



**Whamcloud**

# **Lustre 2.14 and Beyond**

Andreas Dilger, Whamcloud

# Upcoming Release Feature Highlights

## ▶ **2.13** feature complete, ETA November, 2019

- Persistent Client Cache (PCC) – store file data in client-local NVMe/NVRAM
- LNet Multi-Rail Routing – extend MR to/through routers, handle mixed interfaces
- DNE space balanced remote directory – improve load/space balance across MDTs
- Layout OST Overstriping – allow multiple objects from one OST in a striped file
- Self-Extending Layouts (SEL) – better handle OST out-of-space in the middle of a file

## ▶ **2.14** has a number of features under active development

- DNE directory auto-split – improve usability and performance with multiple MDTs
- File Level Redundancy - Erasure Coding (EC) – efficiently store striped file redundancy
- OST Pool Quotas – manage space on tiered storage targets using OST pools

## ▶ **2.15** plans continued functional and performance improvements

- Metadata Writeback Cache (WBC) – low latency file operations in client RAM
- Client-side data encryption – persistent encryption from client to disk

# Persistent Client Cache (PCC) ([LU-10092](#))

(2.13)



- ▶ **Reduce latency**, improve small/unaligned IOPS, reduce network traffic
- ▶ PCC integrates Lustre with a persistent per-client **local cache storage**
  - A local filesystem (e.g. ext4 or 1diskfs) is created on client device (SSD/NVMe/NVRAM)
  - Data is local to client, no global/visible namespace is provided by PCC
  - HSM POSIX copytool fetches whole files into PCC by user command, job script, or policy
  - New files created in PCC are *also* created on Lustre MDS

## ▶ Lustre uses local PCC data, or normal OST RPCs

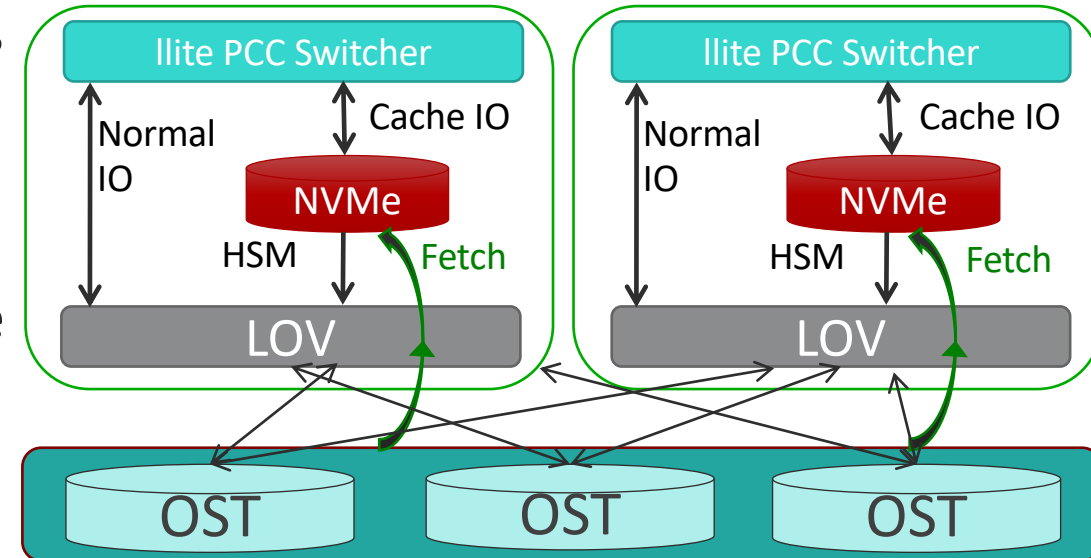
- Further file read/write access “directly” to cache file
- No data/IOPS/attributes off client while file in PCC
- File migrated out of PCC via HSM upon remote access

