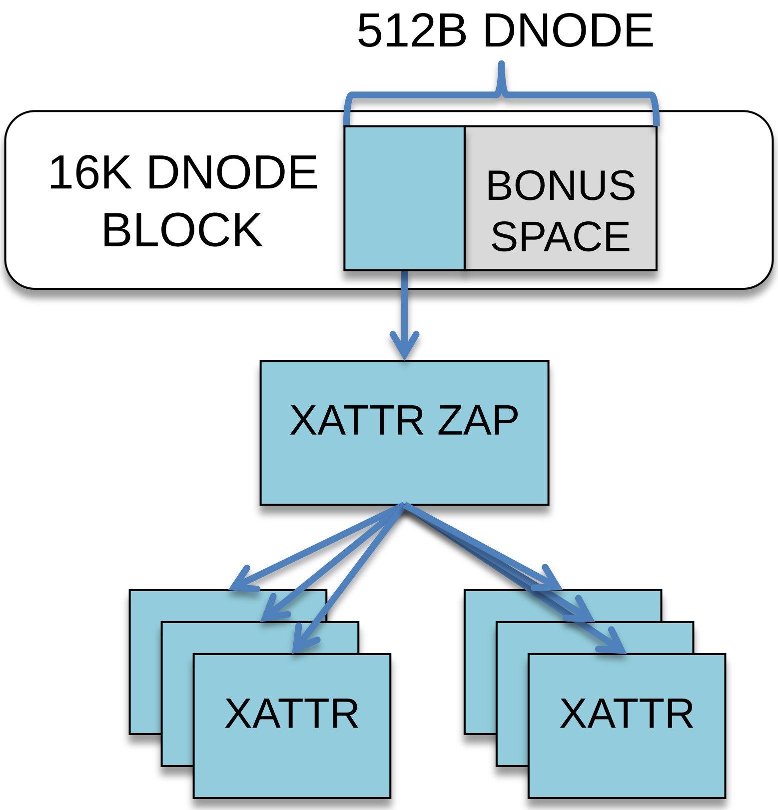
- Optimized for LDISKFS
- 1K inodes with inline xattr
- ***Single IO*** per xattr

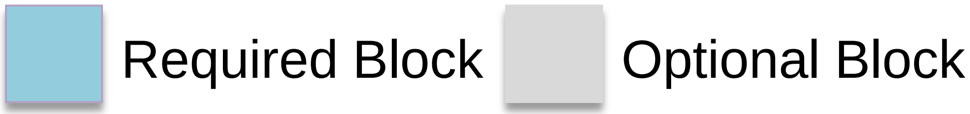


LDISKFS was optimized long long ago for Lustre

File Forks used on illumos / FreeBSD

- Linux "xattrs=on" property
  - Maps xattr to file forks
  - No compatibility issues
  - No limit on xattr size
  - No limit on number of xattrs
- ***Three IOs*** per xattr

