



Paris, Sept. 30 & Oct. 01, 2026



Lustre 2.18 and Beyond

Andreas Dilger, Lustre Principal Architect

Planned Feature Release Highlights

2.18 has multiple features landing to the development branch

- **File Level Erasure Coding - Read Only (FLR-ECRO)** – M:N file data redundancy (WC, ORNL, TLC)
- **Persistent Client Cache – Read Only (PCC-RO)** – "Tier 0" cache in client NVMe (WC, TLC)
- **Fault Tolerant MGS (LMR-FTM)** – replicate MGS config and recovery across MDS (WC)

2.19 (3.0?) has several features under development and design

- **Client-side Data Compression (CSDC)** – reduce network and storage usage/cost (WC, TLC)
- **Trash Can/Undelete (TCU)** – allow file recovery after accidental/malicious deletion (WC)
- **Lustre Metadata Redundancy (LMR-MDT0)** – MDT0000 services redundancy on other MDTs (WC)
- **Reduced Recovery Time (RRT)** – reduce recovery time after target failover (TLC)
- **File Level Immediate Write Mirror (FLR-IWM)** – redundancy as clients write files (WC, TLC)

2.20 (3.1?) features under discussion for development

- **Lustre Metadata Redundancy (LMR-ROOT)** – ROOT/ directory mirroring to other MDTs
- **Metadata Writeback Cache (WBC2)** – single-client metadata speedup (WC, TLC)
- **Lustre Metadata Redundancy (LMR-DIR)** – subdirectory mirroring to other MDTs

Client File Level Erasure Code Read-Only (2.18+ WC, ORNL, TLC)

Dramatic improvement in data availability with low overhead

FLR-ECRO - Redundant *file data* without 2-3x mirror overhead ([LU-10911](#))

- Mostly benefits large **read-mostly** files, improve redundancy/availability

Large striped files - add N+M RAID Sets (e.g. 8d+2p or 16d+3p)

- Declustered across files, CPU-optimized ([Intel ISA-L](#)), OST loss has small read delay (seconds)

Add EC to new (*and existing!*) striped files **after** write finished

- **Delayed redundancy** avoids overhead during initial write, **needs explicit resync by utility!**
- Use "lfs setstripe/mirror create ... **--ec N+M**" for new files, or default directory layout
- Use "lfs mirror extend ... **--ec N+M**" to add EC to existing files
- Use "lfs mirror resync FILE" to resync parity mirror (script following Changelog)

Immediate Write Mirror and Immediate Write EC developed in later phases ([LU-13643](#))

Data component	dat0	dat1	...	dat7	dat8	dat9	...	dat15	...
	0MB	1MB	...	7M	8M	9M	...	15M	...
	128	129	...	143	144	145	...	159	...
	256	257	...	271	272	273	...	287	...



Parity Component	par0	par1	par2	par3	...
	p0.0	q0.0	p1.0	q1.0	...
	p0.1	q0.1	p1.1	q1.1	...
	p0.2	q0.2	p1.2	q1.2	...

FLR-EC Developments with Broader Usefulness

OST Failure Domain identifies OSTs sharing external resources ([LU-19066](#))

- Integer for OSTs that share enclosure, PSU, switch, ..., `obdfilter.*.failure_domain`
- Can check domains on client `"lfs df --output domain"`
- MDS object allocator uses different domains for FLR-EC and mirror objects if set

Single-OST-object migration/layout swap ([LU-20534](#))

- Avoids migrating whole file when one OST is offline, works for any file layout

Quota cache on MDS0 to handle (temporarily) offline OSTs ([LU-20354](#))

- Quota usage cannot change while OST offline, so cache is valid until it returns

`truncate(0)` replaces files layout if it has offline OST(s) ([LU-20211](#))

- Allow immediate write to file instead of waiting for recovery

Persistent Client Cache (PCC-RO) (LU-10499 2.18 WC, TLC)

Leverage client-local storage to accelerate workloads ("Tier 0")

Reduce latency, avoid network traffic to servers, fast small/unaligned IOPS

- Ideal for read-heavy workloads reusing files multiple times (e.g. training, KV Cache)
- Cache large shared libraries (Python!), large datasets, current training data, etc.

PCC integrates Lustre with **local cache storage** on client (NVMe, NVRAM, CMX+BlueField)

- A local directory (e.g. /var/cache/lustre) or mountpoint is used on the client
- **File data mirrored by client** on first read based on PCC policy, or explicit request

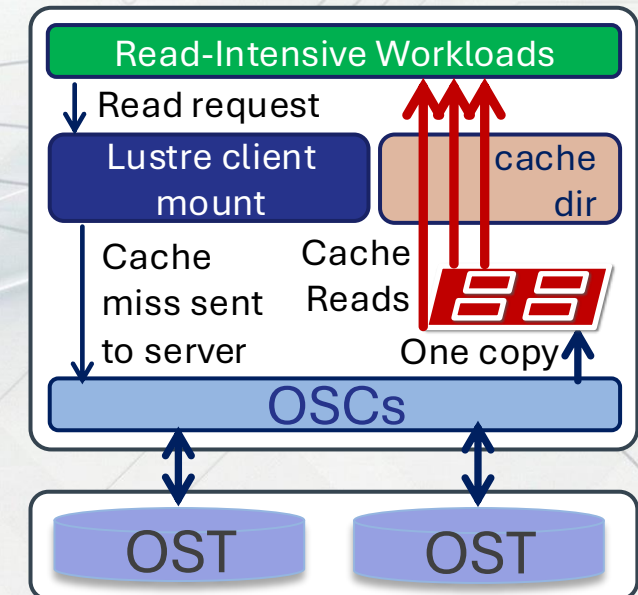
Apps transparently read from local file on cache hit, or from Lustre

