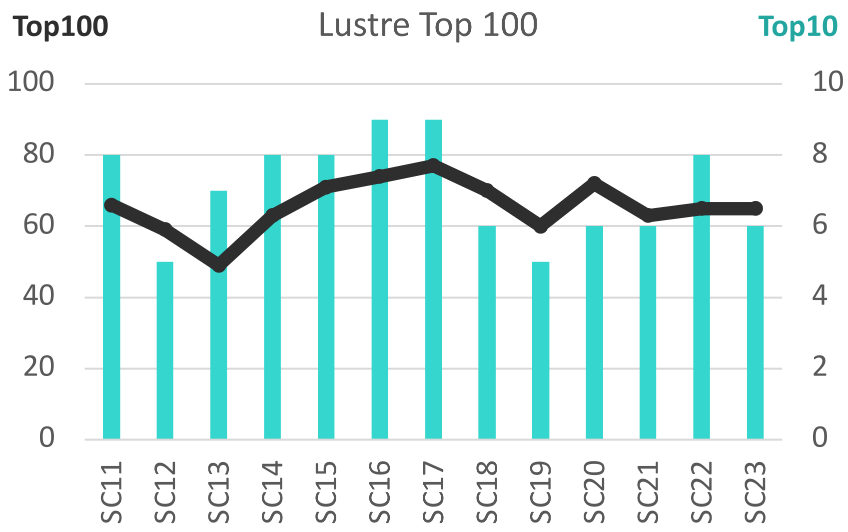
LAD24
& LUSTRE ADMIN DEV WORKSHOP
Paris 23.24.25/9/2024

Whamcloud

Lustre 2.17 and Beyond

Andreas Dilger
Lustre Principal Architect

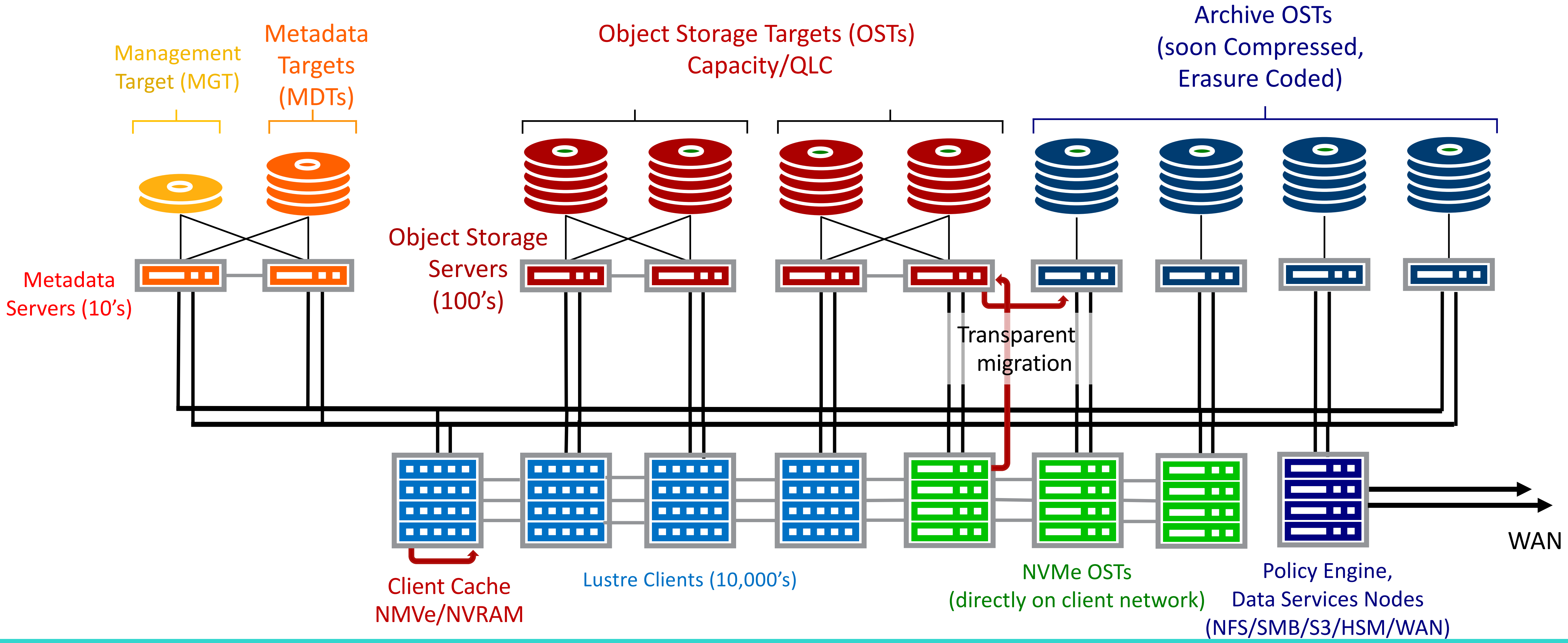
Lustre Committed to Exascale and the Future



- **The preferred choice for the world's largest systems**
  - Majority top 10/100 HPC systems use Lustre
  - World's largest AI/ML systems (Eos, Selene, Cam1, Scaleway)
  ... or 2RU server for workgroup with 16 GPU nodes
- **Scalability of servers and clients almost without limits**
  - 100M+ IOPS, 10 M+ metadata op/sec, 100B+ files
  - Capacity for any need – 10s TB/s read/write, 100s of PB today and 1 EB+ in the near future
  - Fully support large clients - 100s of cores, TBs of RAM, multi-100Gbps NICs, GPU RDMA
- **Continued improvements for large system deployments**
  - Steady feature development to meet evolving system and application needs
  - Virtualization of filesystem for multi-tenant and data privacy
- **Improving ease-of-use and reliability**
  - Demand for fast storage is everywhere

Tiered Storage and File Level Redundancy

Data locality, with *direct access* from clients to all storage tiers as needed



Planned Feature Release Highlights

- **2.17** has major features ready for feature landing window to open
  - **Hybrid IO Optimizations** – Hybrid BIO/DIO and server writeback cache (WC, Oracle)
  - **Dynamic Nodemaps** – ephemeral/hierarchical configuration for subdirectory trees (WC)
  - **Client-side Data Compression** – reduce network and storage usage/cost (WC, UHamburg)
- **2.18** already has some features under development
  - **Metadata Writeback Cache** (WBC2) – single-client metadata speedup (WC)
  - **File Level Redundancy - Erasure Coding** (FLR-EC) – M:N data redundancy (ORNL)
  - **Lustre Metadata Redundancy** (LMR1) – distribute and replicate services from MDT0000
- **2.19** features under discussion for development
  - **Metadata Writeback Cache** (WBC3) – existing directory tree support (WC)
  - **Lustre Metadata Redundancy** (LMR2) – ROOT directory mirroring to other MDTs

LNet Improvements

Whamcloud

Demand for IPv6 in cloud deployments as IPv4 addresses are exhausted

- IPv6 large NID support (LU-10391 SuSE, ORNL, HPE)
  - Variable-sized NIDs for future expandability (e.g. IB GUID addressing)
  - Interoperable with existing current LNDs whenever possible
  - Enhancements to LNet/`socklnd`/`o2iblnd` for large NIDs
  - Testing underway for 2.16.0
- Handle large NIDs in Lustre configuration/mounting code
  - 2.16 Mount, config logs, Imperative Recovery, Nodemaps, root squash, etc.