2.13

2.14

## ▶ Separate **shared read vs. exclusive write** cache

## ▶ Integrate with DAX for NVRAM cache device



# DNE Improvements

(2.13+)



## ▶ **Space balance new directories** on "best" MDT based on available inodes/space

- Simplifies multiple MDTs without overhead of striping all directories, similar to OST balance

2.12

- Explicitly when creating a new directory with "`lfs mkdir -i -1`" ([LU-10277](#))
- Transparently select "best" MDT for normal `mkdir()` based on parent policy ([LU-10784](#), [LU-11213](#))
  - Set default policy on parent via "`lfs setdirstripe -i -1 dir`"
  - Most useful for root directory and top-level user directories

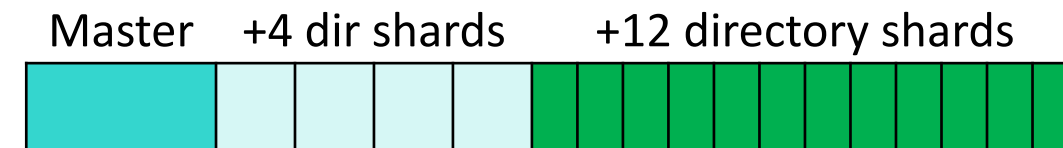
2.13

## ▶ **Improved DNE file create performance** for clients ([LU-11999](#), Uber)

2.14

## ▶ **Automatic directory restriping** as directory size grows ([LU-11025](#))

- Create one-stripe directory for low overhead, scale shards/capacity/performance with size
- Add extra directory shards when master directory grows large enough (e.g. 10k entries)
- Move existing dirents to new directory shards
- New dirents and inodes created on new MDTs



# Data-on-MDT (DoM) Improvements

(2.13+)

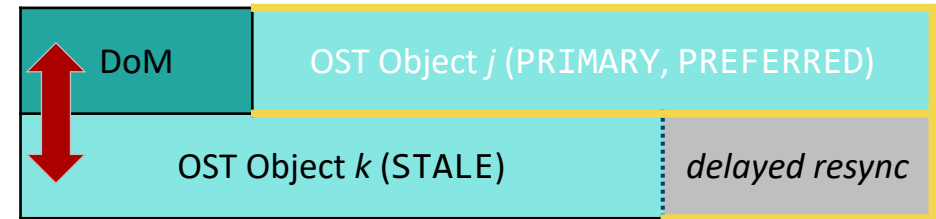


▶ Convert write locks to read locks w/o cache flush ([LU-10175](#))

▶ General usability and stability improvements

▶ FLR mirror/migrate DoM file ([LU-11421](#))

- Mirror DoM data to OST object
- Migrate DoM data to/from OST object
- No MDT-MDT mirroring yet



▶ Performance and functional improvements

- Target IO-500 mdtest-hard-`{write,read}` (3901-byte parallel file create in shared dir)

2.13

2.14 ▶ Dynamic DoM component size by MDT free space ([LU-12785](#))

▶ Merge data write with MDS\_CLOSE RPC ([LU-11428](#))

▶ Cross-file data prefetch via statahead ([LU-10280](#))

▶ Allow MDT-only filesystem ([LU-10995](#))

# LNet Multi-Rail Selection Policy

(2.13+)