- Cache files maintain fscrypt encryption if used in Lustre
- Can "pin" important datasets into cache, others cached on demand

Some work remaining to finish PCC-RO for 2.18 release

- **Userspace tool for cache space management, removal of old files**
- Simplify config/usage (e.g. "-o cache=/var/cache/lustre")

PCC-RW **prototype** exists, more work needed for production

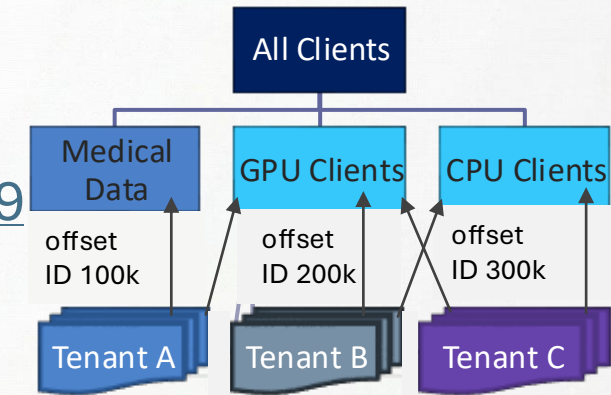


Data Security and User Isolation via Nodemaps

(2.18+)

Strongly isolate user/tenant data for legal/operational reasons

- Auto-load SSK `.lgss` key from client mountpoint ([LU-19326 WC](#))
- **Client nodemap mapping only by SSK/KRB5 (no NIDs!)** ([LU-19079](#))
- Multiple SSK **client** keys for different Nodemaps ([LU-19079 WC](#))
- Multiple SSK **server** keys for key rotation/expiry ([LU-8229 WC](#))
- Allow setting RBAC roles on default nodemap ([LU-19975 WC](#))
- Add `foreign_dir`, `lqa`, `projid` ops to nodemap RBAC ([LU-19884](#), [LU-19963 TLC, WC](#))
- **Quota Aggregation** (LQA) allows range of IDs to be accounted together ([LU-18222 WC](#))
- 2.18 • New GSS IAM authentication mechanism ([LU-19921 Google, WC, TLC](#))
- 2.19 • Filter client NID discovery by Nodemap and NUMA node ([LU-20290 TLC, WC](#))
- Block client mounts based on client build version ([LU-17217 TLC](#))
- Block MGS target registration after `allow_register` timeout ([LU-18857 WC, TLC](#))
- Encrypted `fscrypt` backup/restore without user key ([LU-16374 WC, TLC](#))



Lustre Quota Aggregate (LQA)

(LU-18222 2.18 WC)

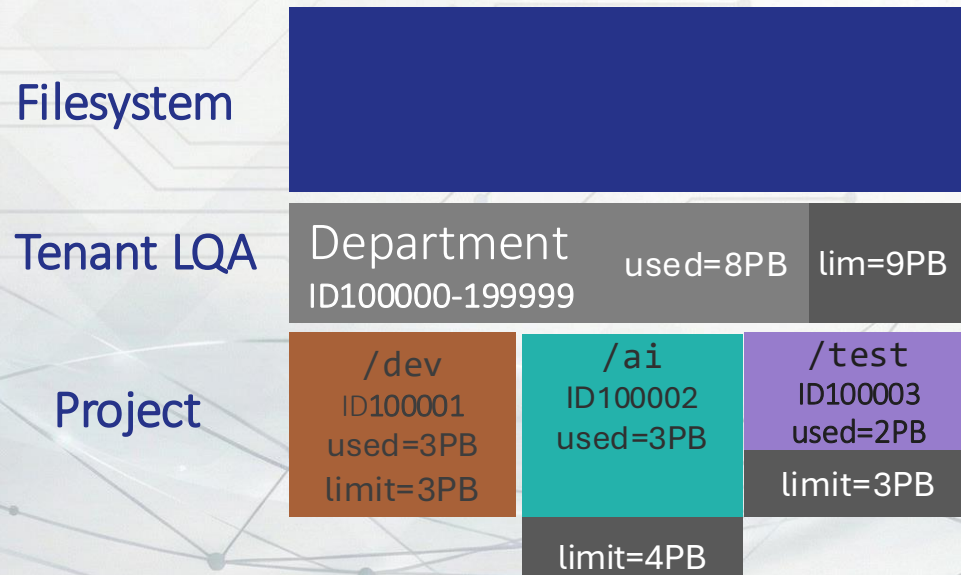
Aggregates quotas across a ranges of IDs

