ZFS Improvements for Lustre* - 0.7 & Beyond

Andreas Dilger, Intel High Performance Data Division

LUG'18

Legal Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life-saving, life-sustaining, critical control or safety systems, or in nuclear facility applications.

Intel products may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

This document may contain information on products in the design phase of development. The information herein is subject to change without notice. Do not finalize a design with this information. Intel may make changes to dates, specifications, product descriptions, and plans referenced in this document at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Intel Corporation or its subsidiaries in the United States and other countries may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Performance estimates or simulated results based on internal Intel analysis or architecture simulation or modeling are provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804.

Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at http://www.intel.com/go/turbo.

Intel processors of the same SKU may vary in frequency or power as a result of natural variability in the production process.

Intel, the Intel logo, 3D-Xpoint, Optane, Xeon Phi, and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands may be claimed as the property of others.

ZFS Enhancements of Interest to Lustre

OpenZFS

Changes included in ZFS 0.7.x

- Multi-modifier protection (MMP) for improved HA safety (LLNL)
- Dynamic dnode size (large dnodes) (LLNL)
- Native dnode quota accounting (Intel)
- Multi-threaded dnode allocation, locking, APIs (Delphix, Intel, LLNL)
- CPU-optimized checksums, RAID parity (Uni Hamburg, Intel)
- QAT hardware-assisted checksums and compression (Intel)
- Improved kernel memory allocation for IO buffers (ABD) (others, Intel)
- Improved JBOD fault detection, handling, enclosure LEDs (LLNL)
- Fault Management Architecture (FMA)/ZED integration (others, Intel)

Landing for ZFS 0.8.x

- Sequential scrub/resilver (Datto, Nexenta)
- On-disk data/metadata encryption (Datto)
- QAT-accelerated encryption (Datto, Intel)
- ZFS Channel Programs (Delphix)
- Project quota accounting (Intel)
- MMP fixes and improvements (LLNL)
- Device removal/remapping (Delphix)

- Metadata Allocation Class (Intel, Delphix)
- Declustered parity RAID (dRAID) (Intel)

Lustre with ZFS-on-Linux - Release Notes

Lustre 2.10.4/2.11/2.12 currently updated to build and test with ZFS 0.7.8

- Will continue updating 2.10/2.12 to track latest releases
- Sites can build preferred ZoL version (skip 0.7.7), modulo build incompatibilities

ZoL 0.8.0 will hopefully be released much quicker than 0.7.0 was

- ZoL 0.7.0 released 53 months after 0.6.0, and 21 months after 0.6.5
- Target ZoL 0.8.0 release currently 2018-09 (15 months after 0.7.0)
- Lustre 2.12 will be *able* to build with 0.8.x, even if 0.8.0 unreleased

Lustre will start to build using upstream ZoL RPMs for supported distros

- Allows use of pre-built Lustre RPMs to work on ZFS backends w/o DKMS

Lots of other improvements underway, too many to cover all here

Multi-Modifier Protection

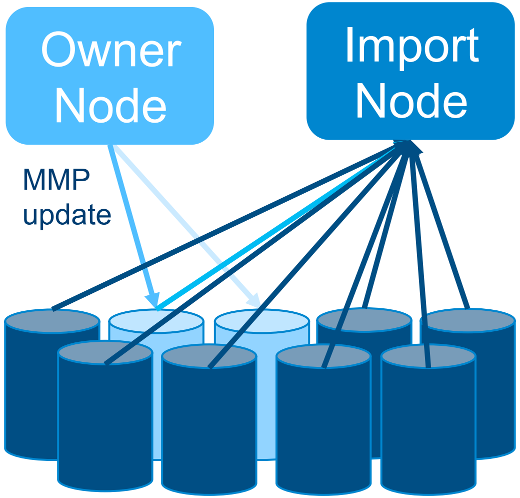
(PR6279 LLNL, 0.7)

MMP prevents multiple nodes importing the same pool

- Significant risk if HA software or STONITH fails
- ZFS not robust with this kind of problem
  - Wrong blocks with valid checksums are BAD

Owner node writes to **one** random VDEV label per update interval

- Update only timestamp in reserved MMP überblock(s)
- No extra MMP überblock write if normal TXG recently written
- *Each* VDEV should get one MMP write per check interval



Import node checks *all* VDEV überblocks for *any* modifications

- Won't import pool if it detects *any* modified blocks after delay

Enabled by default in Lustre 2.10.1+, or with "`zpool property multihost=on`" for existing pools