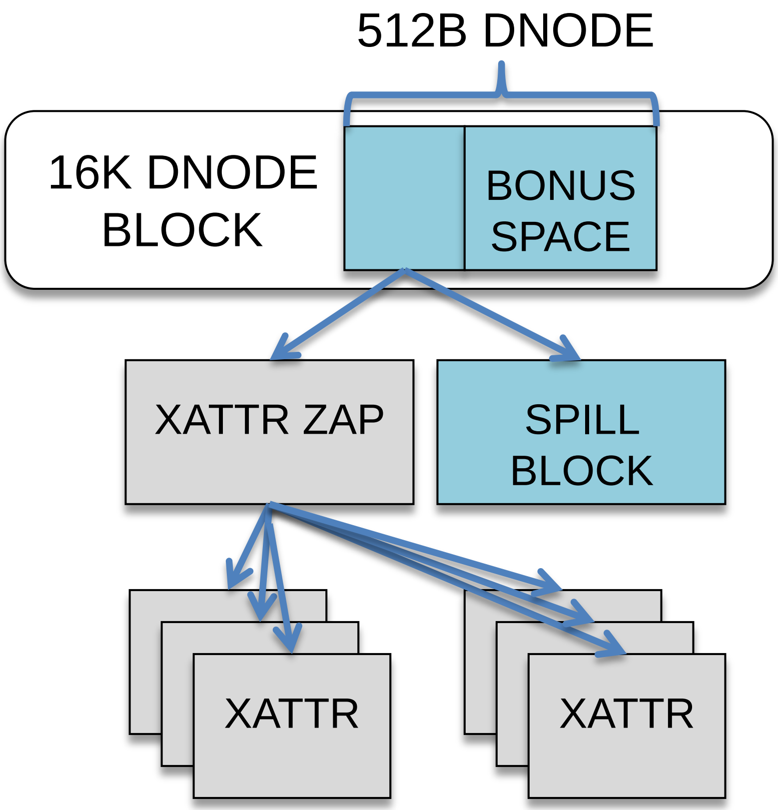




The need for fast xattrs was known early in development

Extended Attributes in SAs

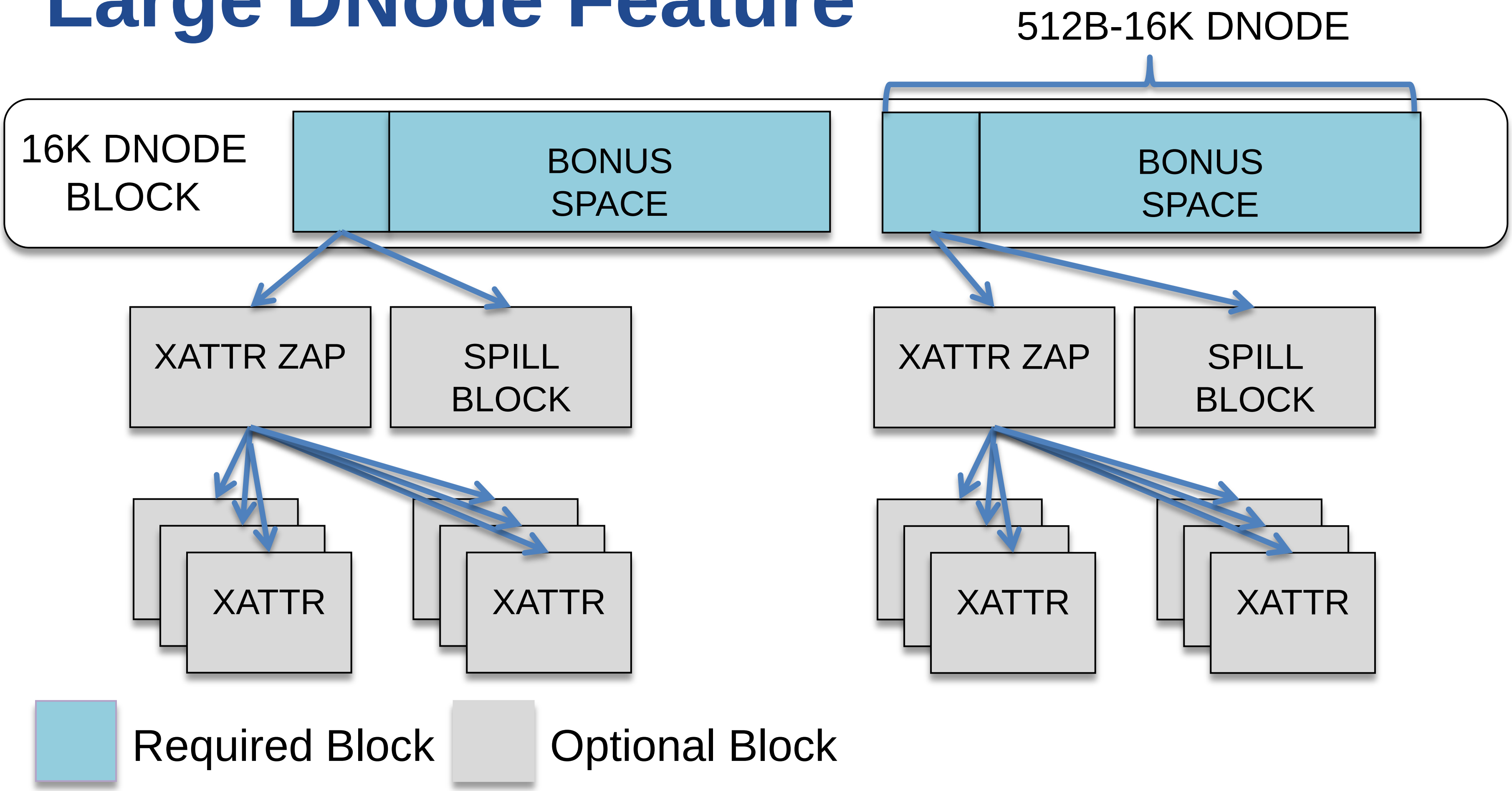
- Goal is to reduce I/Os
- Linux "xattrs=sa" property
  - Maps xattr to SA
  - 1 block of SAs (128K…)
- Much faster
- ***Two IOs*** per xattr