- 2.17 Detect added/changed server NIDs automatically (LU-10360)
- Simplified configuration for IPv6 NIDs by clients (LU-14668)
- Improve handling of MGS with many NIDs (LU-16738)
- Improve network transfer for sparse reads (LU-16897)

Hybrid IO: Improved Application IO Performance (2.17)

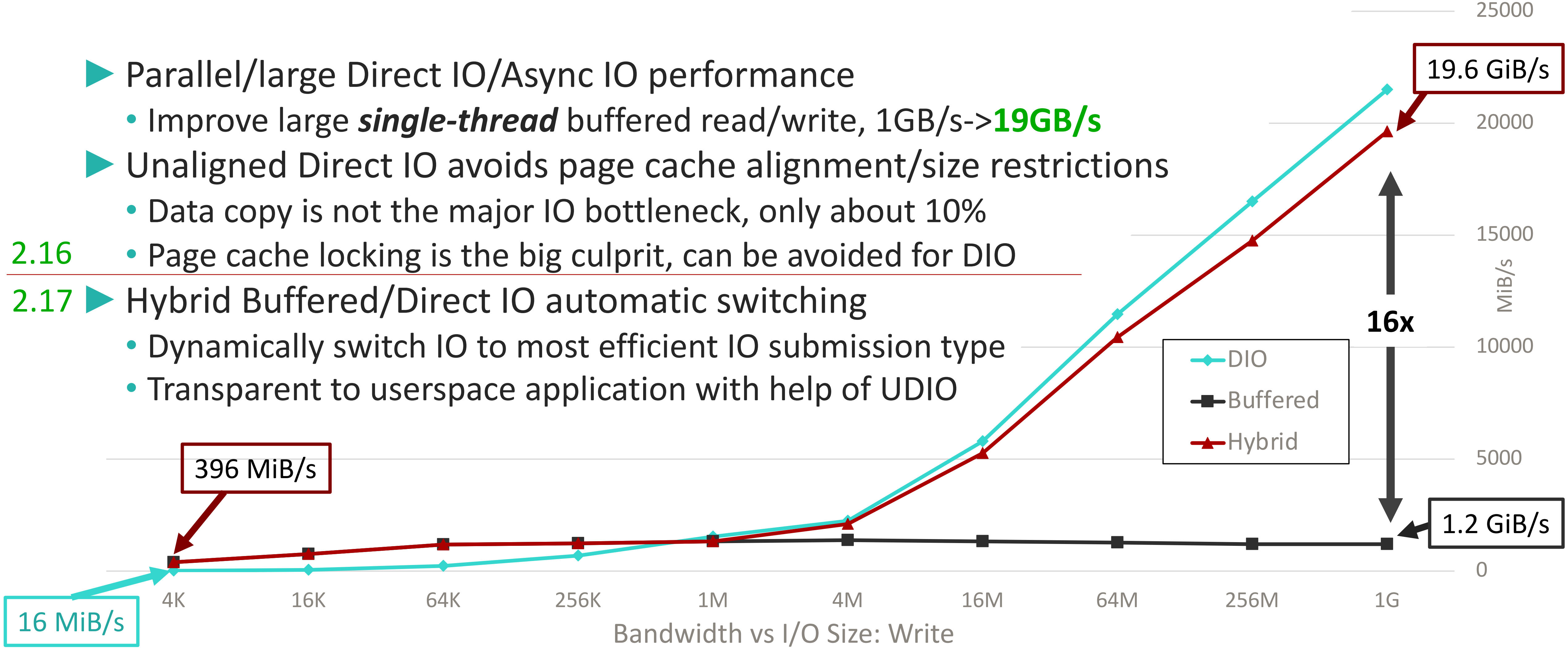
Client nodes are increasingly powerful (CPU, RAM, network) and data intensive

- Parallel/large Direct IO/Async IO performance
  - Improve large ***single-thread*** buffered read/write, 1GB/s->**19GB/s**
- Unaligned Direct IO avoids page cache alignment/size restrictions
  - Data copy is not the major IO bottleneck, only about 10%
  - Page cache locking is the big culprit, can be avoided for DIO

2.16

2.17

- Hybrid Buffered/Direct IO automatic switching
  - Dynamically switch IO to most efficient IO submission type
  - Transparent to userspace application with help of UDIO



Client-Side User Tools Improvements

Sometimes it's the small things that make a big difference

- ▶ `lfs df --mdt/--ost` shows only MDTs or OSTs (LU-17516 WC)
- ▶ `lfs find -xattr/attr` finds files with specific attributes (LU-15743 LU-16760 LANL)
- ▶ `lfs find -printf/ls` to better display found files (LU-7495 LU-15504 LANL, WC)
- ▶ `lfs find --skip` drop some found files for `lfs migrate` balancing (LU-17699 WC)
- ▶ `lfs mkdir –C` allows more directory stripes than MDTs, up to 5x (LU-12273 WC)
- ▶ `lfs setstripe –C -`*N* creates N (over)stripes per OST, up to 32x (LU-16938 HPE)

2.16 ▶ `lctl list_param --path` prints full pathname for data scraping (LU-17343 WC)

2.17 ▶ `lctl list_param --tunable` lists only tunable parameters (LU-11077 WC)

- ▶ `lctl get/list_param --merge` aggregates similar output lines (LU-14590 WC)
- ▶ `mount.lustre` can set client-local parameters at mount (LU-11077 WC)
- ▶ `lfs find/project/quota` allows named projects from `/etc/projects` (LU-13335 WC)

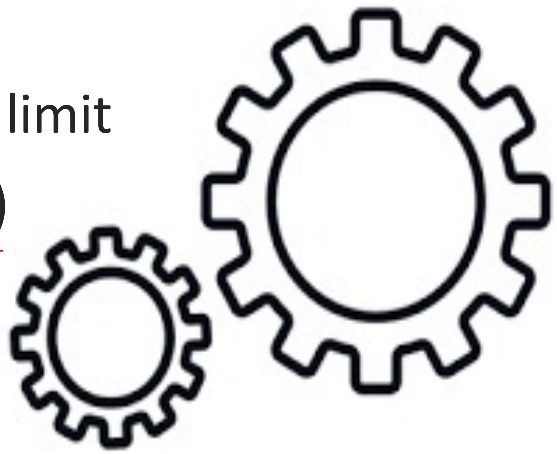
Client-Side Functionality Improvements

Ongoing ease-of-use and performance improvements for users and admins

- Remove 8192-device static limit, for multiple large mounts (LU-8802 Amazon)
- Ongoing code updates/cleanup for upstream 6.x kernels (ORNL, HPE, WC, SuSE)
- Allow specifying CPU cores to exclude from CPT list (LU-17501 WC)
  - 2.16 `options libcfs cpu_pattern="C[0,1]"` skips cores on each NUMA node