Dynamic (large) dnode Size (PR3542 LLNL, 0.7)

ZFS 0.6.x and earlier supported only 512-byte dnodes

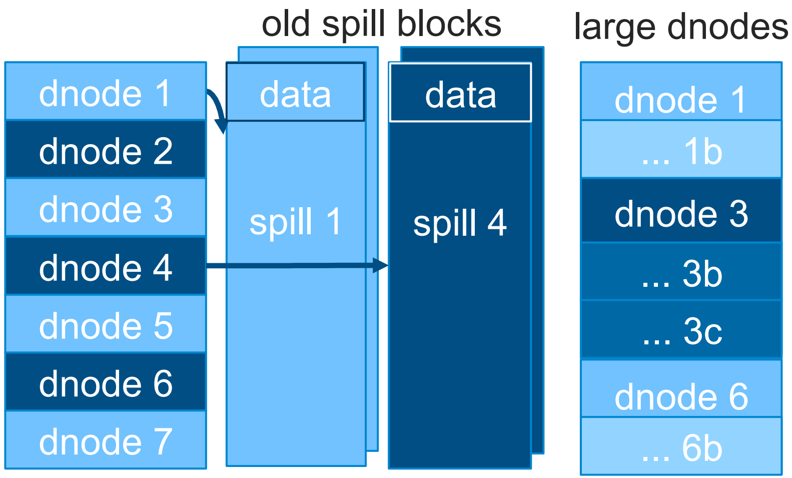
Lustre xattrs (LMA, LinkEA, LOV + PFL, ...) didn't all fit within 512-byte dnode

- Each dnode allocates two extra 4K blocks (*spill block* + mirror copy) for xattrs
- Over 9 GB mirrored writes to create 1M files

Large dnodes improve seek and I/O efficiency

- Variable dnode size from 0.5KB-16K
- Only 2 GB mirrored writes to create 1M files
- Reduce seeks by 50% (no seek for spill block)

Enable with "`dnodesize=auto`" in **0.7.1+ ONLY**



User/Group dnode Quota Accounting (PR3500 Intel, 0.7)
Project Quota Accounting (PR6290 Intel, 0.8)

ZFS didn't support native dnode accounting, only block accounting

Lustre 2.9 and earlier implemented its own code for file quota

- Didn't scale well - two extra files updated on every file creation

Add native dnode (2.10+0.7) and project quota (2.11+0.8) using block code

- Updates a single accounting file for all quota types

Project ID for quota accounting independent of access control (UID/GID)

- Project ID inherited from parent dir, compatible with ext4/XFS interfaces

Fault Management Architecture ZED Integration (PR4673 Intel 0.7)

Fault Management Architecture (FMA) correlates events

- Repeated checksum errors, totally failed drives, ...

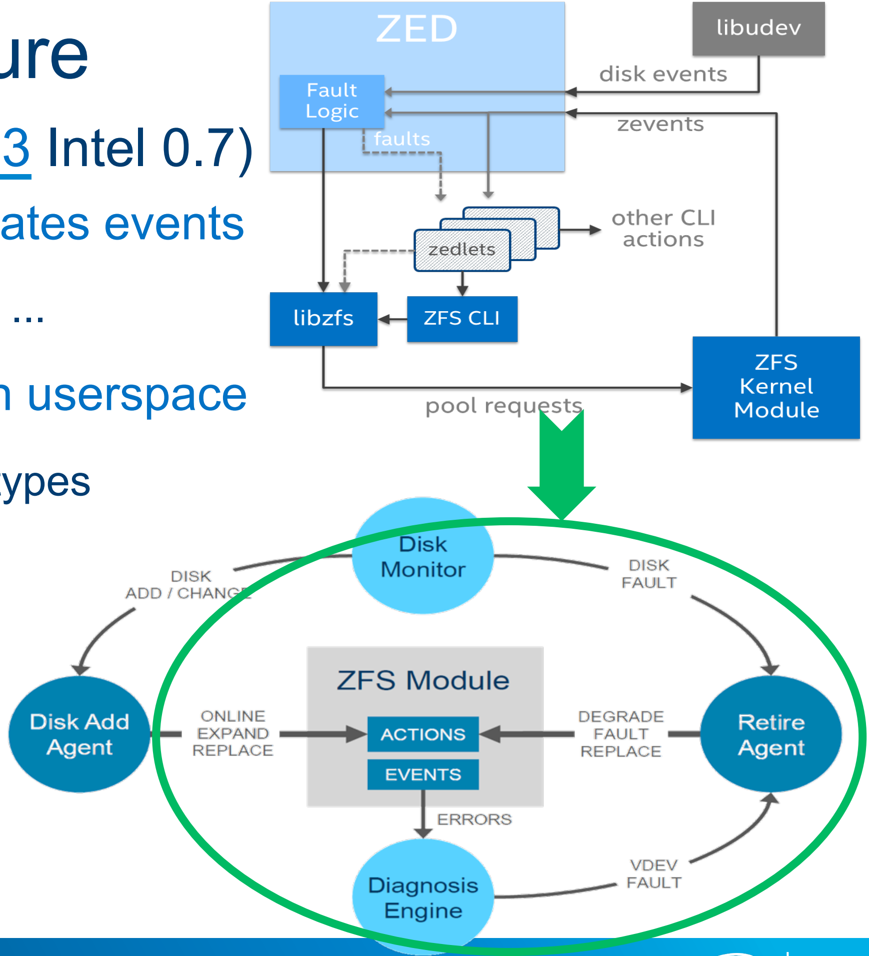
ZFS Event Daemon (ZED) handles actions in userspace

