Lustre* 2.9 and Beyond

Andreas Dilger, High Performance Data Division

Overview of Features

Features completed for 2.8

- LFSCK Phase 4 - Performance Improvements (Intel, OpenSFS)
- DNE Phase 2 Striped Directories - Asynchronous Commits (Intel, OpenSFS)
- Client IO Simplification (Intel, OpenSFS)
- Multiple metadata-modifying RPCs (multi-slot `last_rcvd`) (Bull=Atos)
- Kerberos/GSS revival (Bull=Atos, Seagate)

Features starting development for 2.9 and later

- UID/GID mapping (IU)
- ZFS* Enhancements (Intel, LLNL)
- Project quotas (DDN)
- Shared-key/GSS crypto (IU)
- Progressive File Layout Prototype (Intel & ORNL)
- Data on MDT Prototype (Intel)

ZFS Enhancements (Intel/LLNL, 2.9+)

Changes for ZFS OSD (2.9)

- 1MB+ ZFS blocksize (IO performance, LLNL)
- Read IO optimization (IO performance, Intel)
- ZIL support for fast sync (IO & metadata performance, Intel)

Changes to core ZFS code (2.9+)

- Inode quota accounting (base functionality, Intel)
- Large dnodes (metadata performance, LLNL)
- Parity declustering (reliability & availability, Intel)
- Distributed hot spares (reliability & availability, Intel)

Miscellaneous features

Code cleanups (Cray*/Intel®/ORNL)

- Update to match upstream kernel coding style
- Port patches to/from upstream kernel
- Clean up and/or eliminate server kernel/ldiskfs patches

Project Quotas (DDN*)

- Allow quota tracking on directory subtrees independent of UID/GID
- Not strictly hierarchical, can be multiple trees with the same project

Network Authentication and Encryption (Bull*/IU*/Seagate*)

- Kerberos user/node authentication, RPC encryption
- Shared Secret Key node authentication, RPC encryption

Data on MDT (Intel, 2.10+)

Efficiently store small files on the MDT(s)

- Avoid OST BRW RPC + disk seek + OST lock for each file access
- Use small-file optimized MDT storage (RAID-10/SSD/NVRAM)
- Avoid RAID-5/6 read-modify-write for small writes

Space usage on MDT(s) managed by quota

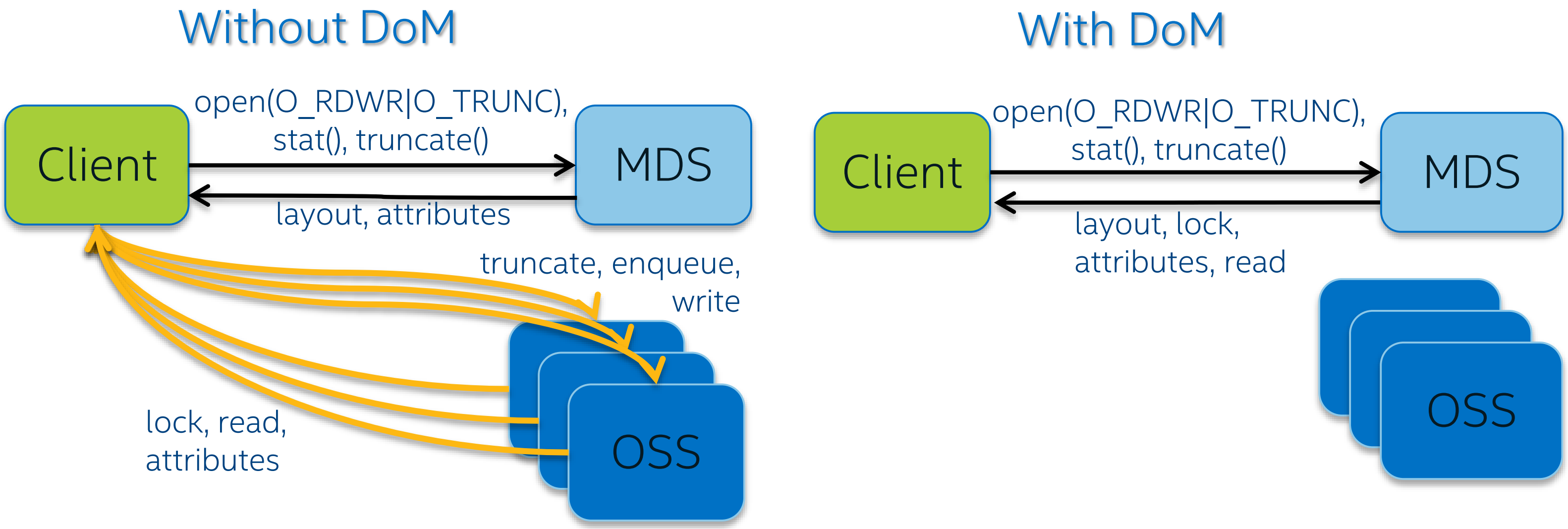
Small files are determined by the file layout