2.17
- Project quota aggregation into groups (LU-18222 WC)
  - Similar to OST Pool Quotas, allows project to be "nested" into one larger limit
- Client-side performance stats via statfs for each target (LU-7880 WC)

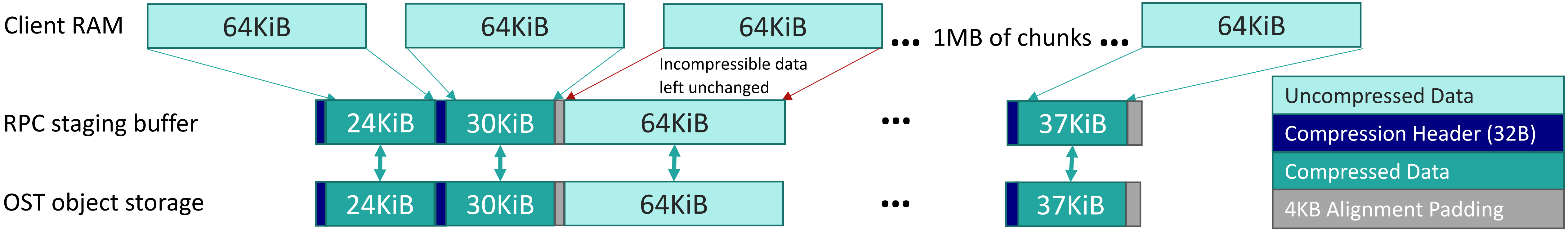
2.18
- FLR Erasure Coded files with delayed resync (LU-10911 ORNL)



Client-Side Data Compression (2.17+ WC)

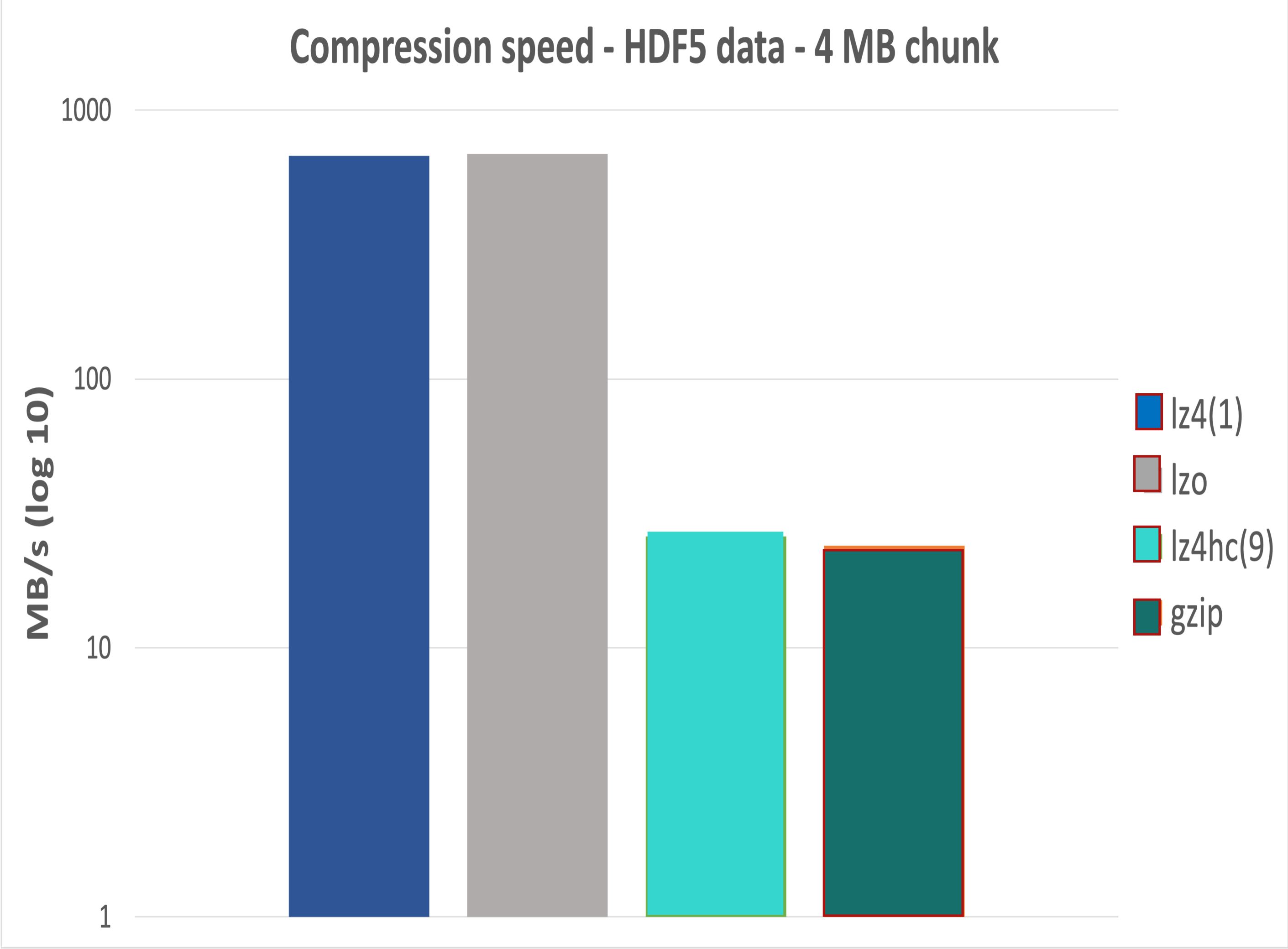
Increased capacity and lower cost per GB for all-flash OSTs