Required Block   Optional Block

The first step was to store xattrs as a system attribute (SA)

Large DNode Feature

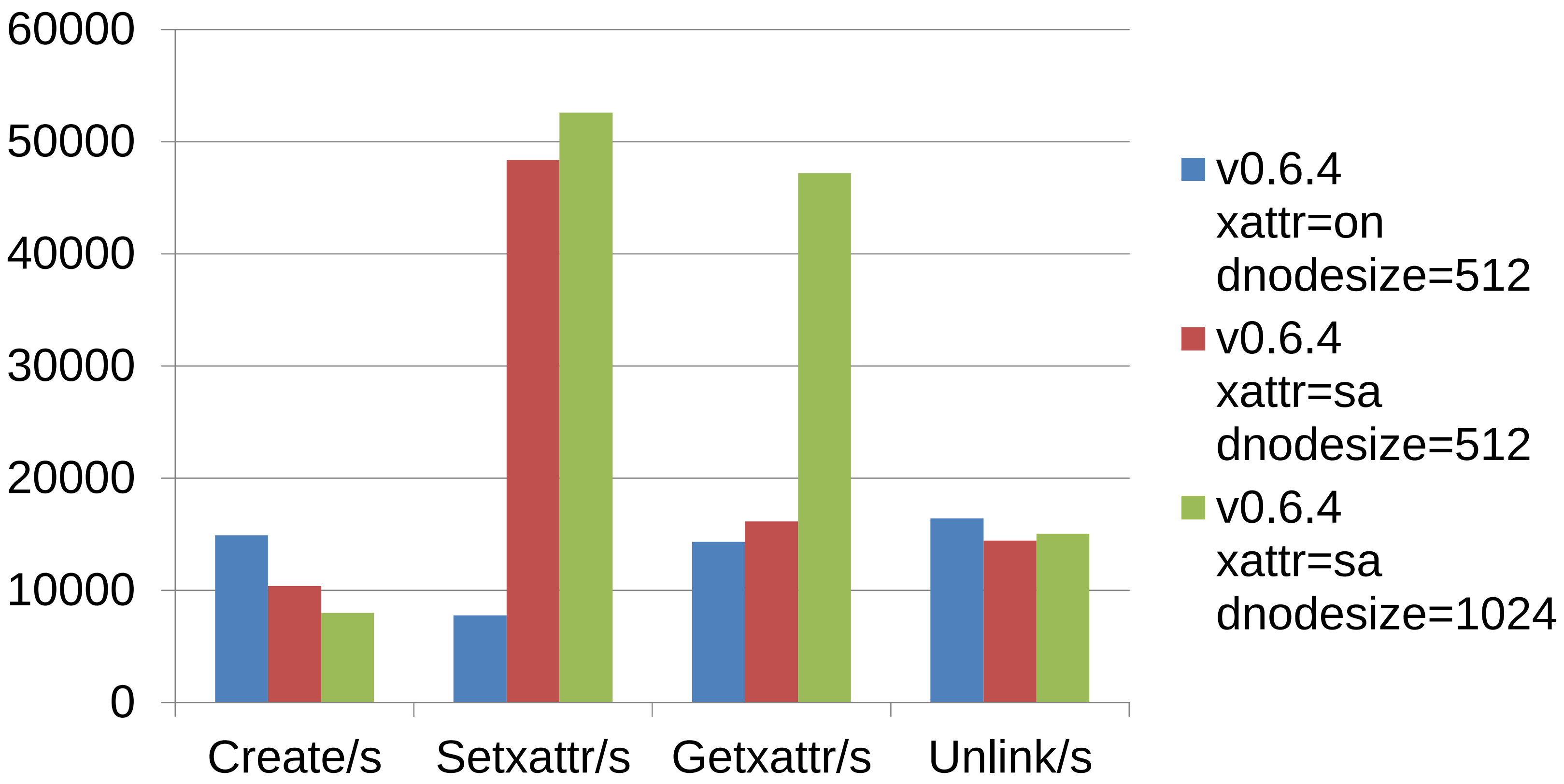


Only a single I/O is needed to read multiple dnodes with xattrs

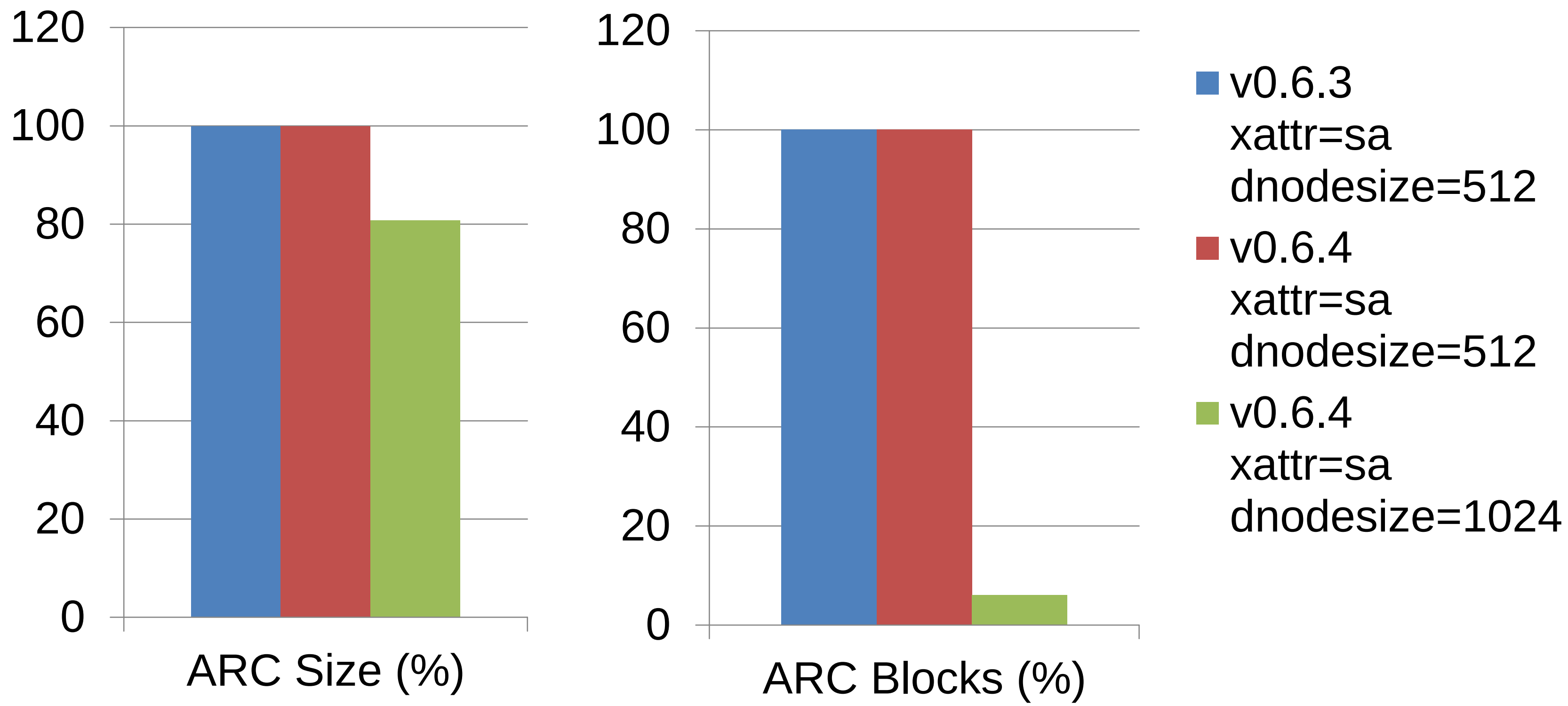
Meta Data Workload

- xattrtest – performance and correctness test
  - https://github.com/behlendorf/xattrtest
  - 4096 files / process, 512 bytes xattr per file
  - 64 processes
- All tests run after import to ensure a cold cache
- 16-disk (HDD) mirror pool
- All default ZFS tunings