- zedlets (scripts) run based on registered event types
- Alert admins of failures, mark disks offline
- Auto-replace bad drive with hot spare

SCSI Enclosure Services handles JBODs

- Illuminate enclosure LED blinkenlights

ZED/FMA integration with IML in 4.1 release



File Creation Performance (Delphix, Intel, 0.7)

Multi-threaded transaction group (TXG) syncing (PR5752)

- Flush dirty dnode blocks to multiple devices in parallel

Improved object allocation (PR6564, PR6611, PR6117)

- Multi-threaded dnode allocation to avoid lock contention

Batched quota updates (PR4642)

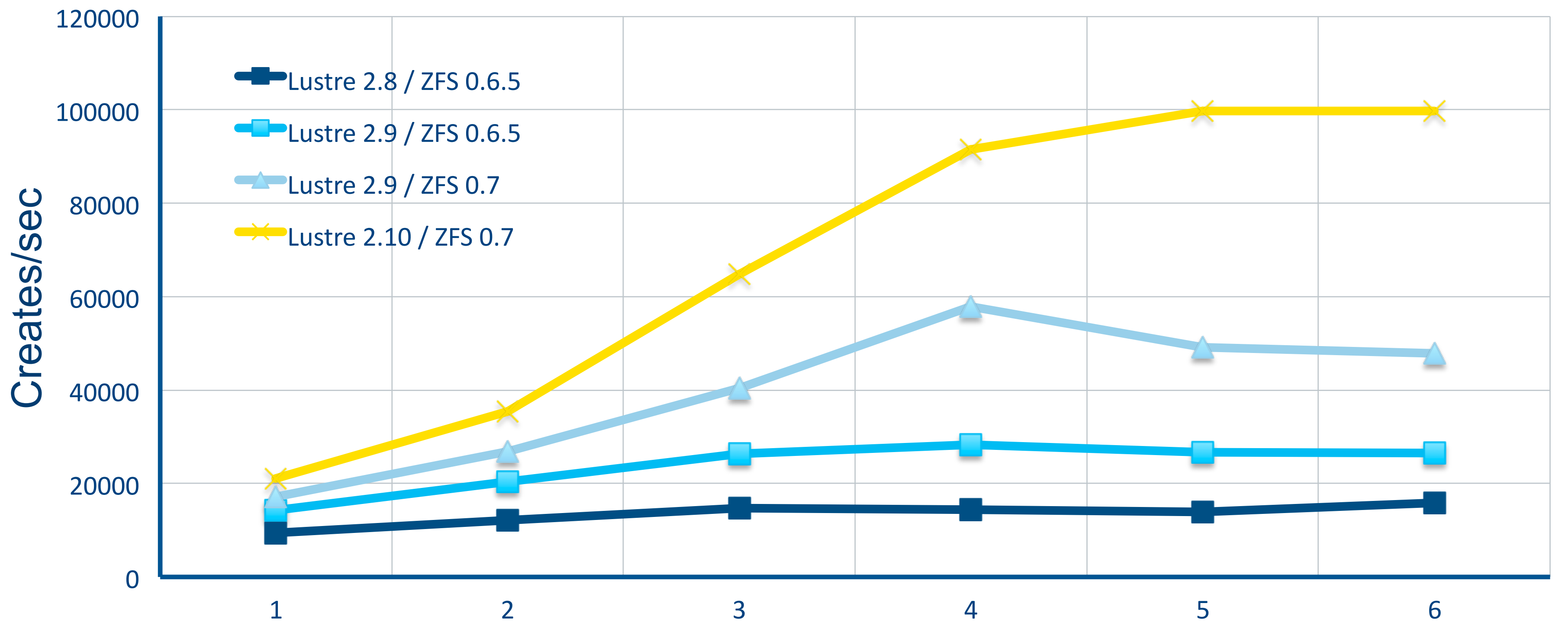
- Modify quota updates once per TXG (+/-n), *not* once per block (+/-1)