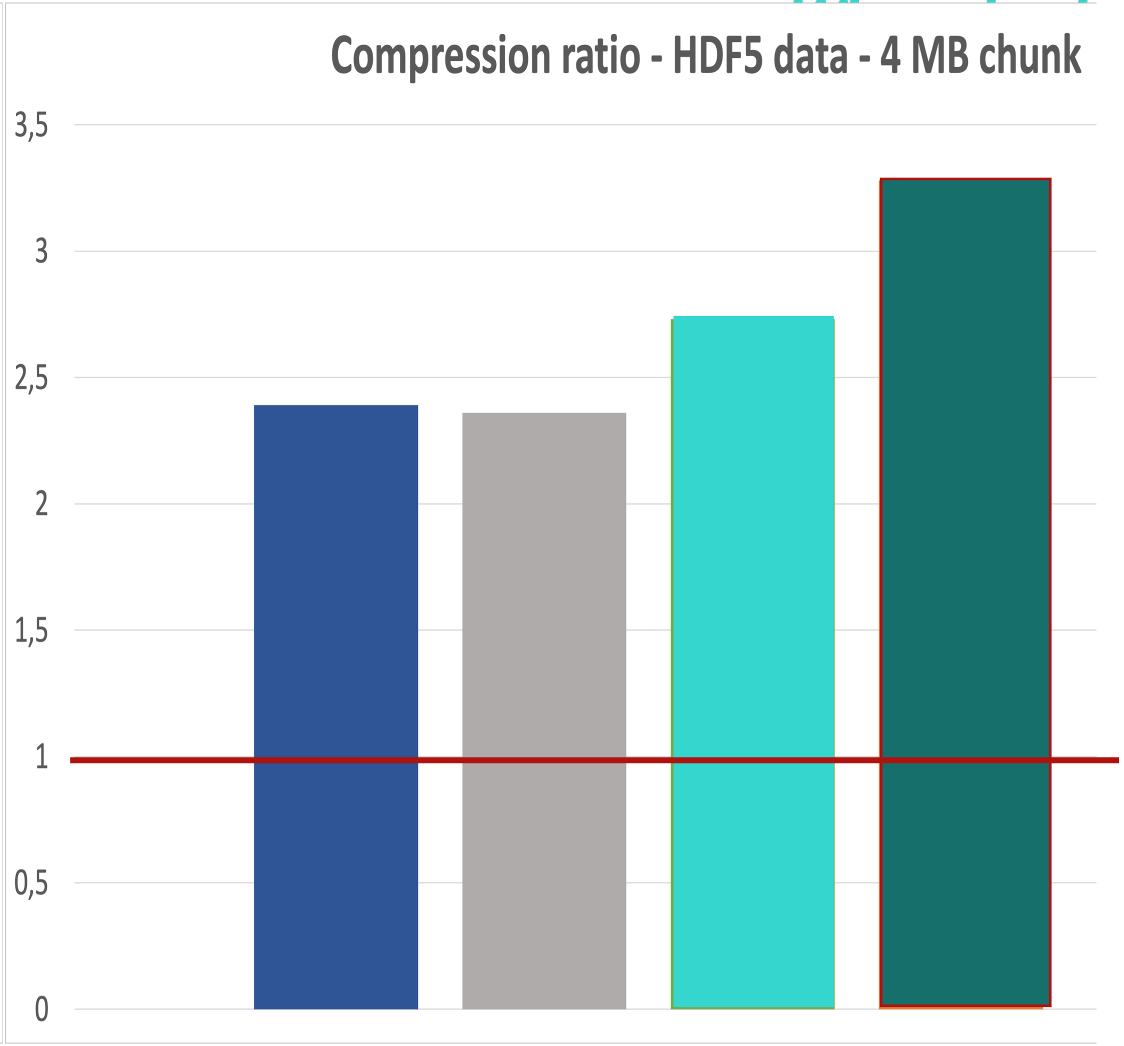
- Parallel (de-)compression of RPCs on client cores for GB/s speeds, **reduces server CPU load**
- (De-)Compress (`lzo`, `lz4`, `zstd,`…) RPC on client in chunks (64KiB-4MiB+)
  - **Per directory or file component** selection of algorithm, level, chunk size (PFL, FLR)
  - Keep "uncompressed" chunks as-is for incompressible data/file (`.gz`, `.jpg`, `.mpg`, …)



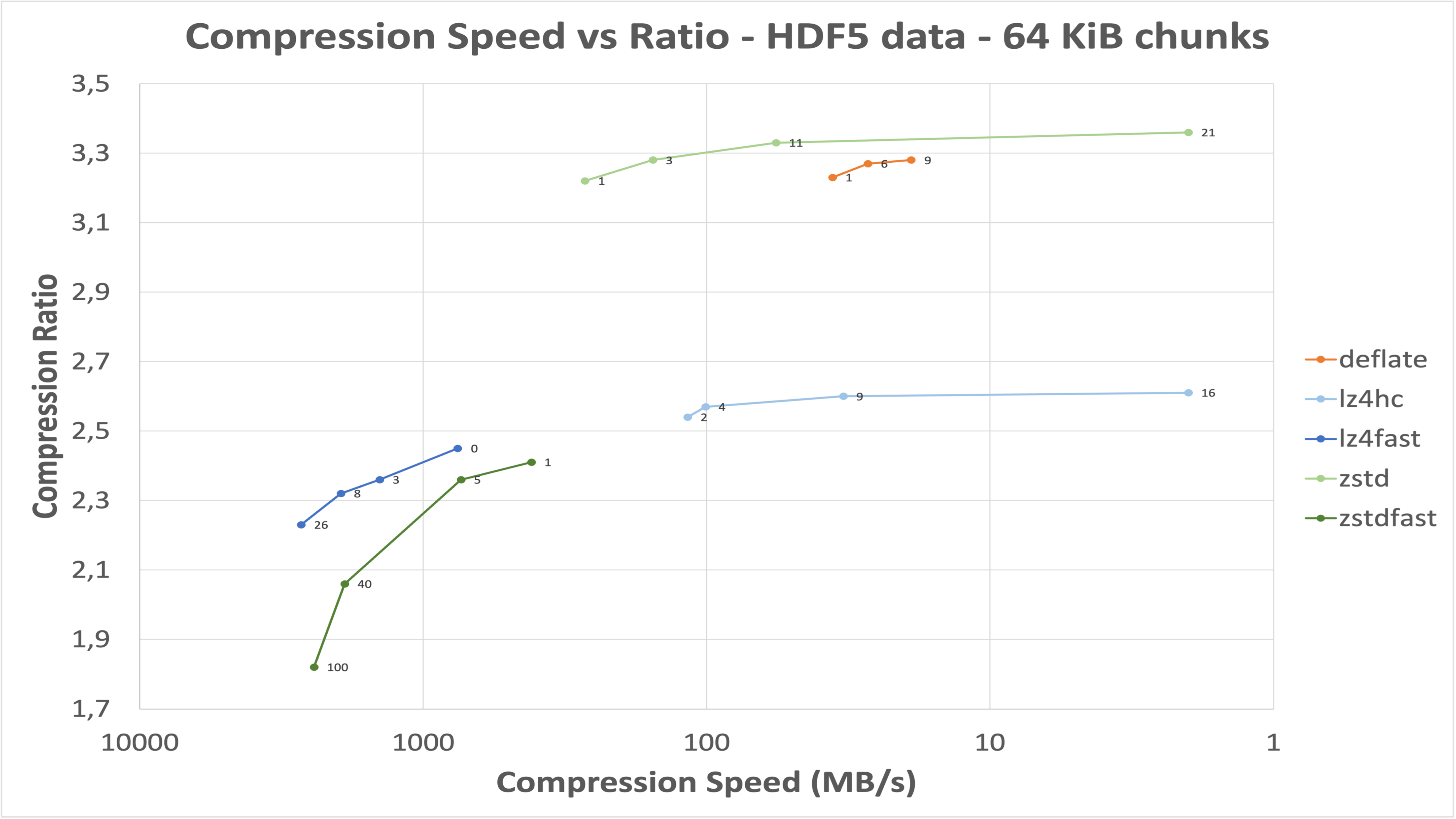
- Client writes/reads whole chunk(s), (de-)compresses to/from RPC staging buffer
  - Larger chunks improve compression, but higher decompress/read-modify-write overhead
- Could write uncompressed to one FLR mirror for random IO pattern
  - Data (re-)compression during mirror/migrate to second mirror on slow tier (via data mover)