Range quota independent of limit on individual UID/GID/PROJID

- **Lowest** quota limit reached for any ID is enforced first

Allows multi-level quota limits for Nodemaps, if mapped to same ID mapping range

- Delegate usage of Project, User, Group quota to tenant administrators



Aggregated Quota Limit per Nodemap

- LQA defined over PROJID range [100000-199999]
- Limit for cumulative usage over all IDs in range exceeds LQA limit
 - If over limit, return EDQUOT for **any ID** in the range

Separate project quotas on directories

- If directory project quota reached first, user gets EDQUOT
- **/dev hit 3GB project limit**, cannot write any more
- **/ai, /test together have 1PB for LQA limit**

Fault Tolerant MGS (LMR-FTM)

(LU-17819 2.18 WC)

Replicate MGT to multiple server nodes for better availability

- Clients can get config logs and server alerts from **any server node**
- Reduces mount time/timeouts, reduced MGS load in large clusters
- MGS Imperative Recovery works even if MGS0 primary node restarts

Initial steps for replication of other MDT0000 databases

- FLDB, Quota, Nodemap, ...

Implementation undergoing review and testing



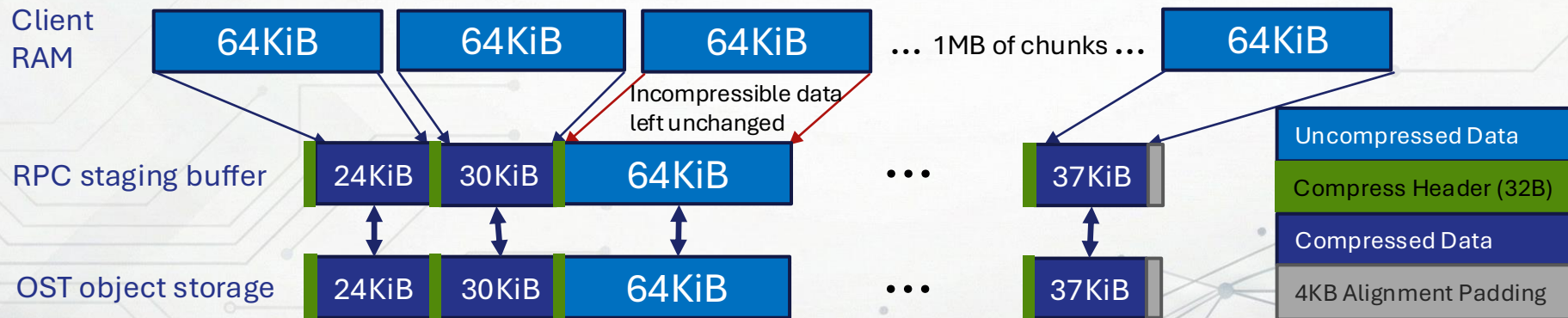
Client-Side Data Compression (LU-10026 2.19 WC, TLC, MS)

Parallel (de-)compression of RPCs on clients for GB/s speeds

- Lots of client cores, reduced server CPU load, network bandwidth

(De-)Compress (gzip, 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 write/read whole chunk(s), (de-)compress in RPC bounce buffer

- Large chunk size improves compression, but more overhead if file read non-sequentially

Could write uncompressed to one FLR mirror for random IO pattern, recompress to another

- Data (re-)compression during mirror/migrate to slow tier mirror (via data mover)

Reduced Recovery Time

(2.19 TLC, WC, Google)

Have your cake and eat it too?

Lustre IO typically very fast due to stateful clients and async writes ...

- Reduce write time during normal operation (avoid sync IO = less application waiting)
- ... but servers need to recover client state if they fail to avoid losing unwritten client data
- Server failover "pauses" filesystem access during this time (5s-600s, depends on load/scale)
- Delays over 10-15s unacceptable for some workloads, *even if saving hours of IO time per day*
- 10000+ clients can DDOS on Lustre servers, "just wait less" is not a robust solution

Multiple areas need improvement to avoid application "pause" during failover

- Better stats where server recovery time is being spent ([LU-18681](#) TLC, Google)
- 2.18 • FLR-mirror, FLR-ECRO – will hide target failover/recovery delay for file *reads* ([LU-10911](#) WC, TLC)
- 2.19 • FLR-IMW/ECWR - will hide target failover/recovery delay for file *writes* ([LU-19988](#) TLC, WC)
- LMR directory/inode mirroring will hide failover delay for file/directory access ([LU-17818](#))

Scheduled Maintenance Mode for fast failover, limit recovery time ([LU-20359](#) TLC)

- Clients notified to flush cache, servers sync all changes immediately to minimize RPC replay
- Unresponsive clients evicted **before** server unmount, server active reconnect to clients on mount

Lustre Find Utility (LFU)

(2.19+ TLC)

Lustre integrated/optimized namespace scanning and reporting

- Server-based target scanner for speed ([Lester the Lustre Lister](#), OSD OI Scrub, Changelog)
- Structured binary Object Stream for efficiency and flexibility (FlatBuffers, Cap'n Proto?)
- **Parallel at multiple levels, minimizes data transfer/translation during scan**

Modular stackable interface to allow component re-use

- Merge Object Streams while scanning multiple targets in parallel
- Remote bulk RPC transport to client (with appropriate access control filters)
- Efficient zero-copy lockless ring-buffer export from kernel to userspace