- Maximum MDT file size can be specified by **min(**user, admin**)**
- Typically expected to be <= 1MB, dependent on MDT space

Complementary with DNE 2 striped directories

- Scale small file IOPS horizontally with multiple MDTs

Data on MDT Implementation (Intel, 2.10+)


Without DoM
Client
open(O_RDWR|O_TRUNC), stat(), truncate()
MDS
layout, attributes
truncate, enqueue, write
lock, read, attributes
OSS
With DoM
Client
open(O_RDWR|O_TRUNC), stat(), truncate()
MDS
layout, lock, attributes, read
OSS

DoM layout chosen at file creation time like files on OSTs

- Can't do it after write because objects are allocated at **open()**
- Default DoM striping on subdirectories inherited by newly created files

http://cdn.opensfs.org/wp-content/uploads/2014/04/D1_S10_LustreFeatureDetails_Pershin.pdf

http://wiki.opensfs.org/images/b/be/DataonMDSDesign_HighLevelDesign.pdf

Composite Layouts (Intel, 2.10)

Add *Composite Layouts* for regular files

- Allow describing more complex file structures and interactions
- A composite layout contains multiple *components* (`LOV_MAGIC_V[13]`)
- Composite layouts do not restrict components themselves
- Specific features may impose their own restrictions

```
struct lov_comp_md_v1 {
        __u32 lcm_magic;         /* LCM_MAGIC_V1 */
        __u32 lcm_size;          /* overall size including this struct */
        __u32 lcm_layout_gen;    /* incremented each time layout changes */
        __u16 lcm_flags;         /* LCM_FL_RS_READ_ONLY, LCM_FL_RS_SYNC_PENDING, ... */
        __u16 lcm_entry_count;   /* number of components in lcm_entries[] */
        __u64 lcm_padding[2];
        struct lov_comp_md_entry_v1 lcm_entries[];
};
```

Composite Layout Components

- A *Component* describes one extent of a composite file
- Each component is a separate *plain* layout within a file
  - Currently `LOV_MAGIC_V[13]` (RAID-0) layouts are handled
  - Other layout patterns can be added in the future (`LOV_MAGIC_DOM`, …)
- Components cannot be nested
- Objects are not shared between components

```
struct lov_comp_md_entry_v1 {
    __u32 lcme_id;                  /* unique identifier of component within composite */
    __u32 lcme_flags;               /* LCME_FL_STALE, LCME_FL_PRIMARY, LCME_FL_PREFERRED */
    struct lu_extent lcme_extent;   /* file logical extent for component */
    __u32 lcme_offset;              /* offset of component layout from start of composite */
    __u32 lcme_size;                /* size of component layout data in bytes */
    __u64 lcme_padding;
};
```

What can be done with Composite Layouts?

Progressive File Layouts

- Non-overlapping component layouts for different parts of the file
- Increasing stripe count as file grows larger is expected, but not required

File Level Replication

- Overlapping component layouts provide redundancy
- Replica components can be marked stale or offline if OST failure is detected
- Resync stale components when OST online or add new replicas for failed OSTs

File versioning

- Replica components that are not updated by later writes or resync'd
- Old versions could be accessed via `lfs` or via `ioctl()` on open file descriptor

HSMv2 partial file restore

- One component for each archive copy, along with a timestamp/version for age
- Regular file component(s) for online data, may not cover whole file