Compression Algorithm Testing: HDF5 climate data





Expect 1.5x-4x compression ratio for (uncompressed) data



Compression Speed vs Ratio - HDF5 data - 64 KiB chunks
Compression Ratio
Compression Speed (MB/s)
3,5
3,3
3,1
2,9
2,7
2,5
2,3
2,1
1,9
1,7
10000
1000
100
10
1
deflate
lz4hc
lz4fast
zstd
zstdfast
21
11
3
1
9
6
1
16
9
4
2
0
3
8
26
1
5
40
100

Server-side Capacity and Efficiency Improvements

Ongoing performance and capacity scaling for next-gen servers and storage

- Optimized locking for rename-intensive workloads (Spark, …)
  - Same-directory file/subdirectory rename optimization (LU-12125 WC)
  - MDS parallel cross-directory file rename optimization (LU-17426 WC)
  - Move rename to separate portal to avoid blocking other RPCs (LU-17441 WC)
  - Rename of regular file across projid directories without copy (LU-13176 WC)
- OST object directory scalability for multi-PB OSTs
  - Reduced transaction size for many-striped files/dirs (LU-14918 WC)
  - 2.16 Handling billions of objects on a single OST (LU-11912 WC)
- 2.17 **OST writeback cache** for small, lockless, direct writes (LU-12916 WC)
  - Lower latency, small write aggregation, no lock ping-pong
  - Use ldiskfs delayed allocation (`delalloc`) until OST write is large enough, default 64KiB
  - Dynamic cache selection, complementary with client Hybrid Buffered/Direct IO

Server-side Usability Improvements

Ongoing improvements to usability and robustness for ease of management

- OST Pool Spilling avoid out of space with hybrid OST tiers (LU-15011 WC)
- More robust MDT-MDT recovery llog handling (LU-several WC, CEA)
- Read-only mount of OST and MDT devices (LU-15873 WC)
- Hardening of online MDT/OST addition under load (LU-12998 LU-17334 WC)
  - MDT/OST "`-o no_create`" mount option to avoid directory/object creation on new OSTs
- `lljobstat` utility for easily monitoring "top/bad" jobs on MDT/OST (LU-16228 WC)
  - Add IO size histograms to `job_stats` output, handle bad job names better
- Store JobID into `user.job` xattr on inodes at create (LU-13031 LANL)
  - Provenance tracking for files/objects, post-mortem analysis of file creation issues

2.16

2.17

- Enable default PFL layout on newly-formatted filesystems (LU-11918)
- Default NRS TBF rule(s) to keep "bad" jobs in check out of the box (LU-17296)

Ongoing ldiskfs and e2fsprogs Improvements

Whamcloud

- Persistent `TRIMMED` flag on block groups during `fstrim` (LU-14712 WC)
  - Avoid useless `TRIM` commands on device after reformat and remount
- `mkfs.lustre` to use `sparse_super2` feature for new filesystems (LU-15002 WC)
  - 2.16 More efficient metadata for filesystems > 256TiB, optimized for hybrid storage layout

---

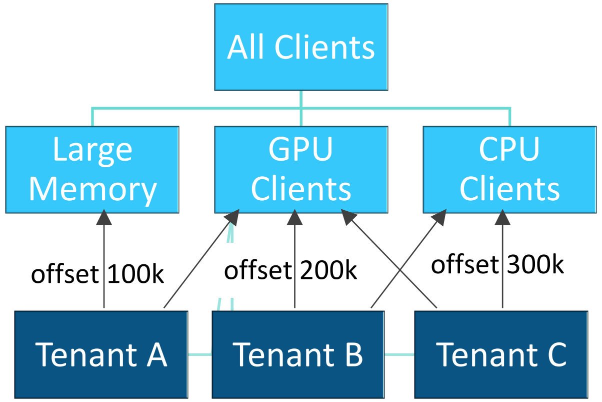
2.17

- More efficient ldiskfs mballoc for large filesystems (LU-14438 Google, IBM, WC)
  - Backport improved list-/tree-based group selection from upstream kernel
- Hybrid ldiskfs LVM storage devices (NVMe+HDD) (LU-16750 WC)
  - Allow storing metadata on NVMe at start of device, data on HDDs at end of device
- Enable ldiskfs delayed allocation for writeback cache (LU-12916 WC)
  - Allow aggregating small writes in server RAM instead of read-modify-write to client
- Parallel e2fsck for pass2/3 (directory entries, name linkage) (LU-14679 WC)

Improved Data Security and Multi-tenancy

Increasing demand to isolate users and their data for legal/operational reasons

- Configurable capabilities mask (LU-17410 WC)
  - Defaults to all client capabilities disabled for security
- Cgroup/memcg memory limits for containers on clients (LU-16671 WC, HPE)
- 2.16 Nodemap Role-Based Admin Controls (fscrypt, changelog, chown, quota) (LU-16524 WC)
- 2.17 Nodemap offset range for multi-tenant UID/GID/PROJID config (LU-18109 WC)
- Dynamic/hierarchical nodemap configuration (LU-17431 WC)
  - In-memory nodemap configuration for short-lived group (batch job)
  - Inherit parameters from static parent nodemap for most settings
- Encrypted fscrypt backup/restore without key (LU-16374 WC)



Metadata Server Improvements (WC 2.15+)

Whamcloud

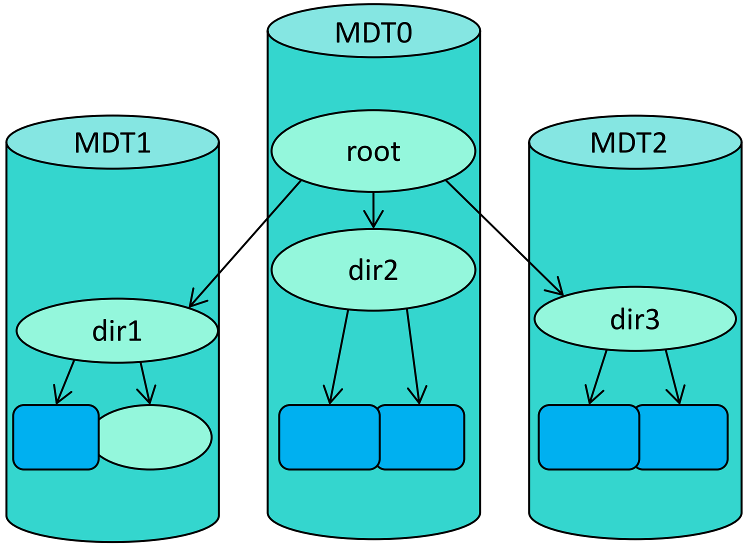
Improve usability and ease of DNE metadata horizontal performance/capacity scaling