Avoid dnode lookups *per function call* to avoid needless overhead (PR5534, PR5894)

- Add `*_by_dnode()` APIs after initial dnode lookup is done

Reduce unnecessary allocations during create (PR6048)

Lustre File Creation: step-by-step (`mds-survey`)



2x Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz – 20 cores, 64GB RAM, 3 x 500GB HDD

QAT Hardware Compression (PR5846 Intel, 0.7)
QAT Checksums/Encryption (PR7295 Datto, 0.8)

Compression *improves* performance

Intel Quick Assist Technology (QAT)

- PCI card/chipset accelerators

QAT GZip compression in ZFS 0.7.0

- Not built unless QAT libraries installed

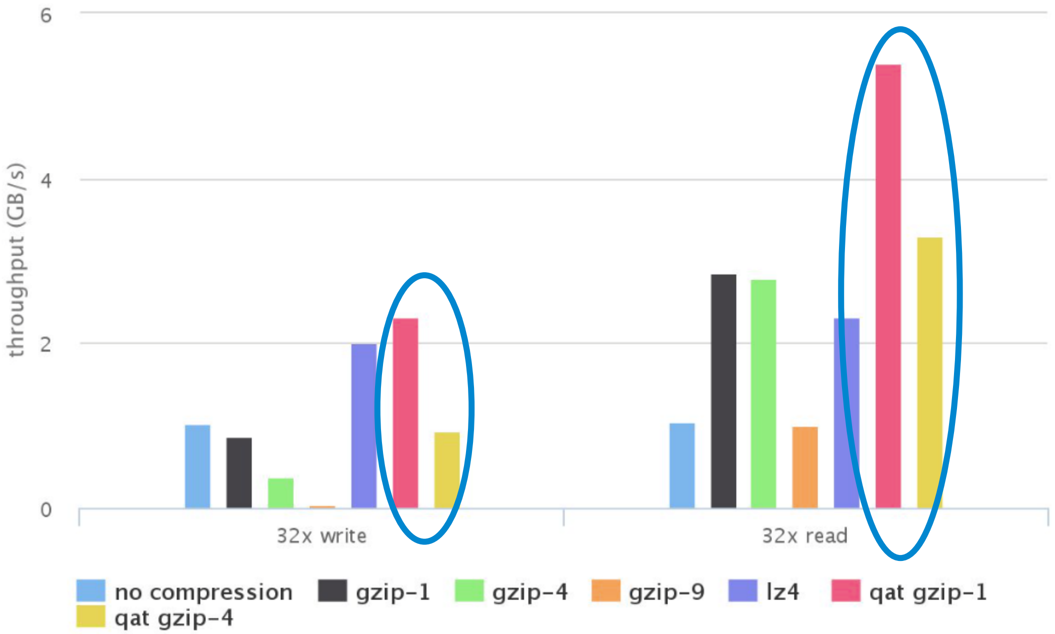
Benchmark shows ZFS local I/O perf

- Data from Beijing Genomics Institute
- 2 Intel® Xeon E5 2620v3 + QAT 8950

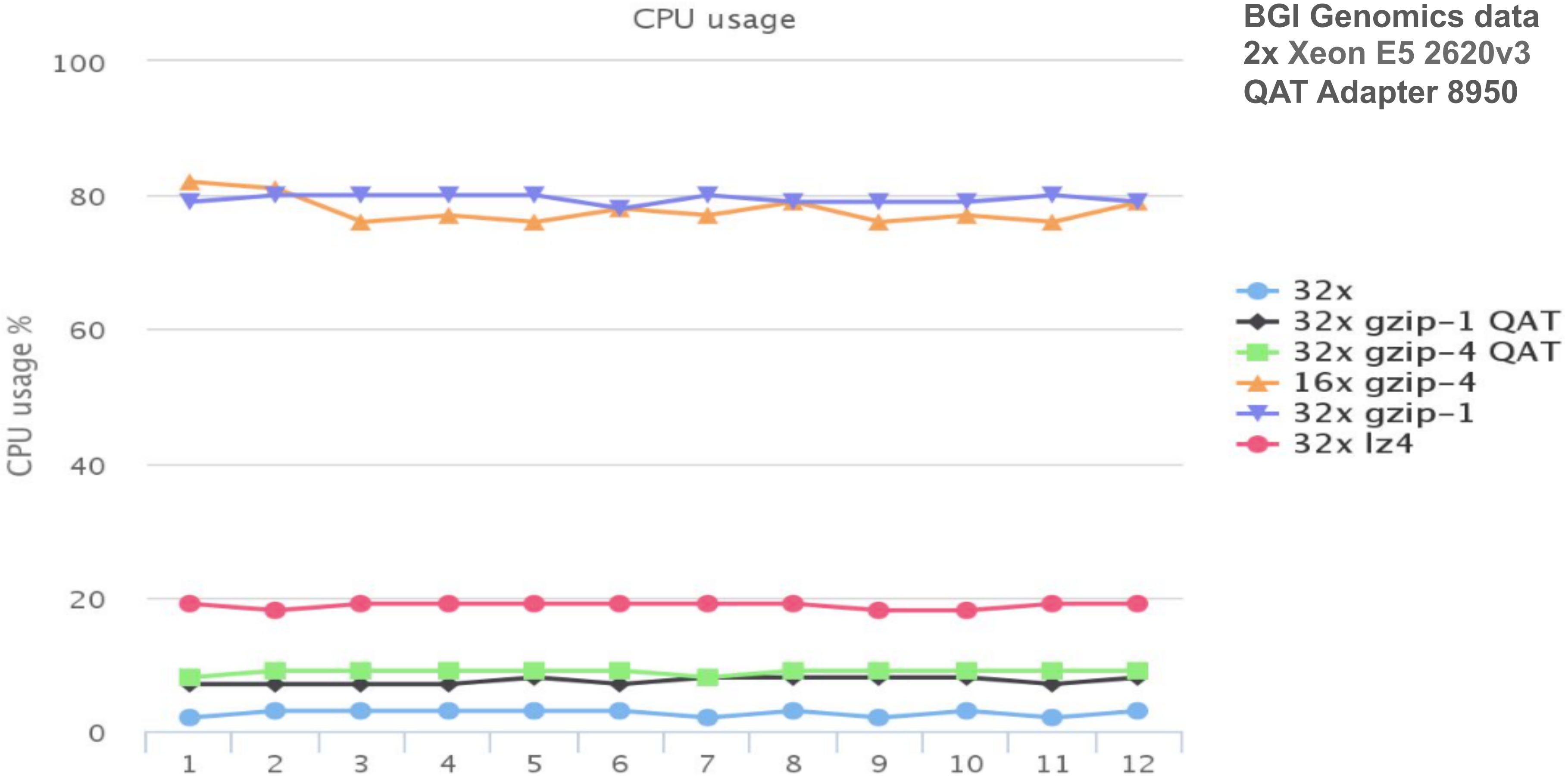
