Lustre 2.16 and Beyond

Andreas Dilger

Lai Siyao

Trends in High Performance Storage

- Ever increasing demand for high-performance storage to feed data pipelines
- AI/ML/ChatGPT/LLM driving surge in new users of parallel (flash) storage
- Existing computation models (weather, finance, …) increasing resolution, data sources, historical data
- Initial demands can be met by all-flash storage, but not everyone has the budget to scale flash

- Need to increase capacity, reduce costs, transparently access multiple storage types/tiers/hybrid
- Lustre already allows transparent data migration between tiers, hybrid storage *within* files

- Disk and QLC, combined with compression, heading to lowest $/TB and more accessible than tape
- Meta/data redundancy improves availability above hardware, simplifies hardware requirements

- Security, multi-tenancy, data isolation demands always increasing (medical, privacy, IP, legislative, …)

Planned Feature Release Highlights

- **2.16** approaching feature completion
  - **LNet IPv6 addressing** – must-have functionality for future deployments (SuSE, ORNL)
  - **Optimized Directory Traversal** (WBC1) – improve efficiency for accessing many files (WC)
- **2.17** has major features already well underway
  - **Client-side data compression** – reduce network and storage usage, costs (WC, UHamburg)
  - **Metadata Writeback Cache** (WBC2) – order of magnitude better metadata speed (WC)
- **2.18** feature proposals in early stages
  - **File Level Redundancy - Erasure Coding** (FLR-EC) – reduce cost, improve availability (ORNL)
  - **Lustre Metadata Redundancy** (LMR1) – improve availability for large DNE systems
  - **Client Container Image** (CCI) – improved handling of aggregations of many small files

LNet Improvements

Whamcloud

Demand for IPv6 in new deployments as IPv4 is exhausted

- Relatively few external-facing Lustre systems means 10.x.y.z is still viable for now

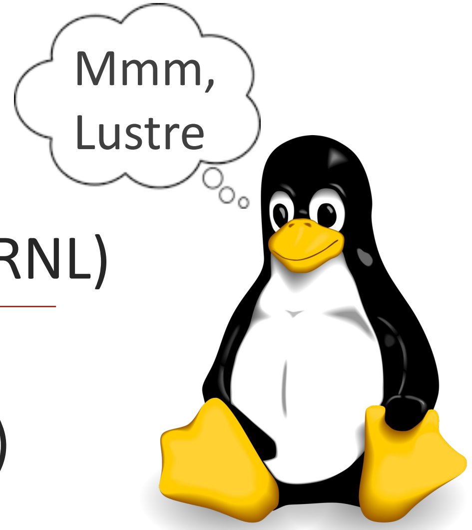
▶ IPv6 large NID support (LU-10391 SuSE, ORNL)
- Variable-sized NIDs (8-bit LND type, 8-bit address size, 16-bit network number, 16-byte+ address)
- Interoperable with existing current LNDs whenever possible
- Enhancements to LNet/`socklnd` for large NIDs mostly finished
- Work ongoing to handle large NIDs in Lustre code
  - Mount, config logs, Imperative Recovery, Nodemaps, root squash, etc.

▶ Improved network discovery/peer health (HPE, WC)

▶ Simplified/dynamic server node addressing (LU-14668 WC)
- Detect added/changed server interfaces automatically (LU-10360)
- Reduce (eventually eliminate) static NIDs in Lustre config logs
- Simplified handling for IPv6 NIDs by clients

Client-Side Usability and Performance Improvements

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



- Parallel file/directory rename within a directory (LU-12125 WC)
- `llstat`, `llobdstat` easier to use (LU-13705 WC)

2.15
- `lfs find -printf` formatted output of specific fields (LU-10378 ORNL)

2.16
- `lfs migrate` performance improvements (LU-16587 HPE, WC)
- `lfs migrate` bandwidth limit, progress updates (LU-13482 Amazon)
- Ongoing code updates/cleanup for newer kernels (ORNL, HPE, SuSE)