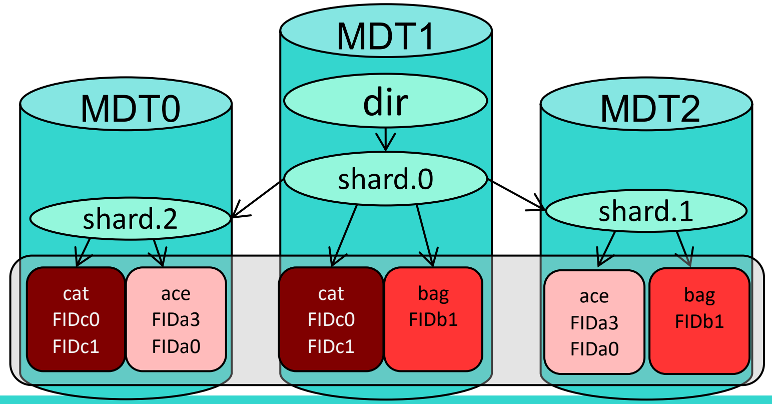
- More robust DNE MDT llog recovery (LU-16203, LU-16159)
  - Handle errors and inconsistencies in recovery logs better
- Store JobID in "`user.job`" xattr at create (LU-13031, LANL)
- Exclude list for pathnames from remote mkdir (LU-17334)
  - 2.16 Ensures that rename from subdir is local to parent

- 2.17 DNE locking, remote RPC optimization (LU-15528)
  - Distributed transaction performance, reduce lock contention

---

- 2.18 Lustre Metadata Robustness/Redundancy (LU-12310)
  - **LMR1** distribute/replicate MDT0000 *services* to other MDTs
  - **LMR2a** mirror ROOT/ directory to multiple MDTs (via setdirstripe)
  - **LMR2b/c** mirror subdirs/files to multiple MDTs (configurable per dir)
  - **LMR3** complex recovery (degraded writes, full MDT rebuild)





IO500 Performance History

Performance improvements go beyond what hardware upgrades have provided

- Hardware Configs
  - 4 x Lustre MDS+OSS
    - 12 x CPU core
    - 142GB RAM
    - 1 x HDR200 InfiniBand
    - 24 x NVMe (shared)
  - 10 x Lustre Client
    - 16 x CPU core
    - 96GB RAM
    - 1 x HDR100 InfiniBand

Storage Platform	1x ES400NV		1x ES400NVX	1x ES400NVX2			
	Pre-SC19	SC19	ISC20	ISC22	SC22	ISC23	ISC23/PreSC19
Lustre Version	Untuned	2.12.58+	2.13.53+		2.15.51+	2.15.55+	
ior-easy-write	25.8	28.62	37.56	55.95	58.07	57.88	2.2x
ior-easy-read	39.9	41.72	45.95	83.86	77.56	79.08	2.0x
ior-hard-write	2.7	2.96	2.77	5.02	5.27	5.38	2.0x
ior-hard-read	8.9	42.19	40.81	39.73	49.36	50.77	5.6x
find	1,735.4	810	1,698.00	6,248.55	12628.78	13,229.11	7.6x
mdtest-easy-write	143.8	152.84	157.22	270.04	312.9	344.70	2.3x
mdtest-easy-stat	455.0	451.97	453.51	740.01	1,278.50	1,276.31	2.8x
mdtest-easy-delete	88.5	132.76	135.09	223.61	272.64	311.16	3.5x
mdtest-hard-write	32.3	79.65	90.47	119.41	157.4	199.36	6.1x
mdtest hard-read	44.9	172.59	169	194.33	238.82	391.09	8.7x
mdtest Hard-stat	20.4	449.93	446.75	514.36	1,214.03	1,105.33	54.1x
mdtest Hard-delete	16.3	75.15	76.94	101.98	122.44	112.58	6.8x
Bandwdith	12.68	19.65	21.02	31.10	32.90	33.43	2.6x
IOPS	91.41	207.6	232.6	368.4	544.2	603.39	6.6x
Score	34.05	63.87	69.93	107.0	133.8	142.03	4.1x

https://io500.org/submissions/view/657

Optimized Directory Traversal (WBC1) (WC 2.16+)

Improved access speed and efficiency for large directories/trees

- IO500 `mdtest-{easy/hard}-stat` performance improved **77%**/**95%**

▶ **Batched RPC** infrastructure for multi-update operations (LU-13045)

- Allow multiple getattrs/updates packed into a single MDS RPC
- More efficient network and server-side request handling

▶ **Batched statahead** for `ls -l`, `find`, etc. (LU-14139)

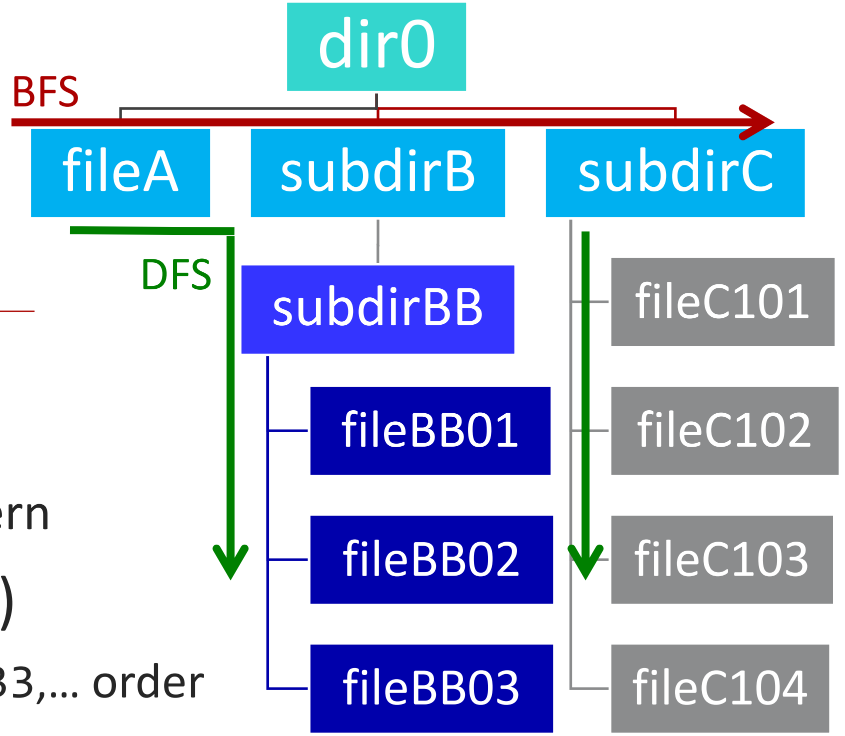
2.16 • Aggregate getattr RPCs for existing statahead mechanism

2.17 ▶ **Cross-Directory statahead** pattern matching (LU-14380)

- Detect breadth-first (**BFS**) depth-first (**DFS**) directory tree walk
- Direct statahead to next file/subdirectory based on tree walk pattern

