Lustre® 2.11 and Beyond

Andreas Dilger, Intel High Performance Data Division

SC 2017
Intel & Lustre* Software

Intel continues to invest in Lustre with:

- The latest Major Releases of Lustre
- Next generation Feature Development and enhancements
- Public LTS Maintenance Releases providing ongoing updates
- Expansion of the Worldwide Lustre Support ecosystem
- Ongoing Investment in test, validation, and release processes
- Community Participation in OpenSFS and EOFS

Lines of Code Contributed to Lustre 2.10**

**Data courtesy of Dustin Leverman (ORNL)
Upcoming Feature Highlights

2.11 landings in progress with several features landed or underway
- File DLM lockahead
- Data-on-MDT for improved small file performance/latency
- File Level Redundancy begins (FLR Phase 1 Delayed Resync)

2.12/2.13 plans continued functional and performance improvements
- File Level Redundancy continues (FLR Phase 2 Immediate Resync)
- DNE2 directory auto-stripe to improve usability and performance
- FLR Phase 3 Erasure Coded Striped Files
- Persistent Client Cache
LNet Dynamic Discovery

Builds on LNet Multi-Rail in Lustre 2.10 (Intel, HPE/SGI*)

LNet Dynamic Discovery

• *Automatically* configure peers that share multiple LNet networks
• Avoids need for admin to specify Multi-Rail configuration for nodes

LNet Network Health

• Detect network interface, router faults
• Handle LNet fault w/o Lustre recovery
• Restore connection when available

* Other names and brands may be claimed as the property of others.
Data-on-MDT Small File Perf

Avoid OST overhead (data, lock RPCs)
High-IOPS MDTs (mirrored SSD vs. RAID-6 HDD)
Avoid contention with streaming IO to OSTs
Prefetch file data with metadata
Size on MDT for small files
Integrates with PFL to simplify usage
  - Start file on MDT, grow onto OSTs if larger
Complementary with DNE 2 striped directories
  - Scale small file IOPS with multiple MDTs

Example DoM/PFL File Layout

<table>
<thead>
<tr>
<th>MDT</th>
<th>4 OST stripes</th>
<th>60 OST stripes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 1MB)</td>
<td>[1MB, 1GB)</td>
<td>[1GB, ∞)</td>
</tr>
</tbody>
</table>

Small file IO directly to MDS

https://jira.hpdd.intel.com/browse/LU-3285

Statements regarding future functionality are estimates only and are subject to change without notice.
* Other names and brands may be claimed as the property of others.
Directory migration from single to striped/sharded directories

- Rebalance space usage, improve large directory performance
- Inodes are also migrated along with directory entries

Automatic directory restriping to reduce/avoid need for explicit striping at create

- Start with single-stripe directory for low overhead in common use cases
- Add extra shards when master directory grows large enough (e.g. 32k entries)
- New entries+inodes created in new directory shards on MDTs to distribute load
- Performance scales as directory grows

MDT Pools for space/class management
## ZFS Enhancements Related to Lustre (2.11+)

**Lustre 2.10.1/2.11 osd-zfs updated to use ZFS 0.7.1**

- File create performance (parallel lock/alloc, new APIs) (Intel)
- LFSCK ZFS OI Scrub, Idiskfs->ZFS backup/restore (LU-7585 Intel)

### Features in ZFS 0.7.x

- Dynamic dnode size for better xattr performance/space (LLNL)
- Optimized parallel dnode allocation (Delphix®, LLNL, Intel)
- Improved kernel IO buffers allocation (ABD) (others, Intel)
- Multi-mount protection (MMP) for improved HA safety (LLNL)
- Optimized CPU and QAT h/w checksums, parity (others, Intel)
- Better JBOD/drive handling (LEDs, auto drive resilver) (LLNL)

### Features for ZFS 0.8.x

- On-disk encryption (Datto®)
- Project quota accounting (Intel)
- Declustered RAID (dRAID) (Intel)
- Metadata Allocation Class (Intel)
- Likely lots more…

---

Statements regarding future functionality are estimates only and are subject to change without notice.
* Other names and brands may be claimed as the property of others.
Based on Progressive File Layout (PFL) feature in Lustre 2.10 (Intel, ORNL)

Significant value and functionality added for HPC and other environments

• Optionally set on a per-file/dir basis (e.g. mirror input files and one daily checkpoint)
• Higher availability for server/network failure – finally better than HA failover
• Robustness against data loss/corruption – mirror (and later M+N erasure coding)
• Increased read speed for widely shared input files – N-way mirror over many OSTs
• Mirror/migrate files over multiple storage classes – NVRAM->SSD->HDD (e.g. Burst Buffer)
• Local vs. remote replicas (WAN)
• Partial HSM file restore
• File versioning (no resync replica)
• Many more possibilities ...