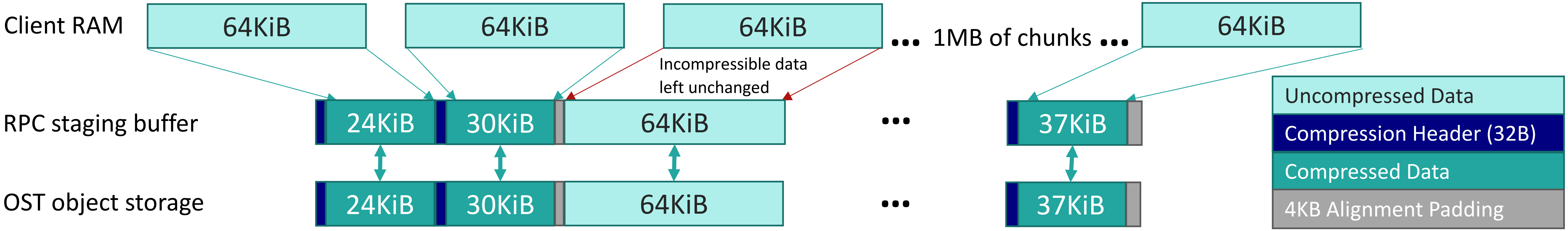
2.17
- Client-side Data Compression (LU-10026 WC, UHamburg, Intel)
- Buffered/DIO performance/efficiency improvements (LU-13805, LU-14950 WC)
- Erasure Coded FLR files (LU-10911 ORNL)

Client-Side Data Compression (WC, UHamburg 2.17+)

Increased capacity and lower cost for all-flash OSTs

- Parallel compression of RPCs on client cores for GB/s speeds, **no server CPU overhead!**

▶ (De-)Compress (`lzo`, `lz4`, `gzip`,…) RPC on client in chunks (64KiB-1MiB+) (LU-10026)
- **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

▶ Optional write uncompressed to one FLR mirror for random IO pattern

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

Server-side Capacity and Efficiency Improvements

Whamcloud

Ongoing performance and capacity scaling for next-gen hardware and systems

- OST object directory scalability for multi-PB OSTs (LU-11912 WC)
  - Regularly create new object subdirectories (every 32M creates vs. 4B creates)
  - Better handling for billions of objects, grouping by age optimizes RAM and IOPS
- Read-only mounting of OST and MDT devices (LU-15873 WC)
- Improved `e2fsck` for large dir and shared block errors (LU-14710, LU-16171 WC)
- `lljobstat` utility for easily monitoring "top" jobs (LU-16228 WC)

2.16
  - Add IO size histograms to job_stats output, handle bad job names better

2.17
- Reduced transaction size for many-striped files/dirs (LU-14918 WC)
- Improved ldiskfs mballoc efficiency for large filesystems (LU-14438 Google, WC, HPE)
- Parallel e2fsck for pass2/3 (directory entries, name linkage) (LU-14679 WC)

Improved Data Security and Containerization

Growing dataset sizes and varied uses increases need to isolate users and their data

- Filenames encrypted on client in directory entries (LU-13717 WC)
- Migrate/mirror of encrypted files without key (LU-14667 WC)
- Encrypted file backup/restore/HSM without key (LU-16374 WC)
- 2.15 Nodemap project quota mapping, squash all files to project (LU-14797 WC)
- 2.16 Read-only mount enforced for nodemap clients (LU-15451 WC)
- Kerberos authentication improvements (LU-16630, LU-16646 WC, NVIDIA)
- Nodemap Role-Based Admin Controls (fscrypt, changelog, chown, quota) (LU-16524 WC)
- Cgroup/memcg memory usage limits for containers/jobs on clients (LU-16671 WC, HPE)

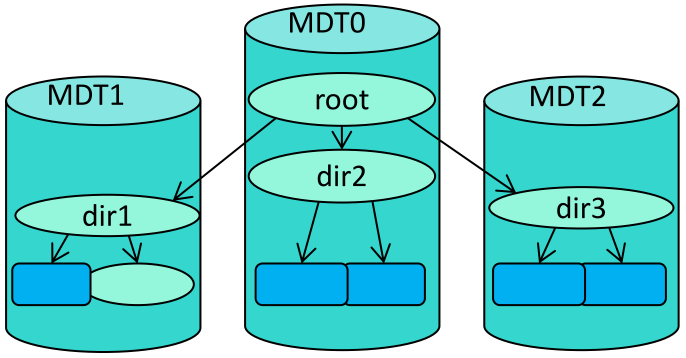
Metadata Scaling Improvements (WC 2.15+)

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

- **DNE MDT Space Balance** - load balancing with normal `mkdir` (LU-13417, LU-13440)
  - Round-robin/balanced subdirs, limited layout inheritance depth, less need for striped dirs
- Single-dir migration without recursion - "`lfs migrate -m -d <dir>`" (LU-14975)