Checksum/Encryption in ZFS 0.8.0

- Accelerate SHA256, AES-GCM

ZFS throughput (BGI genomics data)



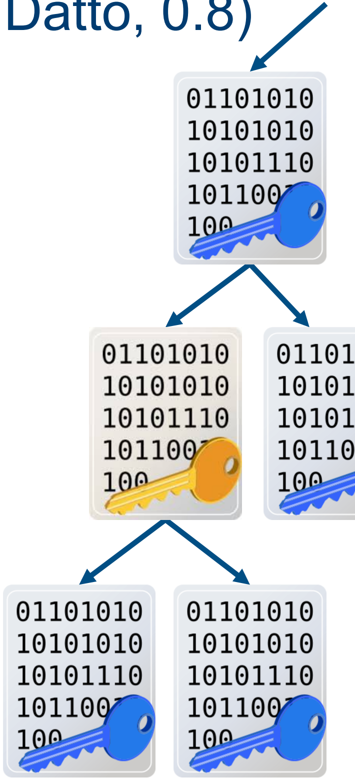
Compressed ZFS Read



BGI Genomics data
2x Xeon E5 2620v3
QAT Adapter 8950

ZFS On-disk Encryption

(PR5769 Datto, 0.8)



Tree-based block-level encryption with per-dataset encryption keys

- Dataset encryption key protected by user-supplied key
- Each dataset has a Merkel tree of MACs that protect lower layers
  - Each block has own encryption key - limit loss in case of corruption
  - Blocks in ZIL and L2ARC are also encrypted, only compressed in ARC
- Allows user key to be updated without re-encrypting entire pool
- Can `zfs send/recv` (backup) encrypted datasets without keys

Lustre still needs a mechanism for managing per-target keys

- MGS or IML distributes keys to MDTs/OSTs at startup time?