Xattrtest Performance

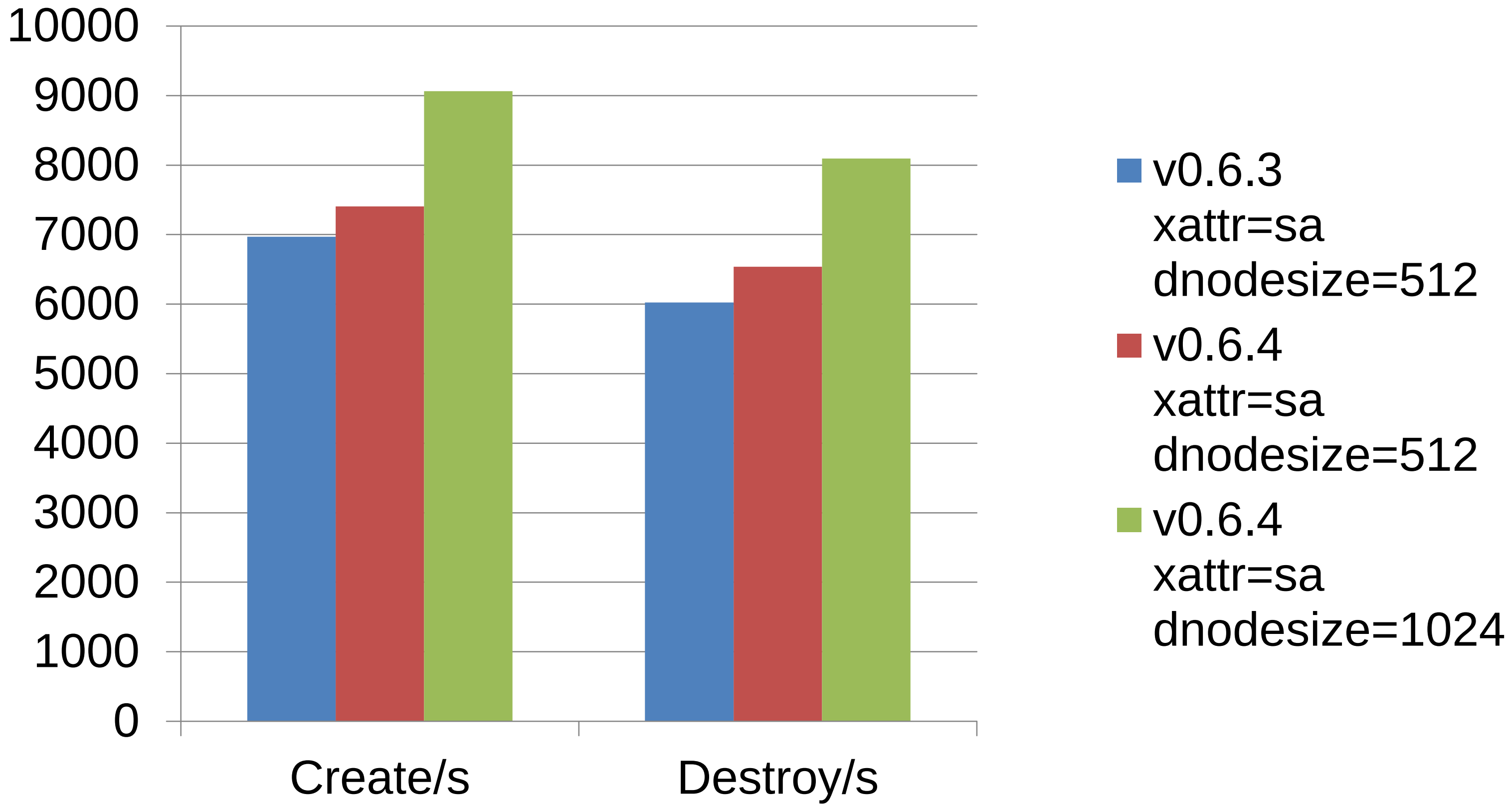


ARC Memory Footprint



16x fewer blocks reduces I/O and saves ARC memory

mds_survey.sh



30% performance improvement for creates and destroys

Large DNode Performance Summary

- Improves performance across the board
- Total I/O required for cold files reduced
- ARC
  - Fewer cached blocks (no spill blocks)
  - Reduced memory usage
  - Smaller MRU/MFU results in faster memory reclaim and reduced lock contention in the ARC
- Reduces pool fragmentation

Increasing the dnode size has many benefits

Large DNode Work in Progress

- Developed by Prakash Surya and Ned Bass
- Conceptually straightforward but challenging
- Undergoing rigorous testing
  - ztest, zfs-stress, xattrtest, mds_survey, ziltest
- 80% done, under active development:
  - ZFS Intent Log (ZIL) support
  - “zfs send/recv” compatibility
- Patches will be submitted for upstream review

Planned Performance Investigations

- Support the ZFS Intent Log (ZIL) in Lustre
- Improve Lustre's use of ZFS level prefetching
- Optimize Lustre specific objects (llogs, etc)
- Explore TinyZAP for Lustre namespace
- More granular ARC locking (issue #3115)