Pluggable filter rules to isolate objects and attributes of interest

- Filter by attrs (ranges), IDs (range, list), xattrs (regex), MDT/OST (range, list), pool, ...
- Aggregates of attributes for efficient reporting/filtering (min (+list), max (+list), count, sum, histogram, ...)
- Selectively pass only attributes of interest to consumers (path, FID, attrs, everything)

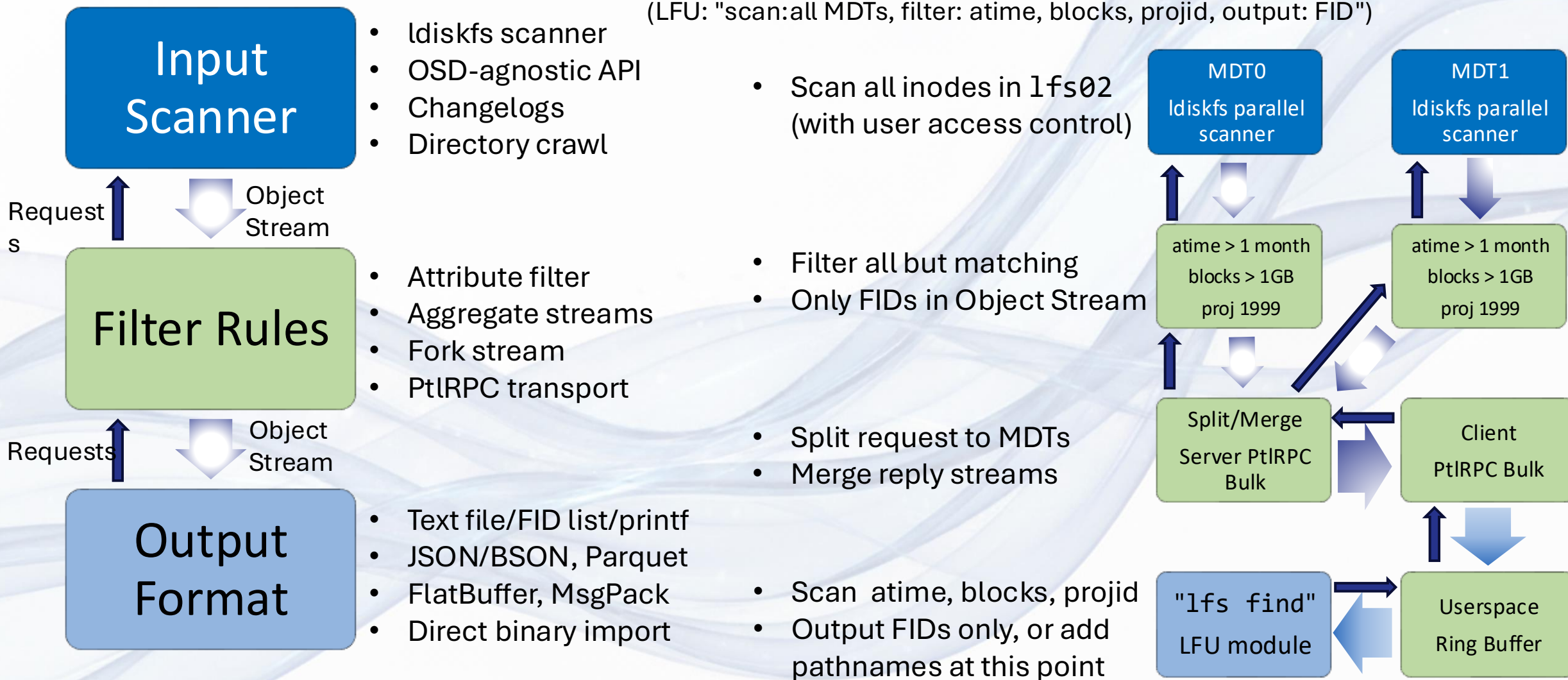
Flexible Output modules to consume Object Stream for end user

- "lfs find" -> text filename/FID list; JSON, CSV, Parquet and other structured text/binary formats
- Direct Object Stream to utilities for FLR mirror/EC resync, Trash Can, PCC-RO cache, external (RBH, GUFI, ...)



LFU Data Flow Overview

```
lfs find /lfs02 -atime +30d -blocks +1G -projid 1999 -printf %LF
(LFU: "scan:all MDTs, filter: atime, blocks, projid, output: FID")
```



Trash Can/Undelete

(LU-18465 2.19 WC)

Allow files/directories to be undeleted after `rm/rmdir`

- Rescue users from fat-finger mistakes, broken scripts, bad actors
- Handle “`rm -r`” properly to allow whole-tree recovery

Deleted files in trash are flagged and treated specially

- Removed from user/group/project quota and `df` usage
- Files in trash cannot be written to avoid abuse

Virtual `.Trash` directory visible in every directory

- Can use normal tools to list and recover files or directories
- `.Trash` is *hidden* from normal directory listing (no backup, app issues)

Users can view and recover their own files until cleanup

- Configurable time/fullness before cleanup (e.g. max age = 7d, 80% full)
- Complex cleanup policies run in userspace (e.g. by user, project, nodemap)



Client-Side Functionality Improvements

Ease-of-use and performance improvements for end users and admins

Ongoing updates for 7.x kernels and later upstreaming (ORNL, HPE, TLC)

RISC-V architecture support and ongoing build bot ([LU-19929](#) OpenChip)

Kernel folios (page extents) for large IO/cache ([LU-17916](#) HPE)

- Significant potential for buffered IO speedups (5-10x) from reduced pagecache overhead

Separate `max_mb_per_rpc_{read,write}` tunables ([LU-18635](#) Rahul!)