Developer presentation https://www.youtube.com/watch?v=frnLiXclAMo

Device (VDEV) Removal

(PR6900 Delphix, 0.8)

Has not been possible previously

- Snapshots and checksums prevent block reallocation

Recover from accidental device addition

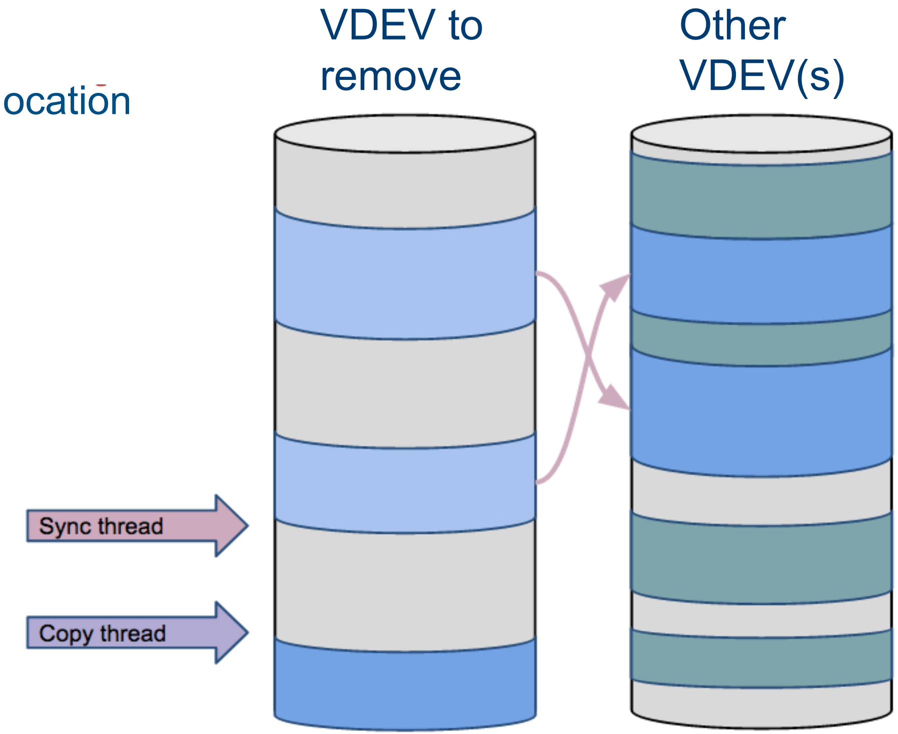
- Single device added without parity

Shrink pool that no longer needs VDEV

- Traverse VDEV in block order
- Map allocated extent into free space
- Copy used extents into virtual VDEV
- Remove VDEV when all extents copied

Mapped extents removed when empty

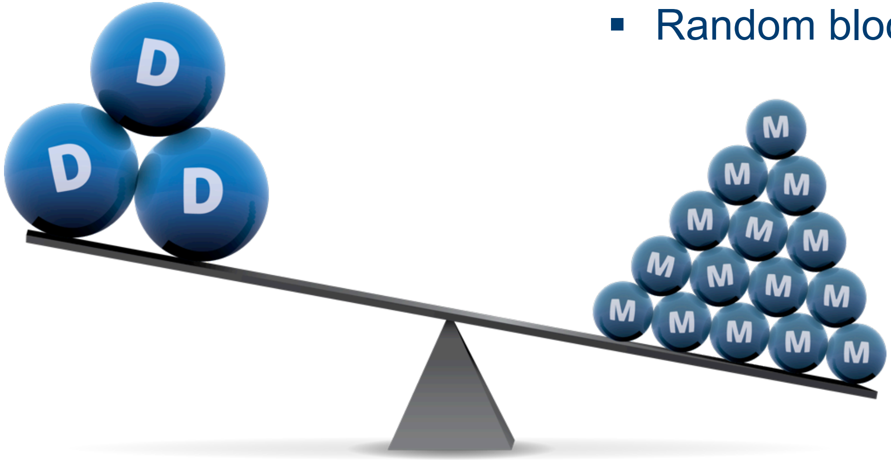
- Normal data aging/removal or copy-on-write