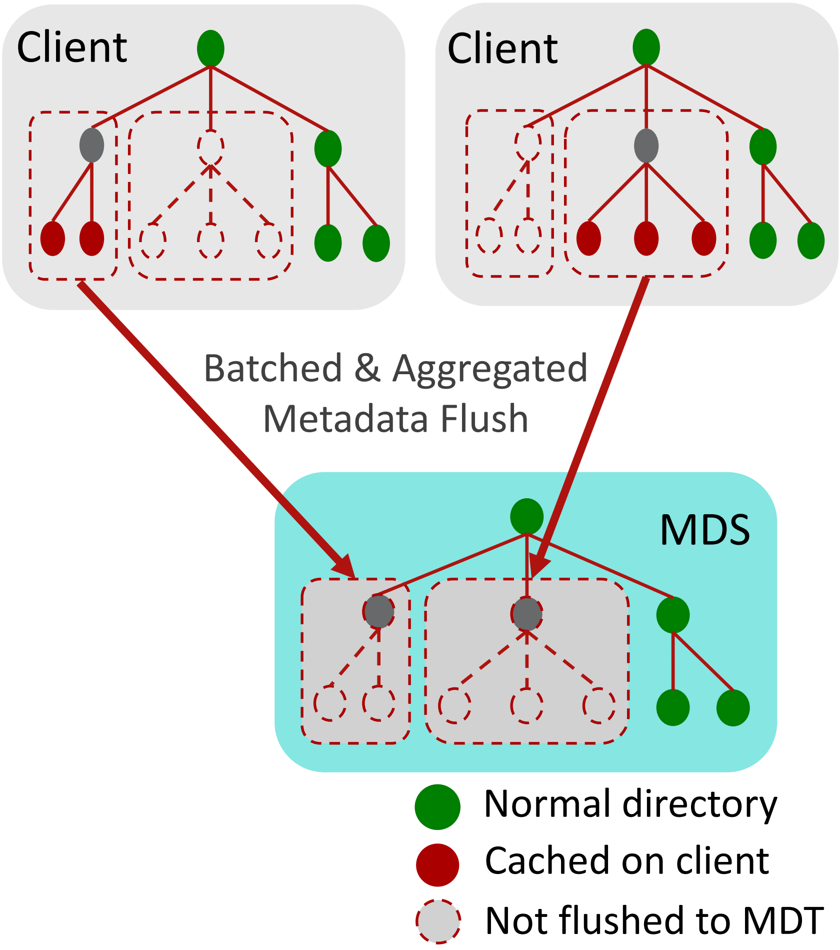
▶ **Strided pattern detection** for ordered `stat()` (LU-14380)

- e.g. `file00001`,`file001001`,`file002001`… or `file1`,`file17`,`file33`,… order



Metadata Writeback Cache (WBC2) (WC 2.17+)

Whamcloud

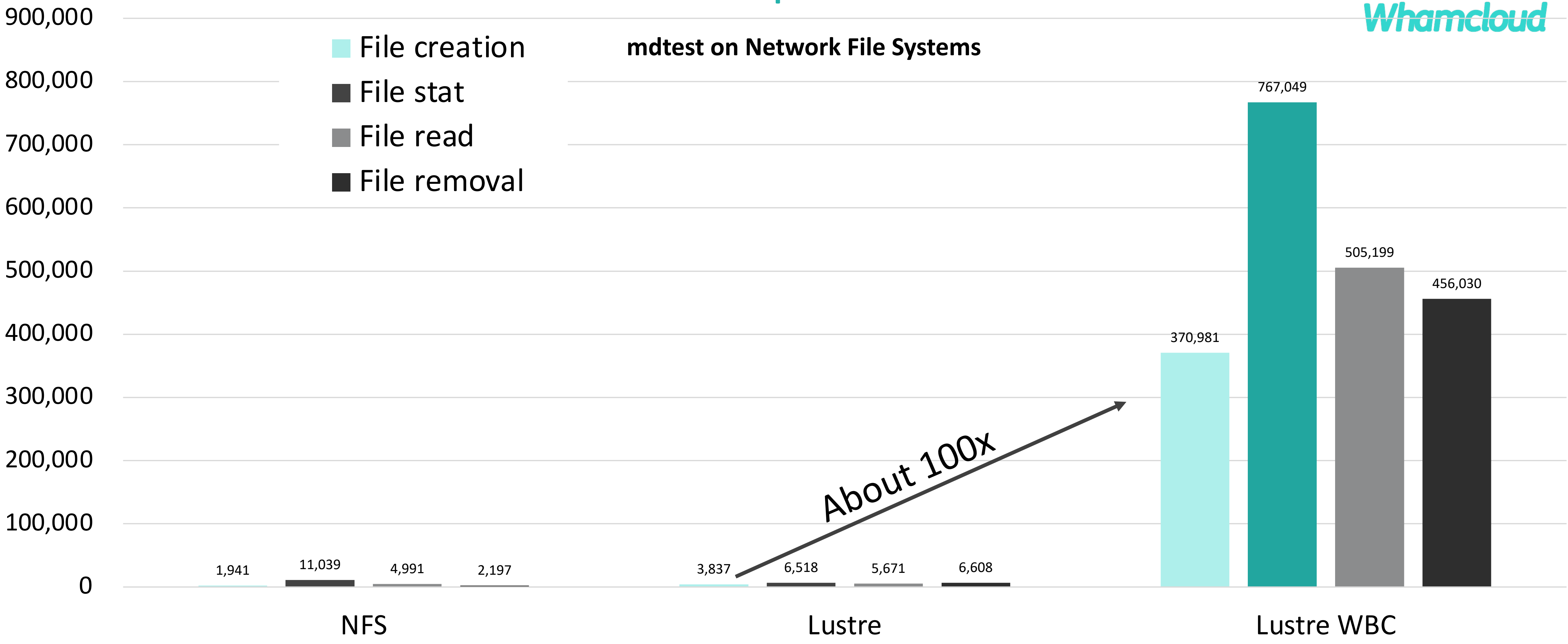


10-100x speedup for single-client create-intensive workloads

- Gene extraction/scanning, untar/build, data ingest, producer/consumer

- Create new dirs/files **in client RAM without RPCs**
  - Lock new directory exclusively at `mkdir` time
  - Cache new files/dirs/data in RAM until cache flush or remote access
- **No RPC round-trips** for file modifications in new directory
- Batch RPC for efficient directory fetch and cache flush
- **Files globally visible on remote client access**
  - Flush top-level entries, exclusively lock new subdirs, unlock parent
  - Flush rest of tree in background to MDS/OSS by age or size limits
- Productization of WBC code well underway
  - Some complexity handling partially-cached directories
  - Need to integrate space usage with quota/grant

Metadata WBC Performance Improvements



Lustre: DDN AI400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre clients: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local File System on SSD: Intel SSDSC2KB240G8

Client FLR Erasure Coded Files (2.18+, ORNL)