2.15
- Balanced migration prefers less full MDTs - "`lfs migrate -m -1 <dir>`" (LU-13076)

2.16
- DNE inode migration improvements (LU-14719, LU-15720)
  - Pre-check target space, stop on error, improved CRUSH2 hash
- More robust DNE MDT llog recovery (LU-16203, LU-16159)
  - Handle errors and inconsistencies in recovery logs better
- DNE locking, remote RPC optimization (LU-15528)
  - Distributed transaction performance, reduce lock contention



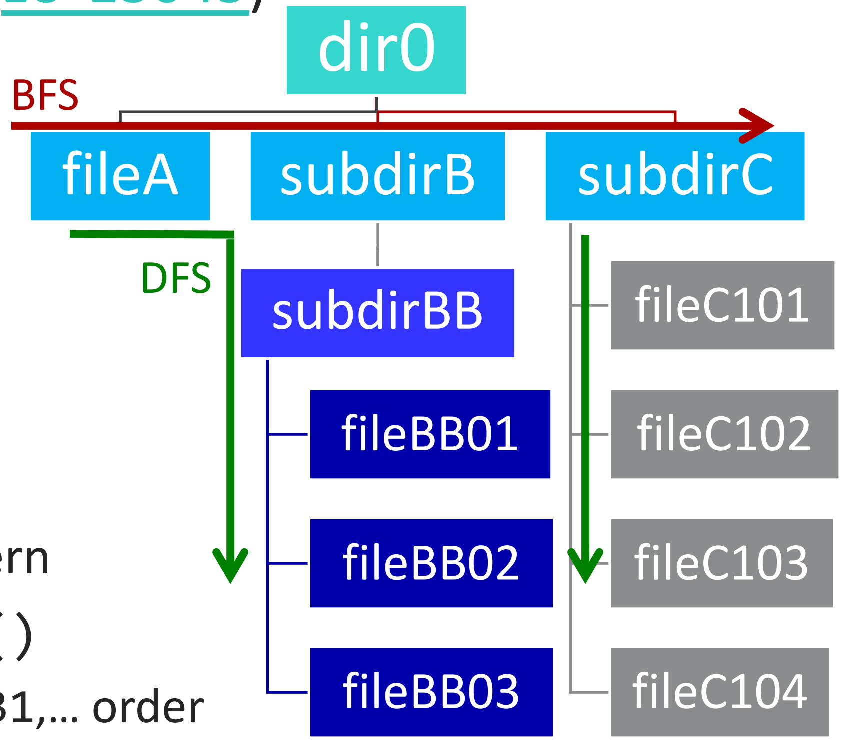
2.17
- Lustre Metadata Robustness/Redundancy (LU-12310)
  - Phase 1 to distribute/mirror MDT0000 services to other MDTs

Batched Cross-Directory Statahead (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)
  - Aggregate getattr RPCs for existing statahead mechanism
- **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
  - Detect strided pattern for alphanumeric ordered traversal + `stat()`
    - e.g. `file00001`,`file001001`,`file002001`… or `file1`,`file17`,`file31`,… order

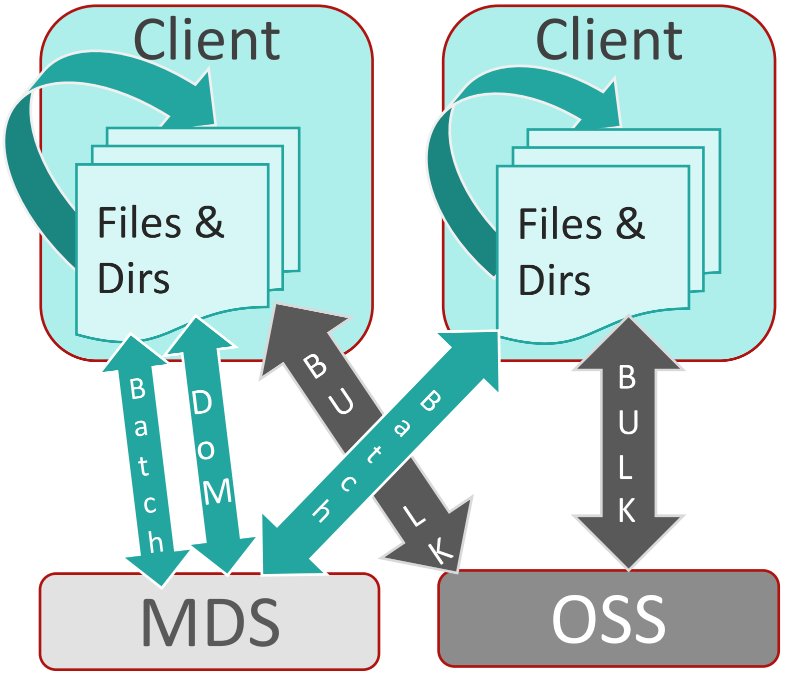


Metadata Writeback Cache (WBC2) (WC 2.17+)