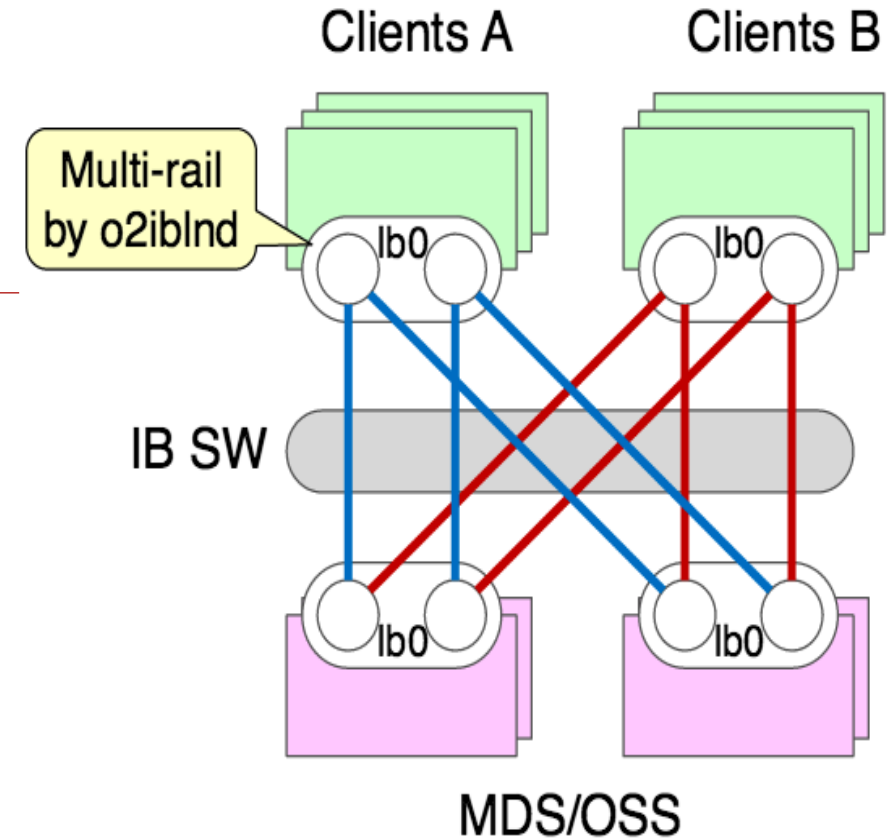
## ► Multi-Rail routing ([LU-11299](#))

- Extend LNet Multi-Rail to router nodes
- Improve handling of mixed MR/single networks

2.13

## 2.14 ► User Defined Selection Policy ([LU-9121](#))

- Fine grained control of interface selection
  - TCP vs. IB networks, primary vs. backup
- Optimize RAM/CPU/PCI data transfers
- Useful for large NUMA machines



# Improved Client Efficiency for AI/ML

(2.13+)



- ▶ **Single thread create performance on DNE** ([LU-11999](#), Uber)
  - Reduce locking overhead/latency for *single-threaded* workloads (780/sec -> 2044/sec)
- ▶ **Parallel client readahead performance** ([LU-8709](#), [LU-12043](#))
  - Improved *single-threaded readahead* (e.g. "dd") from 1.9GB/s -> 4.0GB/s
- ▶ **Overstriping OST objects** better use of large/fast OSTs from fewer clients ([LU-9846](#))
  - "lfs setstripe -C|--overstripe-count *stripe\_count*" for multiple objects per OST
- ▶ **Improved small file handling** (IO-500 mdtest-hard-{write,read} performance)
  - Cache small files after create ([LU-11623](#), [LU-12325](#), [LU-10948](#), ...)
- ▶ **Improved strided read/write** (IO-500 ior-hard-{write,read} performance)
  - Detect and handle page-unaligned strided reads ([LU-12644](#))
- ▶ **Kernel lockahead for strided writes** ([LU-12550](#))
  - Allow readahead to continue for slightly "imprecise" strides
- ▶ **Local client mount on OST/MDT** for data mover/resync ([LU-10191](#))
  - Beginning of optimization for local IO path to avoid RPC + data copy

2.13

2.14

# Performance Improvements for Flash

(2.12+)