Replica 0  Object \( j \) (PRIMARY, PREFERRED)
Replica 1  Object \( k \) (STALE)  \textit{delayed resync}
Upstream Kernel Client (LU-9679 ORNL)

Kernel 4.14 updated to approximately Lustre 2.8, with some fixes from Lustre 2.9
Lustre 2.10 updated to work with kernel ~4.12 (LU-9558)
Improve kernel internal time handling (LU-9019)
  - 64-bit clean to avoid Y2038 issues
  - remove jiffies and cfs_time_*() wrapper functions
Continued user header changes (LU-6401)
  - Allow building user tools against upstream kernel
Kernel tracepoints for logging/debugging/perf analysis (LU-8980)
  - Replace CDEBUG() macros and Lustre kernel debug logs
  - Has potential to improve (or not?) debugging of Lustre problems, needs careful review
Improved client efficiency (2.11/2.12+)

Small file write optimizations (LU-1575, LU-9409 Cray*, Intel)
- Reduce client and RPC/server overhead for small (<= 4KB) reads/writes

Disconnect idle clients from servers (LU-7236 Intel)
- Reduce memory usage on client and server for large systems
- Reduce network pings and recovery times
- Aggregate statfs() RPCs on the MDS (LU-10018)

Reduce wakeups and background tasks on idle clients (LU-9660 Intel)
- Synchronize wakeups between threads/clients (per jobid?) to minimize jitter
- Still need to avoid DOS of server if all clients ping/reconnect at same time
Tiered Storage and File Level Redundancy

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

- **Metadata Servers (~100’s)**
  - Management Target (MGT)
  - Metadata Targets (MDTs)

- **Object Storage Targets (OSTs) (Warm Tier SAS)**
  - Object Storage Servers (~1000’s)

- **Archive OSTs/Tape (Cold Tier Erasure Code)**
  - Archive OSTs/Tape

- **Policy Engine**
- **NVMe MDTs client network**
- **Lustre Clients (~50,000+)**
- **NVMe Burst Buffer/Hot Tier OSTs on client network**

Statements regarding future functionality are estimates only and are subject to change without notice.

* Other names and brands may be claimed as the property of others.
Tiered storage with Composite Layouts (2.12/2.13)

Policy engine to manage migration over tiers, rebuild replicas, ChangeLogs

- Policies for pathname, user, extension, age, OST pool, mirror copies, ...
- FLR provides mechanisms for safe migration of (potentially in-use) data
- Integration with job scheduler and workflow for prestige/drain/archive

Multiple policy and scanning engines presented at LUG'17

Multiple presentations on tiered storage at LAD'17

- Integrated burst buffers are a natural starting point

This is largely a userspace integration task, with some hooks into Lustre
Client-Side Data Compression
University Hamburg

Piecewise compression
- Compressed in 32KB chunks
- Allows sub-block read/write

Integrated with ZFS data blocks
- Leverage per-block type/size
- Code/disk format changes needed

Avoid de-/re-compressing data

Good performance/space benefits
- Graph courtesy Uni Hamburg

Statements regarding future functionality are estimates only and are subject to change without notice.
* Other names and brands may be claimed as the property of others.
Persistent Client-side Cache (LU-10092 DDN* 2.12)
Leverage fast client-local NVMe or NVRAM device

Mount filesystem image file directly on clients
- Used ad-hoc today - single client RW/shared RO mount
- Integrate handling into Lustre transparently

Automatic client-local mount of filesystem image file
- Image is one object on OST, whole directory tree on client
- **Low overhead**, few Lustre locks, **100k+ IOPS/client**
- Access, migrate, replicate with **large reads/writes to OST**

OSS/MDS/client can export directly for shared use
- Use Data-on-MDT to re-export image to other clients
- HSM migrates *from* client, FLR to mirror to OST

Archive whole filesystem tree to tape when unused