Improve LDLM lock cache to LFRU for better scaling, aging ([LU-11509](#) WC)

Latency by RPC size per target in `osc.*.io_latency_stats` ([LU-18933](#) WC)

2.18 Allow PFL layouts for O_APPEND files ([LU-12738](#), [LU-19722](#) HPE, TLC)

2.19 OverlayFS support for Lustre mountpoints ([LU-20253](#) TLC)

Client-side target performance stats via OBD_STATFS ([LU-7880](#) George!)

Client-side negative xattr cache ([LU-17238](#) AWS)

Client cache for sparse file reads ([LU-19469](#) WC)





Client-Side User Tools Improvements

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

lctl find_param to locate parameter names by regexp ([LU-18418 WC](#))

man4/ pages for **lctl get/set_param** parameters (LU-*many* WC, TLC, MSFT)

lctl help_param to show man4 page for matching parameter ([LU-17231 WC](#))

lfs_migrate script merged into **lfs migrate** (-A, --restripe) ([LU-16449 WC](#))

llapi_*() functions for Changelogs, HSM, quota ([LU-19296](#), [LU-19276 WC](#), TLC)

2.18 **lfs migrate** migrate/replace a single object in a file to new OST ([LU-9961 WC](#))

2.19 **lfs getstripe --printf** for printing specific attributes ([LU-20645 Chell!](#))

lctl *_param --module for /sys/module/*/parameters ([LU-14144 WC](#))

lfs migrate --skip-rebalance to balances OST usage ([LU-19052 TLC](#))

Server and Storage Improvements

Improved robustness and ease of management



RHEL/Rocky 10.x servers, Ubuntu 26.04 servers

`io_latency_stats` per target by RPC size ([LU-18934 WC](#))

`ldiskfs` "-o fstrim" mount option for smart continuous TRIM ([LU-14712 WC](#))

`expected_clients` hint to estimate tunable parameters ([LU-17514 WC](#))

Submit `ldiskfs` `dirdata` patches to upstream ext4 ([LU-17423 TLC](#))

New NRS mClock policy to avoid "noisy neighbor" ([LU-17296 HPE, WC](#))

Parallel `e2fsck` pass2 directory entry scan ([LU-14679 WC](#))

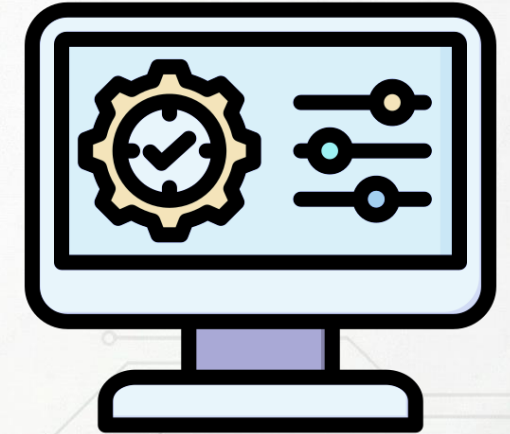
`mdraid-kmec` Erasure Coded declustered parity high-performance SDS (TLC)

2.18 File Join layout for zero-copy S3 multi-part upload merge ([LU-18461 HPE](#))

2.19 Enable default PFL layout on newly-formatted filesystems ([LU-11918 TLC](#))

Delayed block allocation for writeback cache in `ldiskfs` ([LU-12916 WC](#))

LNet and Configuration Improvements



Filter server NIDs for NUMA, Nodemaps ([LU-20290](#) TLC, WC)

EFA optimized LND for Amazon FSX ([LU-18808](#) AWS)

Finish IPv6 NID support ([LU-18417](#) ORNL, HPE, WC, TLC)

TCP zero-copy send enablement ([LU-19763](#) Google)

LND default binding to NUMA domains ([LU-20409](#) Google)

2.18 Peer-to-Peer RDMA (ala AMD GPU Direct) ([LU-20438](#) Google)

2.19 Improve data transfer for sparse file reads ([LU-16897](#) WC)

lnetdump - network agnostic eBPF packet analyzer for LNet ([LU-19410](#) TLC, AWS)

- Generate packet dumps for any LNet network type (TCP / IB / EFA / Loopback / KFI)
- Performant enough for use in production, enable real time monitoring and alerting

API-Based Management and Configuration (2.18+ TLC)

API-driven configuration, management, monitoring

Efficient NetLink interface to Lustre statistics ([LU-19768](#) TLC)

- No more opening, scraping, parsing hundreds of stats files per second

Improve Lustre Library (`llapi_*`()) to cover existing functionality (LU-*many*)