## ▶ Reduce server CPU overhead to improve small flash IOPS ([LU-11164](#))

- Reduced CPU usage translates directly to improved IOPS

## ▶ Avoid page cache on ldiskfs flash OSS ([LU-11347](#))

- Avoids CPU/lock overhead/lock for page eviction

## ▶ TRIM flash storage on ldiskfs ([LU-11355](#))

- Release unused blocks of filesystem via `fstrim`

2.12

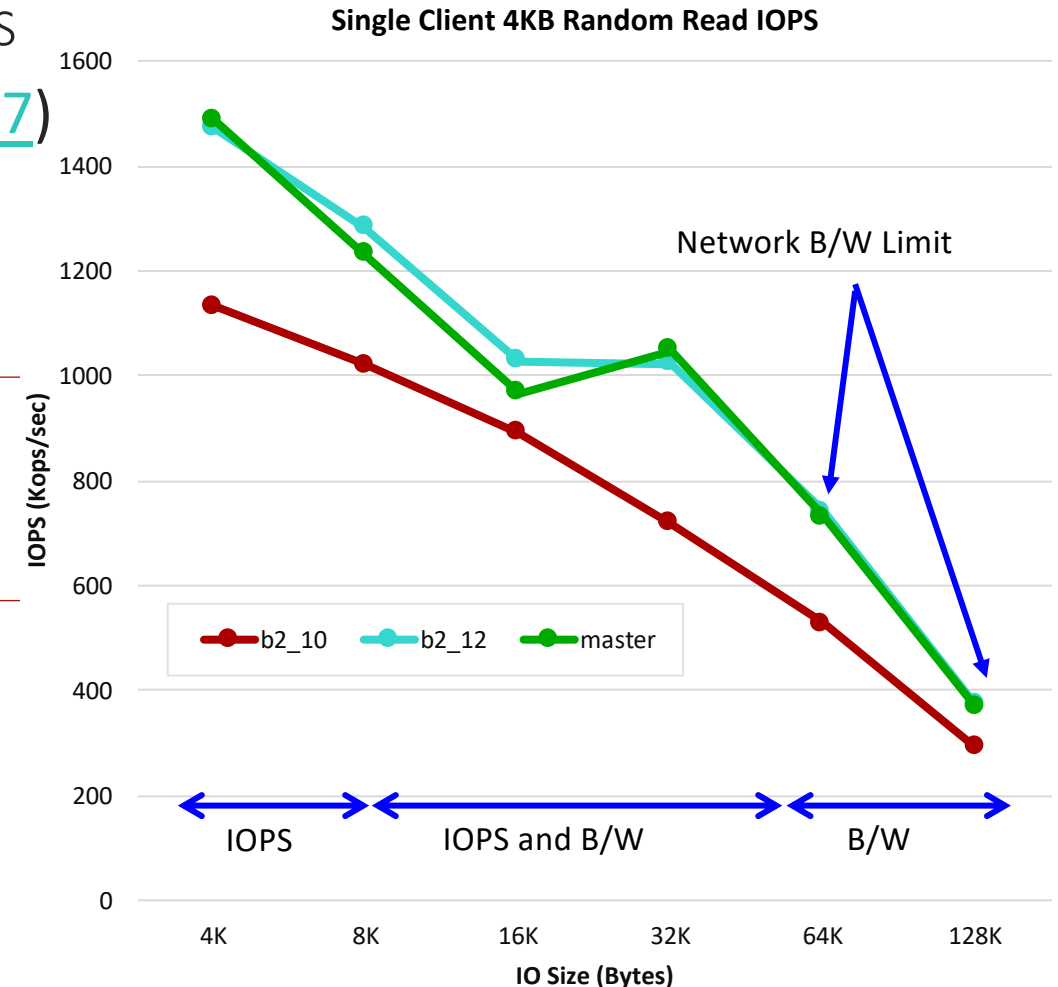
## ▶ Self Extending Layouts ([LU-10070](#), Cray)

- Avoids out-of-space in the middle of files
- Good for PFL with smaller flash OSTs than disk OSTs

2.13

## 2.14 ▶ Continued reductions of overhead and latency

- Improve small, unaligned and interleaved writes
- Improve IO-500 `ior-hard-write`