- Page backed ARC buffers (issue #2129)
- Hardware optimized checksums / parity

There are many areas where Lustre MDT performance can be optimized

Questions?

Lawrence Livermore National Laboratory

SPL-0.6.4 / ZFS-0.6.4 Released

- Released April 8th, 2015. Packages available for:
- Six New Feature Flags
  - *Spacemap Histograms
  - Extensible Datasets
  - Bookmarks
  - Enabled TXGs
  - Hole Birth
  - *Embedded Data

SPL-0.6.4 / ZFS-0.6.4 Released

- New Functionality
  - Asynchronous I/O (AIO) support
  - Hole punching via fallocate(2)
  - Two new properties (redundant_metadata, overlay)
  - Various enhancements to command tools
- Performance Improvements
  - ‘zfs send/recv’
  - Spacemap histograms
  - Faster memory reclaim
- Over 200 Bug Fixes

New KMOD Repo for RHEL/CentOS

- DKMS and KMOD packages for EPEL 6
  - Versions: spl-0.6.4, zfs-0.6.4, lustre-2.5.3
- Enable the repository
  - vim /etc/yum.repos.d/zfs.repo
  - Disable the default zfs (DKMS) repository
  - Enable the zfs-kmod repository
- Install ZFS, Lustre server, or Lustre client
  - yum install zfs
  - yum install lustre-osd-zfs
  - yum install lustre

See zfsonlinux.org/epel for complete instructions