- Move core logic out of `lctl`, `lfs`, configuration scripts into shared library code

Rust-based interface (`rustreapi`) for robust development ([LU-19561](#) TLC, WC)

2.18 [man4](#) pages to document tunable parameters (ongoing, LU-*many*)

2.19 Simplified high-level commands/API for configuration management, ease of use

- **add OST** – format block device, apply parameters, safely add into filesystem
- **remove OST** – migrate all objects off OST, remove from configuration
- **rebalance OSTs** – migrate files/objects across OSTs after new OSTs are added
- **tenant add** – add virtual LNet+NIDs, nodemap ID range, subdirectory, LQA, SSK keys, `fscrypt`

Lustre Metadata Redundancy

(LMR 2.19+)

Metadata mirroring across MDT devices

LMR-FTM: Fault Tolerant MGS ([LU-19116](#), [LU-19117](#) WC)

LMR-MDT0: MDT0000 service redundancy ([LU-18823](#))

- Mirror FLDB, Quota, flock() to other MDTs

LMR-FID: Dir entries with multiple FIDs ([LU-16742](#) TLC)

- Infrastructure for inodes mirrored to multiple MDTs
- Replicas visible from client, directly access if MDT offline

LMR2: Replicate directories for availability ([LU-17820](#))

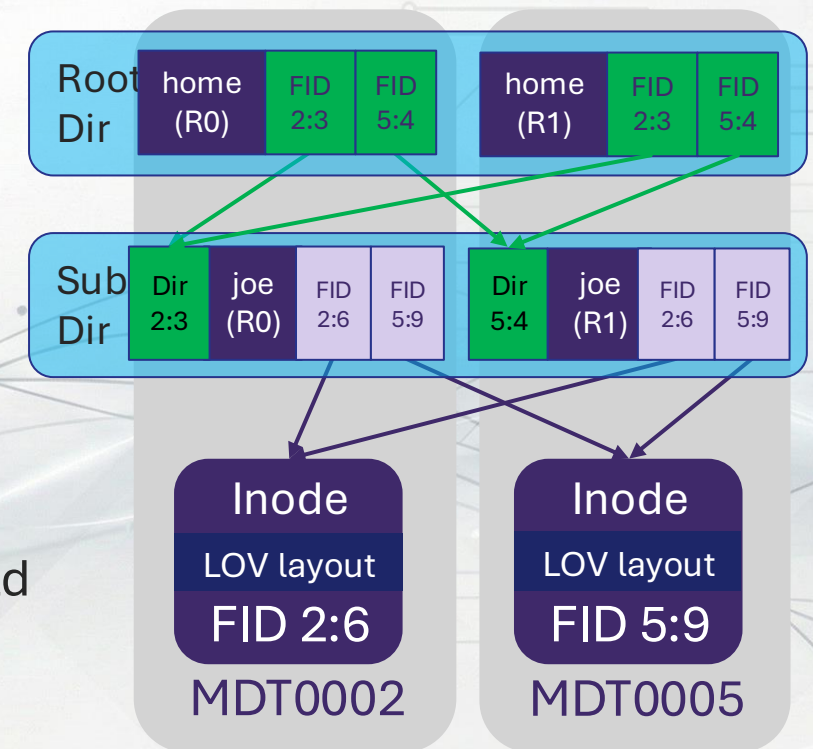
- ROOT: Mirror ROOT/ directory (rarely changed) to other MDTs
- DIR: Subdirectory mirror to 2+ MDTs (setdirstripe)
- FILE: Per-file metadata replication for regular files

LMR3: Complex recovery and write handling

- RECOV: Write to directory while mirror offline, full MDT rebuild

LMR4: LFSCCK to repair/rebuild inconsistent mirrors

- LFSCCK: reconcile inconsistent mirrors, rebuild offline MDTs



MDT0000 replacement

(LMR1b/2a 2.19+)

Remove service dependency on MDT0000 ([LU-18823](#))

- FLDB (FID->MDT/OST map) redundancy ([LU-15414](#))
- Nodemap config redundancy
- Quota ID limit settings redundancy (usage can be rebuilt)

Quota Manager distributed to multiple servers

- Hash distribution by UID/GID/PROJID, easily parallel operations

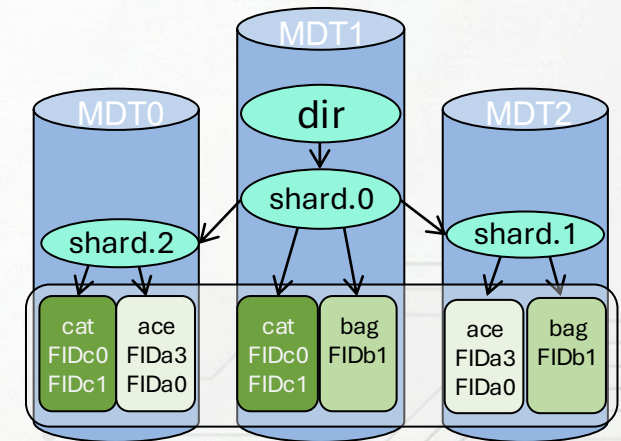
LDLM flock() distribution and scaling over multiple MDS nodes