# Ongoing Idiskfs Improvements

(2.13+)



## ▶ Major `ldiskfs` features merged into upstream `ext4/e2fsprogs`

- Large xattrs (up to 64KB/xattr) stored in separate inode (**`ea_inode`**)
- Large directories over 10M entries/2GB (**`large_dir`**)
- Project quota accounting/enforcement (**`project`**)

2.13

---

## 2.14 ▶ One more Lustre-specific feature remains to be merged to `ext4/e2fsprogs`

- Extended data in directory (**`dirdata`**) - needs unit test interface before merge

## ▶ Existing `ext4` features available that could be used by Lustre on `ldiskfs`

- Efficient block allocation for large OSTs (**`bigalloc`**)
- Tiny files (1-600/3800ish bytes) stored directly in the MDT 1KB/4KB inode (**`inline_data`**)

2.15

---

## ▶ New `ext4` features currently under development

- **Data Verity** – Merkle tree of data checksums stored persistently on *read-only* files
- **Directory shrink** – reduce directory block allocation as files deleted

# File Level Redundancy (FLR) Enhancements

(2.13+)



- ▶ **Lustre-level mirroring for files**, configured arbitrarily per file/directory
- ▶ **Mirror NOSYNC flag** + timestamp to allow *file version/snapshot* ([LU-11400](#))
- ▶ Mount client directly on OSS without impacting recovery ([LU-12722](#))
- 2.13 ▶ **"lfs mirror resync/delete --pool"** to simplify tiering ([LU-11022](#))


- 2.14 ▶ **Erasur coding** adds redundancy without 2x/3x mirror overhead ([LU-10911](#))
  - Add erasure coding to new/old striped files *after* write done
  - Leverage CPU-optimized EC code ([Intel ISA-L](#)) for best client performance
  - For striped files - add N parity per M data *stripes* (e.g. 16d+3p)
  - Fixed RAID-4 parity layout *per file*, **declustered** Parity across files to avoid IO bottlenecks

- 2.15 ▶ **HSM in composite layout** ([LU-10606](#))
  - Allow multiple archives per file (S3, tape, ...)
  - Allow partial file restore from archive

- TBD ▶ **File version/reflink within namespace?**
  - Access like VAX/VMS using "filename,1"?

Replica 0	Flash Object <i>j</i> (PRIMARY, PREFERRED)	
Replica 1	HDD Object <i>k</i> (STALE)	<i>delayed resync</i>
Replica 2	HSM S3 Archive	

# Miscellaneous Improvements

(2.13/2.14)   
Whamcloud

▶ **Overstriping** allows multiple file stripes per OST ([LU-9846](#), Cray/WC)

- Useful for shared-file workloads or very large OSTs

▶ **lfs find** integration with Lazy Size-on-MDT ([LU-11367](#))

2.13 ▶ **Upstream kernel client cleanups** still under active development/merge (ORNL/SuSE)

---

2.14 ▶ **Pool Selection Policy** by filename extension, NID, UID/GID ([LU-11234](#))

▶ **Dynamic OSS page cache** based on RPC IO size ([LU-12071](#))

▶ **fcntlallocate()** for file preallocation (ldiskfs only), hole punch ([LU-3606](#))

▶ **statx()** for lightweight attribute fetching ([LU-10934](#))

▶ **O\_TMPFILE** for creating temporary files outside namespace ([LU-9512](#))

# Pool Quotas for OSTs ([LU-11023](#), Cray) (2.14+)

(2.14+)



## ▶ Account/limit space for OSTs in a specific pool

- Control usage of small flash OSTs in tiered config

## ▶ Use existing Lustre quota infrastructure

- OST already tracks space per UID/GID/ProjID
- Pool usage based on sum of current OSTs in pool