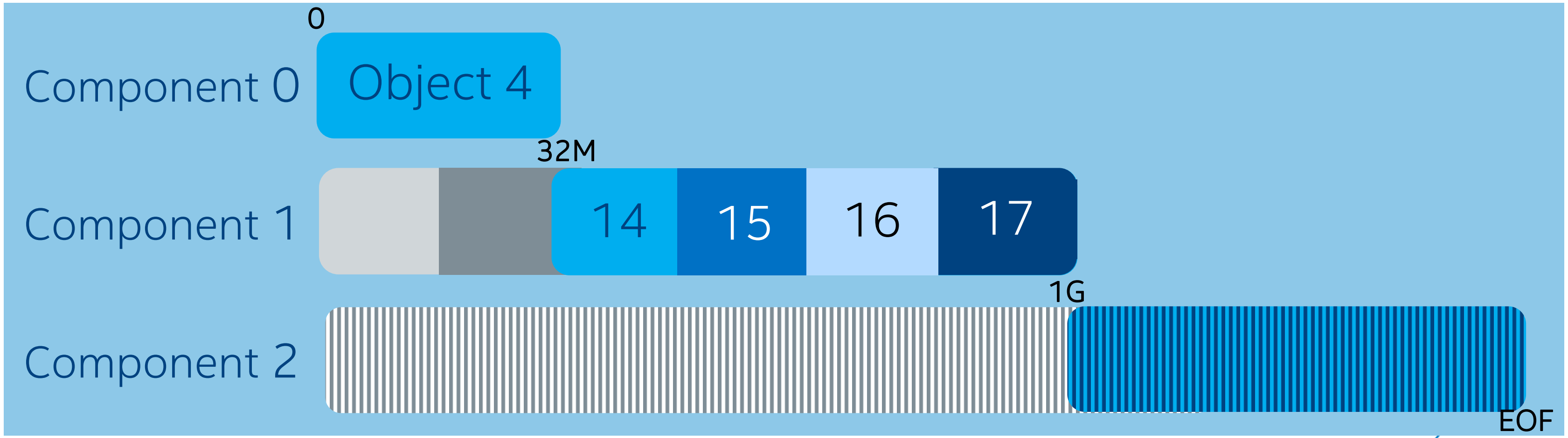
Progressive File Layouts (Intel/ORNL, 2.10)

Allow stripe count to increase for larger files

- Improve aggregate IO bandwidth for large files
- Do not add overhead for small files
- Start with one stripe, add stripes incrementally as file size increases
- Balance lower overhead vs. performance and space balance

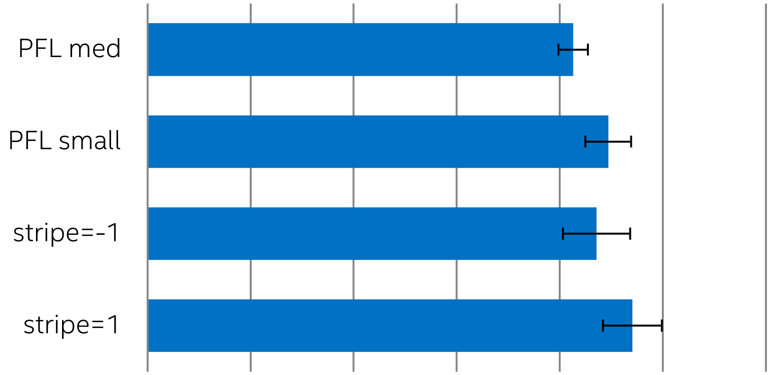
Covered (grey) region of component is inaccessible/sparse