Whamcloud

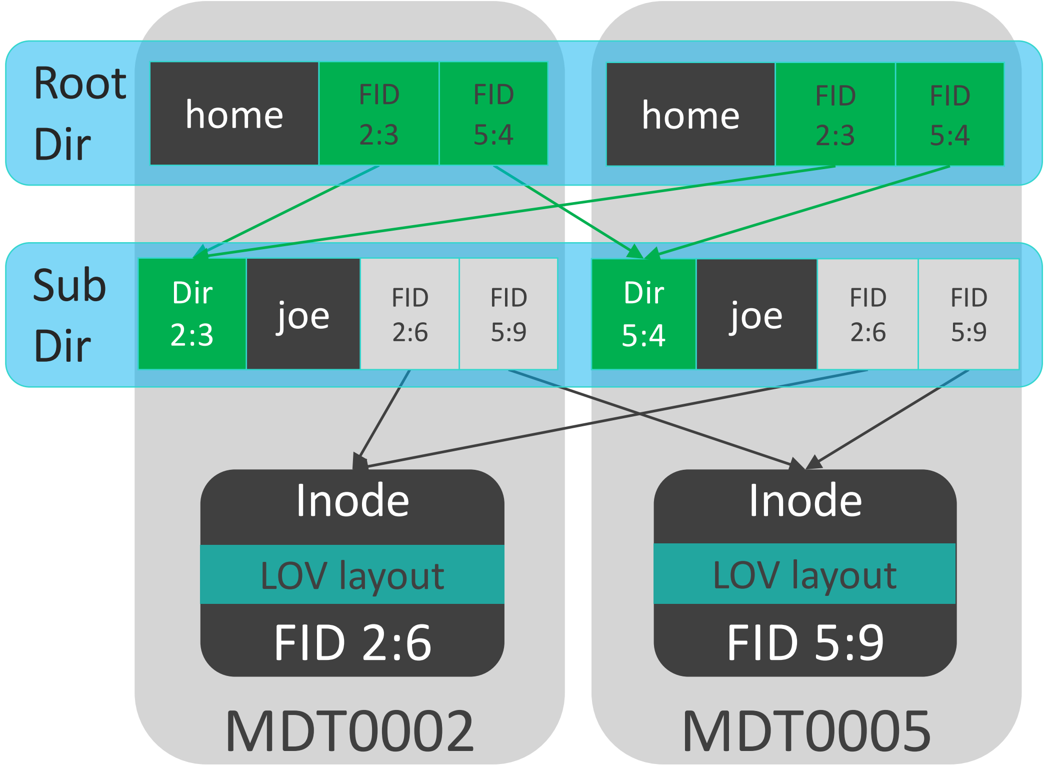
- Erasure coding adds data redundancy without 2x/3x mirror overhead
  - Improve data availability above hardware and network reliability
- Add erasure coding to new/old striped files ***after*** write done
  - Delayed redundancy avoids overhead during initial application write
- For striped files - add N parity per M data *stripes* (e.g. 16d+3p)
  - Fixed **RAID-4** parity layout *per file*, declustered by file, CPU-optimized EC code (Intel ISA-L)
  - Parity declustering avoids IO bottlenecks, CPU overhead of too many parities
    - e.g. split 128-stripe file into 8x (16 data + 3 parity) with 24 total parity stripes

dat0	dat1	…	dat15	par0	par1	par2	dat16	dat17	…	dat31	par3	par4	par5	…
0MB	1MB	…	15M	p0.0	q0.0	r0.0	16M	17M	…	31M	p1.0	q1.0	r1.0	…
128	129	…	143	p0.1	q0.1	r0.1	144	145	…	159	p1.1	q1.1	r1.1	…
256	257	…	271	p0.2	q0.2	r0.2	272	273	…	287	p1.2	q1.2	r1.2	…

Lustre Metadata Redundancy (2.18+)

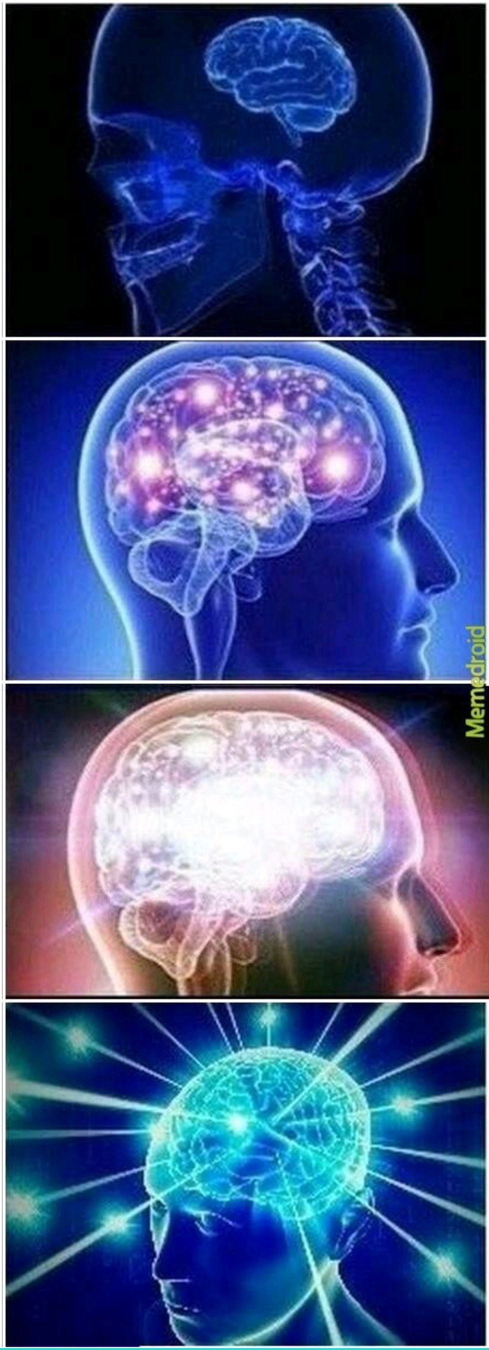
In early discussion and architecture stages

- LMR1a: Replicate services to other MDTs
  - Mirror FLDB, Quota, `flock()` scaling over MDTs
- LMR1b: DNE transaction performance
  - Remove excessive transaction ordering/sync
  - Improves **all** DNE operation performance
- LMR2: Replicate top-level dirs for availability
  - 2a: ROOT/ (rarely changed) mirrored to other MDTs
  - 2b: Subdirectories mirrored to 2+ MDTs (via setdirstripe)
  - 2c: Per-file metadata replication in a later stage
  - 2d: `e2fsck` to allow directory entries with multiple FIDs
- LMR3: Complex recovery handling
  - Write to directory while mirror offline, full MDT rebuild
- LMR4: LFSCK to repair/rebuild inconsistent mirrors



On the Evolution of IO Interfaces

- POSIX has been the standard IO interface for decades
  - Protects significant investment in developed applications and tools
  - Avoids applications chasing interface-of-the-month and expensive rewrites
  - Data portability via protocol export (Lustre, NFS, SMB, S3, …)
- **Opt-in** API *extensions* for apps with special performance needs
  - Relaxed semantics/interfaces when/where applications need/understand it
  - Avoids issues with apps depending on behavior - ***which* subset of POSIX is OK?**
  - Data stored and continues to be accessible via standard APIs afterward
  - Applications can leverage extensions via common libraries or directly
- Keeping up with hardware speedups demands continual optimization
  - Unaligned IO, cross-dir/file prefetch, WBC improves speed transparently
  - Asynchronous meta/data ops via Linux `io_uring`, batched file create
- POSIX will continue to be the most common interface going forward



Whamcloud

Thank You!

ddn