Metadata Isolation from Data

File Data

- Large blocks (up to 16MB)
- Free space fragmentation
- High throughput
- Large capacity
- Sequential
- RAID-Z2
- Typically HDD



Metadata

- Transient lifetime (especially COW)
- Need fast performance for scrub
- Random block access
- Small (<32KB)
- High IOPS
- Low latency
- Mirror
- SSD preferred

Metadata Allocation Classes (PR5182 Intel, Delphix, 0.8)

Virtual Devices divided into *Metaslabs*

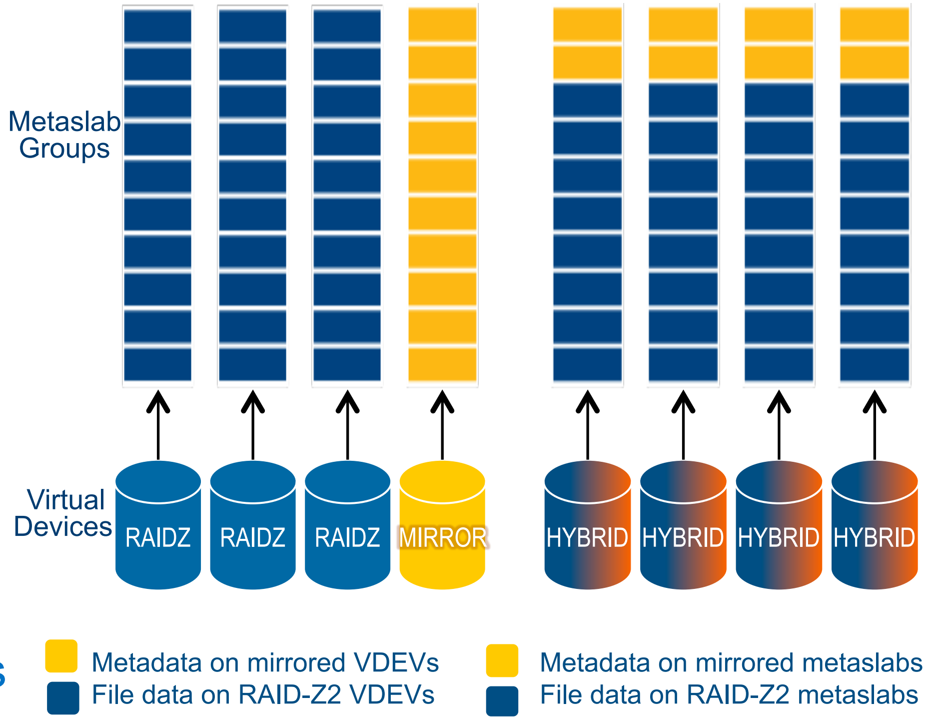
- Metaslabs belong to an *allocation class*
- Metaslab is `normal`, `special`, or `dedup`

Use dedicated VDEVs or hybrid slabs

- Separate metadata, small file, DDT class
- Can use different VDEVs for each class
- E.g. NVMe for metadata, HDD for data

Avoids free space fragmentation

No IO contention with separate VDEVs



Declustered Parity RAID (dRAID) with Distributed Hot Space

(PR7078 Intel, 0.8)

RAID Data+Parity width separated from drive count

- RAID stripes use pseudo-random ordering repeated over drives

Hot spare drive(s) mixed with D+P drives

- Add bandwidth/IOPS of spare devices to normal operation
- Part of each drive is *hot space*, new drives too big anyway

Resilver across all drives in zpool

- Improve resilver speed by factor of number of VDEVs
- Post-resilver scrub is still needed, but no double failure risk

Sequential rebuild scans metaslabs for free space

- Fixed alignment of RAID chunks allows parity reconstruction
- Sequential drive access speeds rebuild, unlike RAID-Z
- Skipping free space speeds rebuild, unlike traditional RAID

	4+1 Redundancy Group					4+1 Redundancy Group					Spares	
Base	**9**	**8**	**10**	**3**	**6**	**5**	**4**	**7**	**0**	**1**	**2**	**11**
Permutation Development	10	9	11	4	7	6	5	8	1	2	3	0
	11	10	0	5	8	7	6	9	2	3	4	1
	0	11	1	6	9	8	7	10	3	4	5	2
	1	0	2	7	10	9	8	11	4	5	6	3
	2	1	3	8	11	10	9	0	5	6	7	4
	3	2	4	9	0	11	10	1	6	7	8	5
	4	3	5	10	1	0	11	2	7	8	9	6
	5	4	6	11	2	1	0	3	8	9	10	7
	6	5	7	0	3	2	1	4	9	10	11	8
	7	6	8	1	4	3	2	5	10	11	0	9
	8	7	9	2	5	4	3	6	11	0	1	10