- Allows merging/replication/separation of components from plain files

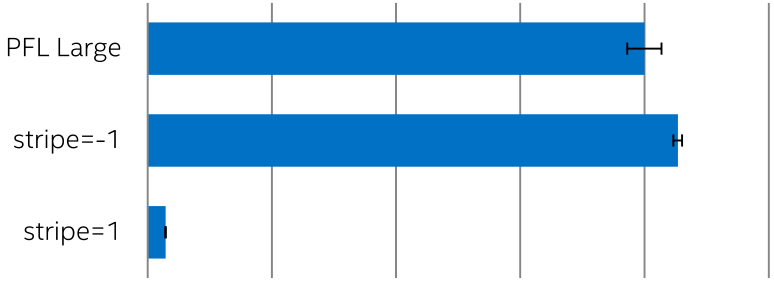


PFL Prototype Performance Comparison

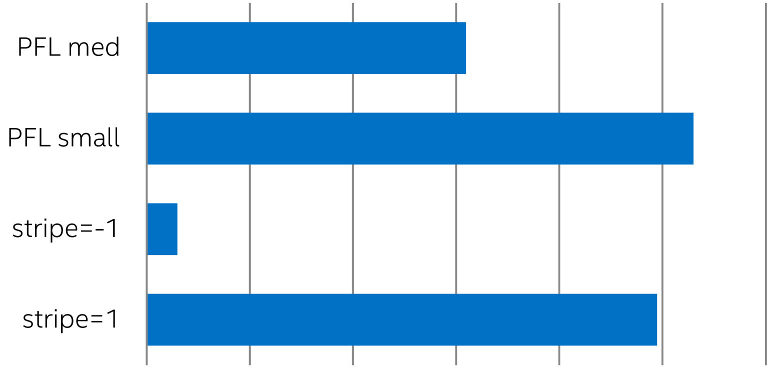
16 threads - Single Client
IOR File per Process Write



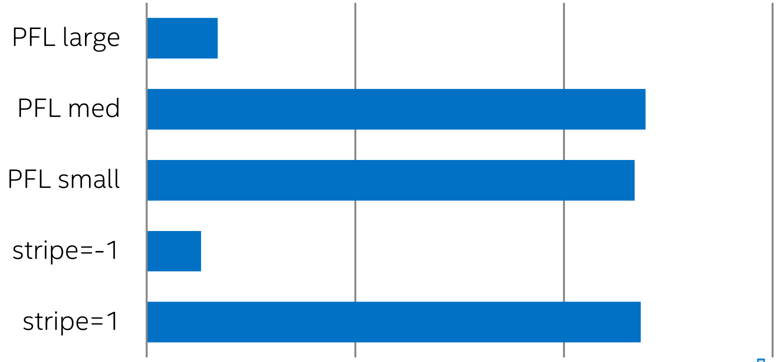
512 Threads - 32 Client
IOR Shared File Write



16 threads - Single Client
mdtest file stat/sec



512 Threads - 32 Client
mdtest file stat/sec



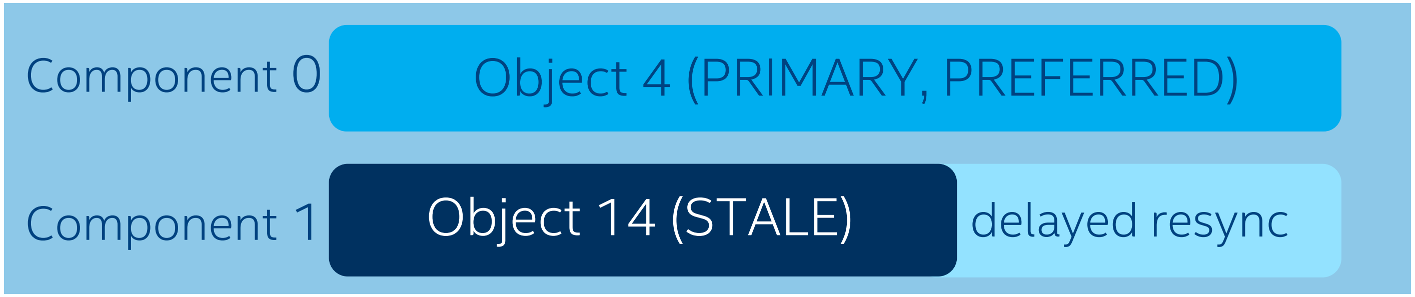
File Level Replication

Allow redundancy at the file level

- Avoid the need for multi-path storage or failover (local server storage OK)
- Redundancy can be selected/added/removed on a per-file basis
- Reads balanced between replicas, recover read errors from replica
- Can tune IO overhead/performance vs. file availability

Phase 1: Delayed replication by external resync tool

- For read-mostly workloads, minimizes write overhead at client
- Only primary replica modified, non-primary replica(s) marked stale on first write
- ChangeLog/copytool drives resync tool after write finished, or if OST is offline in Phase 2



Component 0 Object 4 (PRIMARY, PREFERRED)