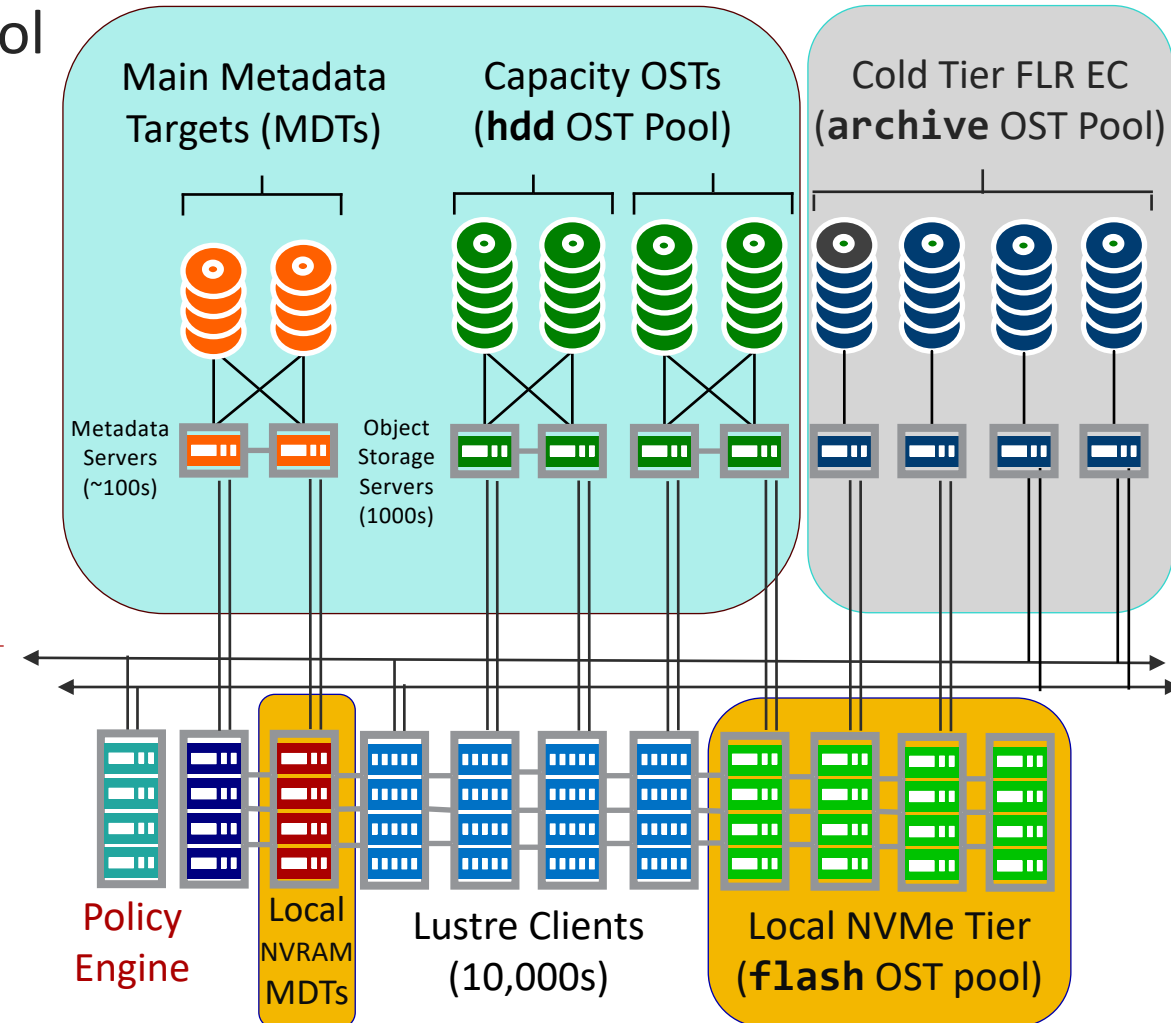
## ▶ Add pool quota limits per UID/GID/ProjID