Numbers are VDEV drive index

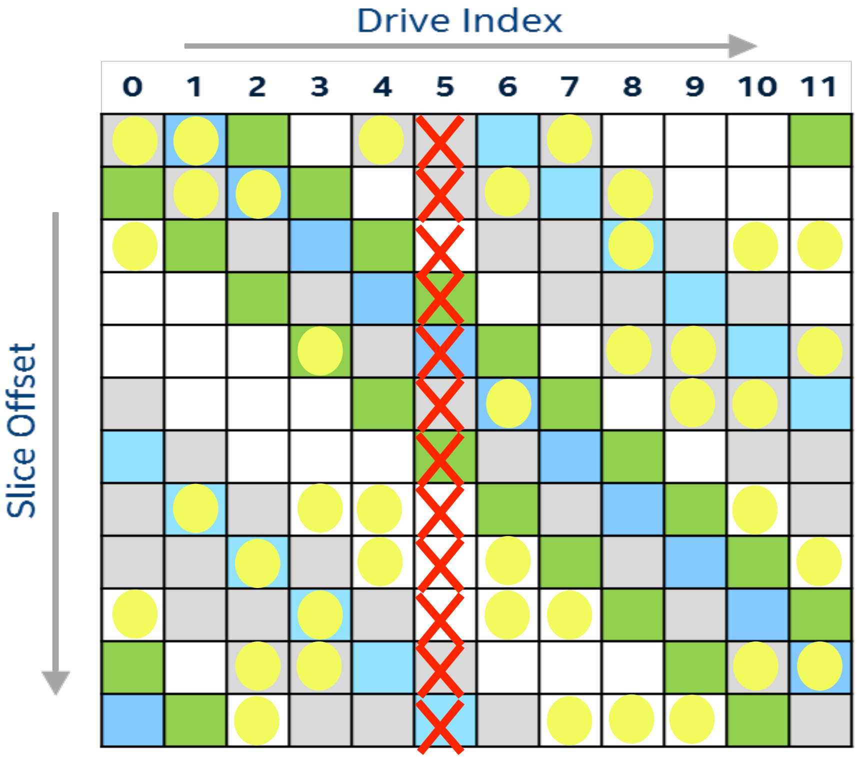
Permutation Layout

Drive Layout

Permutation Development

	4+1 Redundancy Group				4+1 Redundancy Group					Spares	
Base	**9**	**8**	**10**	**3**	**6**	**~~5~~**	**4**	**7**	**0**	**1**	**2**	**11**
	10	9	11	4	7	6	~~5~~	8	1	2	3	0
	11	10	0	~~5~~	8	7	6	9	2	3	4	1
	0	11	1	6	9	8	7	10	3	4	~~5~~	2
	1	0	2	7	10	9	8	11	4	~~5~~	6	3
	2	1	3	8	11	10	9	0	~~5~~	6	7	4
	3	2	4	9	0	11	10	1	6	7	8	~~5~~
	4	3	~~5~~	10	1	0	11	2	7	8	9	6
	~~5~~	4	6	11	2	1	0	3	8	9	10	7
	6	~~5~~	7	0	3	2	1	4	9	10	11	8
	7	6	8	1	4	3	2	~~5~~	10	11	0	9
	8	7	9	2	~~5~~	4	3	6	11	0	1	10

X Drive #5 failure



Active drives for stripe rebuild

Other ZFS Developments of Interest

ZFS Object Index repair (OI Scrub) (LU-7585, Intel Lustre 2.11)

- MDT Object Index + FID rebuild after corruption/bug or tar/rsync backup/restore
- MDT/OST migration with tar/rsync of *ldiskfs* backup and restore to ZFS

Sequential scrub/resilver (PR6256, Nexenta, 0.8)

- Reduce HDD verification/rebuild time by ordered tree scan to minimize seeks

ZFS Channel Programs (ZCP) (PR6558, Delphix, 0.8)

- Complex administrative actions can run atomically in pool transaction
- Avoids need to modify kernel code for (some) new functionality
- Lua script interface runs in kernel interpreter

intel®

Hardware Used in mds-survey Benchmarks

- 2 x Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz – 20 cores
- 64GB RAM
- 3 x 500GB HDD