Component 1 Object 14 (STALE) delayed resync

Phase 2: Replica updated immediately by client

- Client sends writes to each OST, marks component stale if write fails

Legal Information

- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.
- This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.
- Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.
- The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at http://www.intel.com/content/www/us/en/software/intel-solutions-for-lustre-software.html.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase.
- For more complete information about performance and benchmark results, visit http://www.intel.com/performance.
- Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Software

Backup Slides

LFSCK Phase 4 (Intel/OpenSFS 2.8)

LFSCK performance improvements (Phase 4)

- Improve object iteration, don't load objects unnecessarily
- Avoid a full scrub if only a few objects are found inconsistent
  - Tunable, launch full scrub if more than 60 errors within 60s
- Limit DLM locking to only affected name instead of whole directory
- Predict locking based on recent history
  - LFSCK doesn't lock by default, only lock & reverify on inconsistency
  - If errors recently seen LFSCK locks objects before doing checks
- Improved logging of LFSCK-detected inconsistencies

LFSCK Phase 4 is the final phase of this project

http://wiki.opensfs.org/images/3/3c/LFSCK_Performance_SolutionArchitecture.pdf

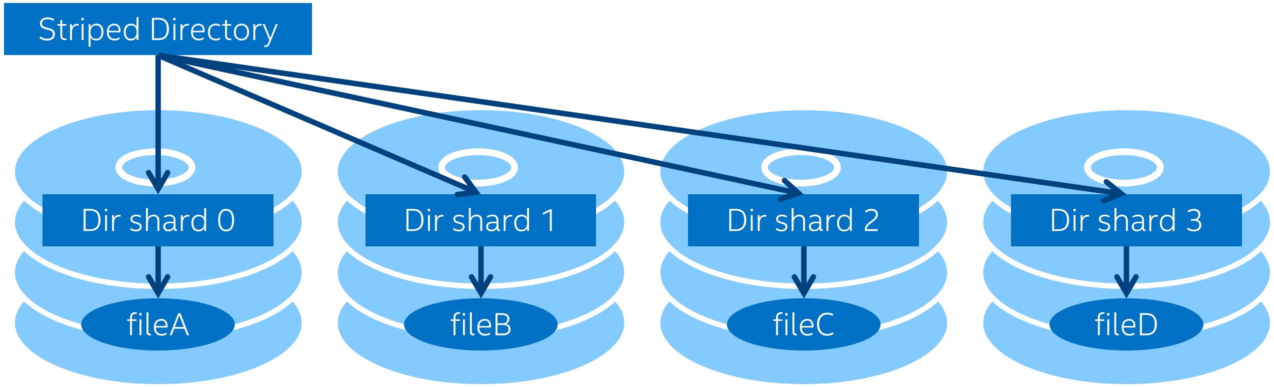
http://cdn.opensfs.org/wp-content/uploads/2013/04/Zhuravlev_LFSCK.pdf

DNE Phase 2 Striped Directories (Intel/OpenSFS 2.8)

Spread a single directory across multiple MDTs

- Reduce contention, improve performance for large directories
- Directory layout + name hash locates slave MDT directory entry
- Directory shard on each MDT independent (lock, lookup, modify)
- Inode created on the same MDT as name entry
- Tool to migrate directories/files from one MDT to another

DNE Phase 2 Async Commits is the final phase of this project



DNE 2 Asynchronous Commit (Intel/OpenSFS 2.8)

Change ***within*** MDT (mkdir, rmdir, rename) **never** synchronous

DNE remote/striped directory create synchronous in 2.4-2.7

- Cross-MDT **rename()** or **link()** weren't working (returned **-EXDEV**)

Async commit implements distributed DNE recovery

- Each target (master/slave) writes a full redo log of all updates
- If *any* target commits a change it can be replayed on *all* involved targets
- Ensures all-or-nothing semantic for namespace-visible changes
- Reduced latency for remote/striped directory creates
- Allow **rename()** and **link()** to work correctly across MDTs
- Foundation for future features (e.g. cross-MDT mirrored objects)

http://wiki.opensfs.org/images/f/ff/DNE_StripedDirectories_HighLevelDesign.pdf

Client IO Cleanup/Speedup (Intel/OpenSFS 2.8+)

Clean up CLIO code and interfaces

- Simplify complex internal locking code
- Replace old ioctl interfaces with proper methods
- Remove non-functional interop code for WinNT and MacOS
  - Remove extra abstraction layer complexity and overhead
  - Remove non-functional liblustre code and abstractions
- Remove access to LOV layout internals throughout code
  - Preparation for handling of more complex file layouts (e.g. PFL)