- flock() locking is not really related to MDT inodes, so can run anywhere
- Need to handle deadlock detection, but is disjoint for process groups, nodemaps

Mirror ROOT/ directory to multiple MDTs ([LU-17820](#))

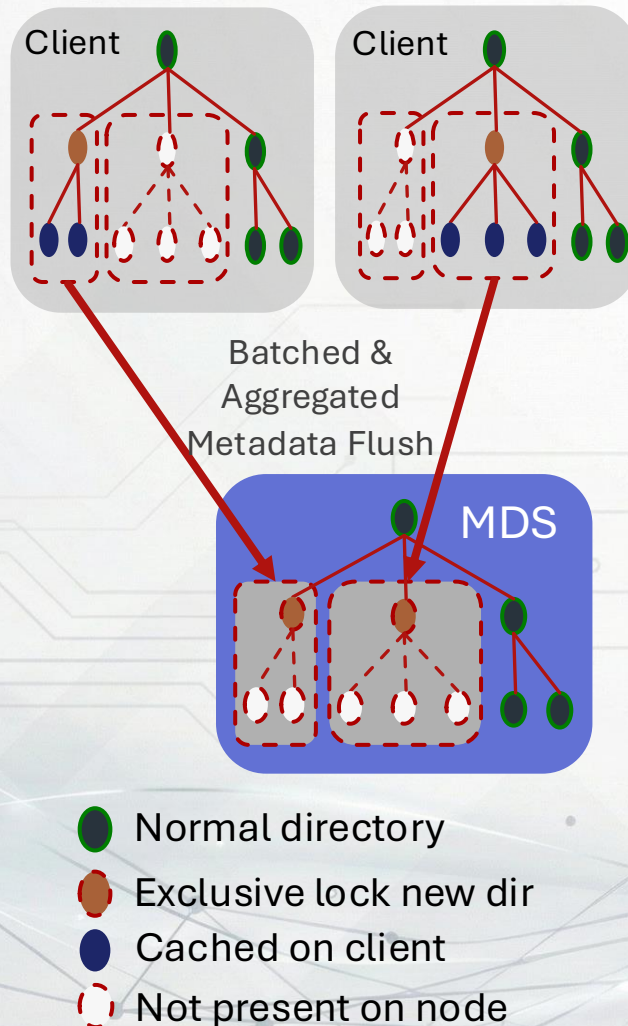
- Rarely changes for most uses, some update overhead is acceptable

Enough to allow replacing MDT0000 on live system



Metadata Writeback Cache

(WBC2 2.19+)



10-100x speedup for single-client create workloads

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

Create new dirs/files in client RAM without RPCs

- Lock new directory exclusively at mkdir time
- Cache files/dirs in RAM until flush or remote access

No RPCs for metadata changes in new dir

Batch RPC for efficient directory read/cache flush

Files globally visible on remote client access

- Flush top dirs, exclusive lock new subdirs, unlock parent
- Flush rest of tree in background by age or size limits

WBC feature already significantly implemented

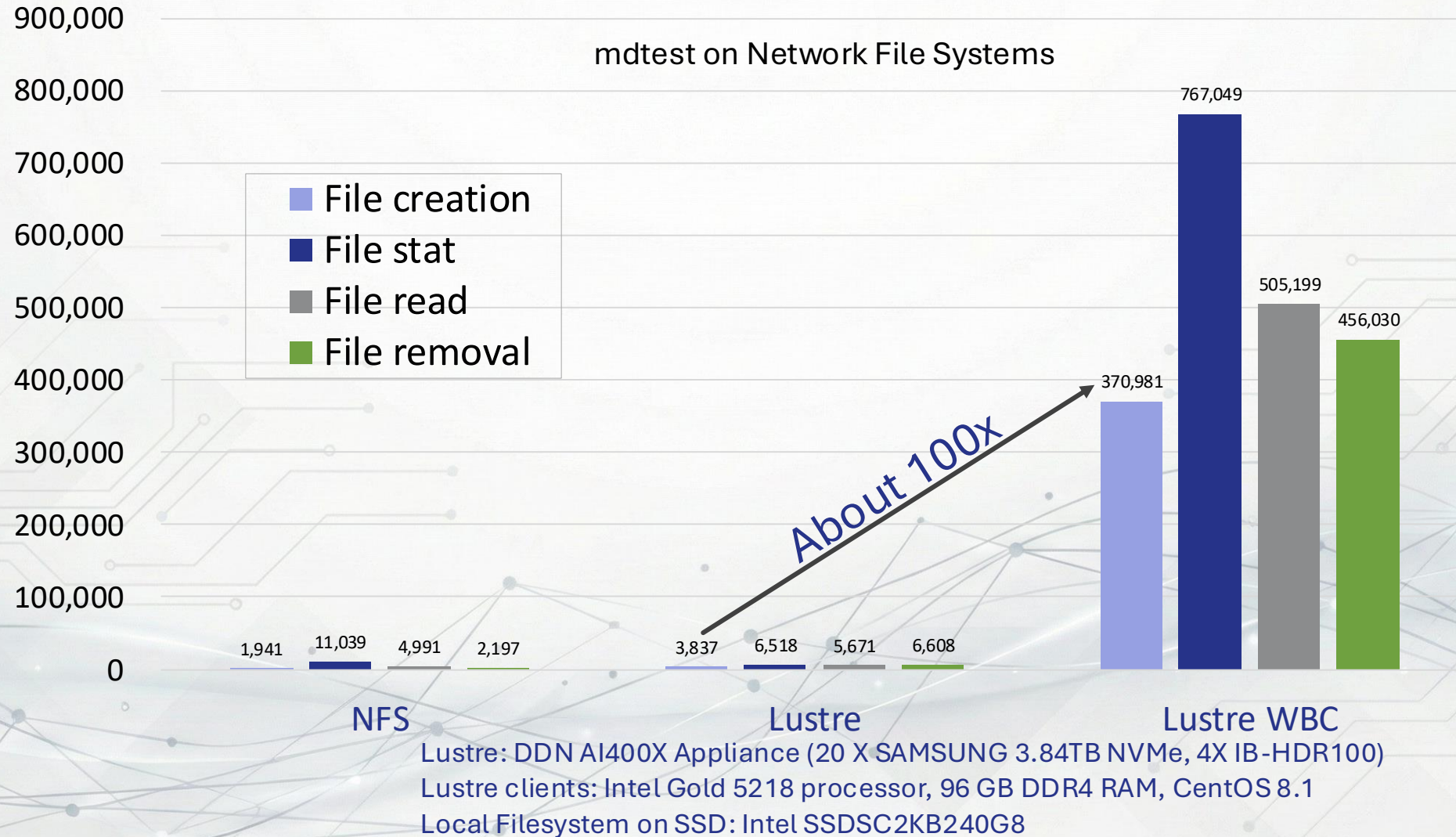
- Some complexity handling partially-cached directories
- Need to integrate space usage with quota/grant