- No extra accounting on the OSTs
- Only new aggregation/reporting by MDS

2.14

## TBD ▶ Add MDT pools after OST pools complete

- Manage DoM space usage
- Allow different MDT storage classes (e.g. NVRAM)



# Client-Side Data Encryption at Rest ([LU-12755](#)) (2.15)



▶ Protect from storage theft/mistakes network/admin snooping

▶ **Encryption on Lustre client** down to storage

- Applications see clear text in client cache
- Data is **encrypted before sending** to servers
- Data is **decrypted after receiving** from servers
- Servers/storage only see encrypted data/filenames
- Only client nodes need access to user encryption keys
- **Transparent to backend** filesystem/storage (ldiskfs/ZFS)
- Utilize larger client CPU/accelerator capacity

▶ **Ext4/f2fs fsencrypt library/tools base** (don't invent it!)

- Tunable encryption setting/key(s) per directory tree
- **Per-file encryption key(s)**, itself encrypted by user key
  - Fast and secure deletion of file once per-file key is erased
- Filenames encrypted in MDT directory entries



# Metadata Writeback Cache (WBC) ([LU-10983](#))

(2.15+)



## ▶ Create new dirs/files **without RPCs in client RAM** (or local NVMe)

- Lock new directory exclusively at mkdir time
- Cache new files/dirs/data only in RAM/local NVMe until cache flush

## ▶ **No RPC round-trips** for file modifications in new directory

## ▶ **Files globally visible on flush to MDS**, *normal usage afterward*

- Flush top directory to MDS upon other client access, lock conflict
  - Create top-level entries, exclusively lock new subdirs, release parent
  - Repeat as needed for portion of namespace being accessed remotely
- Flush rest of tree in background to MDS/OSS by age or size limits

## ▶ **Basic WBC prototype developed to test concept**

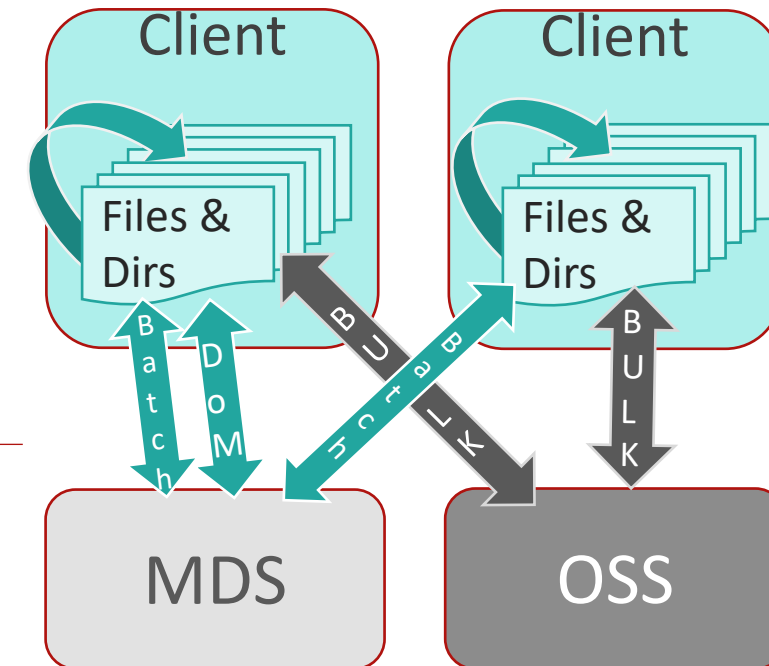
- No cache/quota/space limits, no background flushing, no batching, ...
- 10-20x *single-client* speedup in early testing (untar, make, ...)

2.15

2.16

## ▶ **Aggregate operations to server** to improve performance

- Batch operations in one RCP to reduce network traffic/handling
- Batch operations to filesystem to reduce disk IOPS





***Whamcloud***