Client Performance Improvements

- Larger RPC sizes for improved allocation and disk IO
- Single-threaded IO performance improvements

http://wiki.opensfs.org/images/b/b7/CLIOSimplificationDesign_HighLevelDesign.pdf

Client Metadata RPC Scaling (Bull/Intel 2.8)

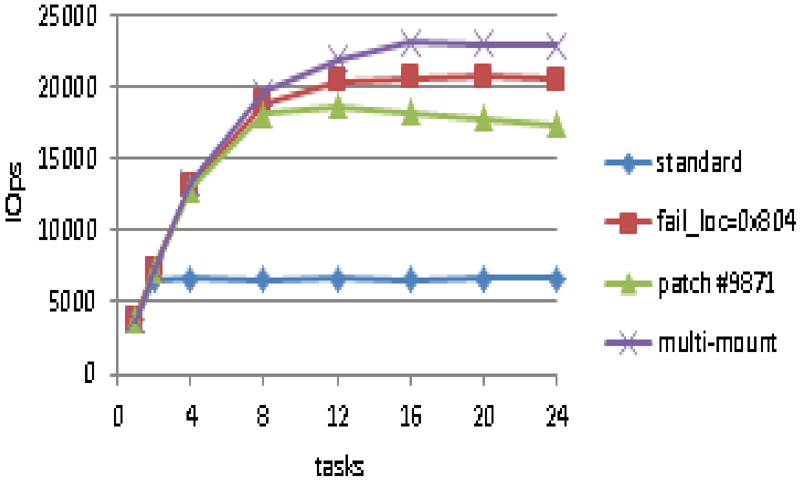
(aka multi-slot **last_rcvd**)

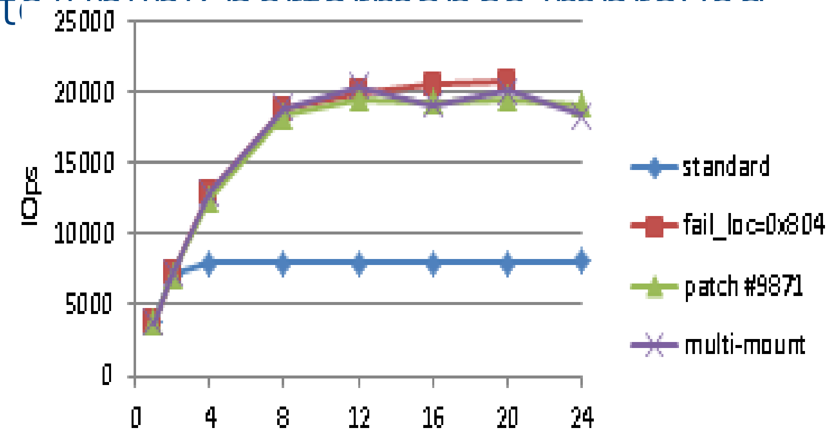
Currently limited to one modifying RPC (+close) per client

- **last_rcvd** slot on MDT for each client to reconstruct reply
- Many concurrent clients limited by MDS performance

Dynamic log on MDT for multiple saved RPC replies per client

- Each metadata-modifying RPC has a separate tag/index
- Single client multi-threaded create/unlink performance improved