THANK YOU!



Backup Slides

Metadata WBC Performance Gains



Readahead Optimizations

(2.18+)

Readahead is fundamental to achieving high bandwidth for many applications

Readahead currently optimizes multiple access patterns

- Sequential, strided forward readahead ($\text{offset} + N \times \{1, 2, 3, \dots\}$)
- Fuzzy readahead for mmap ($\text{offset} + N\text{-ish} \times \{1, 2, 3, \dots\}$)
- Backward readahead "exists" ($\text{offset} - N \times \{1, 2, 3\}$) but is incorrect in some cases

Separate readahead mode for mmap file access missing some features

Substantial improvements possible, work already in progress

- Improved backward fuzzy readahead to better match real client IO ([LU-15274](#) TLC)
- Opportunistic lockahead for readahead to keep pipeline filled ([LU-15155](#) TLC)
- Merge mmap and standard readahead logic so optimizations usable by all ([LU-15516](#) TLC)

Lustre Continues Evolving for the Future

Still the preferred choice for the world's largest systems

- Majority of Top10 and Top100 HPC systems use Lustre
- World's largest HPC, AI systems (ORNL, LLNL, xAI, NVIDIA, ...) ... or 2RU server for lab with a handful of GPU nodes

Scalability of servers and clients almost without limits

- 200 M+ IOPS, 10 M+ metadata op/sec, 500 B+ files
- Capacity for any need – 10s TB/s read **AND** write, 700 PB+ today, 1 EB+ soon
- Fully support large clients - 100s of cores, TBs of RAM, multi-400Gbps NICs, RDMA
- Flash-optimized, and HDD-native performance and capacity since day one

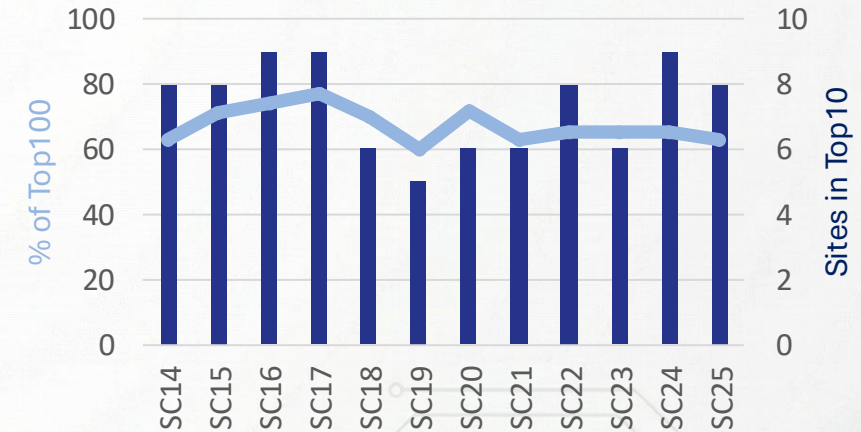
Continued improvements for large system deployments

- Steady feature development to meet evolving system and application needs
- Virtualization of filesystem for user isolation and data privacy

Ongoing improvement for ease-of-use, reliability, and efficiency

- Platform agnostic and fault tolerant on commodity hardware
- Faster failover and less disruption for running applications
- API-driven configuration and management for automation at scale

Lustre in the Top 100



Storage Management Across Entire Cluster

Data locality, with direct client access to all storage tiers