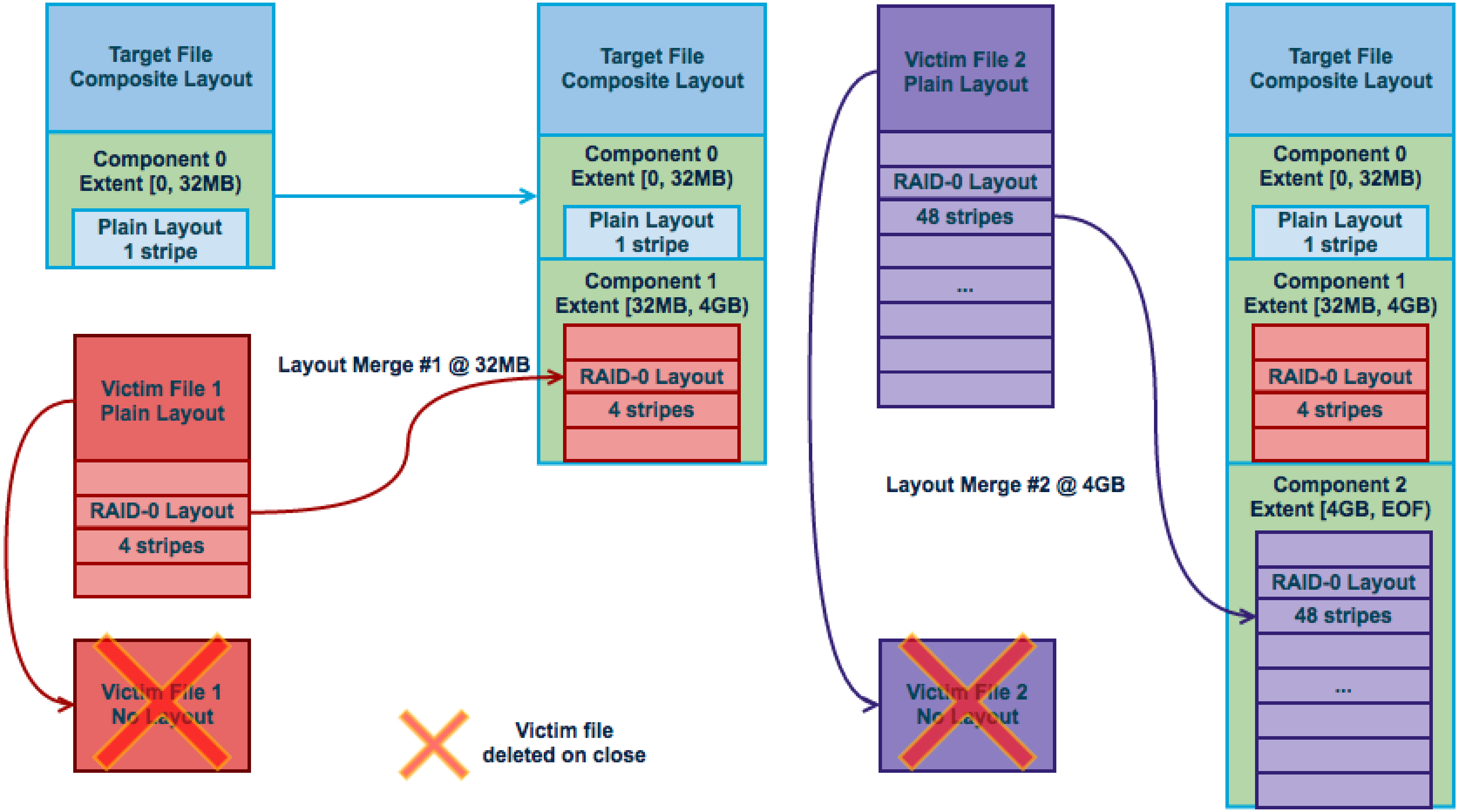
https://jira.hpdd.intel.com/browse/LU-5319

Intel® Omni-Path Architecture Gen 1 (Intel 2.8)

LNet support for Intel® Omni-Path host fabric interface (HFI)

- Next generation interconnect from Intel
- Compatible with OFED verbs interface
- Lustre automatically sets LNet o2iblnd tuning for improved performance

Creating Progressive File Layouts


Target File
Composite Layout
Component 0
Extent [0, 32MB)
Plain Layout
1 stripe
Victim File 1
Plain Layout
RAID-0 Layout
4 stripes
Victim File 1
No Layout
Layout Merge #1 @ 32MB
Target File
Composite Layout
Component 0
Extent [0, 32MB)
Plain Layout
1 stripe
Component 1
Extent [32MB, 4GB)
RAID-0 Layout
4 stripes
Victim file
deleted on close
Victim File 2
Plain Layout
RAID-0 Layout
48 stripes
...
Layout Merge #2 @ 4GB
Victim File 2
No Layout
Target File
Composite Layout
Component 0
Extent [0, 32MB)
Plain Layout
1 stripe
Component 1
Extent [32MB, 4GB)
RAID-0 Layout
4 stripes
Component 2
Extent [4GB, EOF)
RAID-0 Layout
48 stripes
...