10-100x speedup for single-client file/dir create-intensive workloads

- Genome extraction/processing, untar/build, data ingest, producer/consumer

- ▶ Create new dirs/files **in client RAM without RPCs** (LU-10983)
  - 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
  - Repeat as needed for subdirectories being accessed remotely
  - 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
  - Able to benchmark under intensive multi-client workloads
- ▶ WBC for pre-existing directories, PCC integration in later release
  - Read all directory entries before create to avoid duplicate filenames



Enhance Dir Migration (LU-17148) (2.17)

Whamcloud

Enhance directory migration robustness

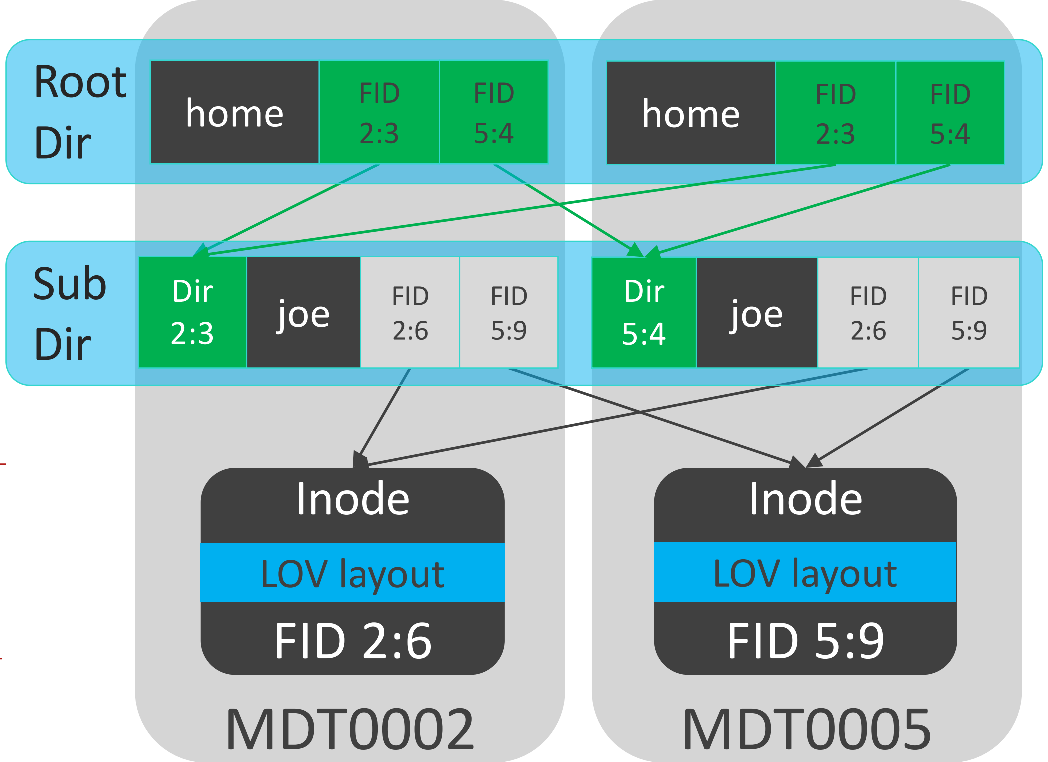
- Directory migration relies on recovery to avoid file missing upon failure
- Once recovery is aborted, a different mechanism should be applied:
  - Source inodes and dirents of sub files are kept till migration of this directory finishes
  - Upon a directory migration finish, sync directory and verify sub file targets against retained sources
  - If some files are missing, migrate them, and redo the above step
  - Otherwise remove retained sources
- MDT resumes failed migrations automatically

Lustre Metadata Redundancy (LU-12310) (2.17+)

Whamcloud

Improve metadata (data) availability in face of network/server errors

- In early discussion and planning stages



▶ LMR1a: Redundant services on other MDTs
- Mirror FLDB, Quota, `flock()` across MDTs

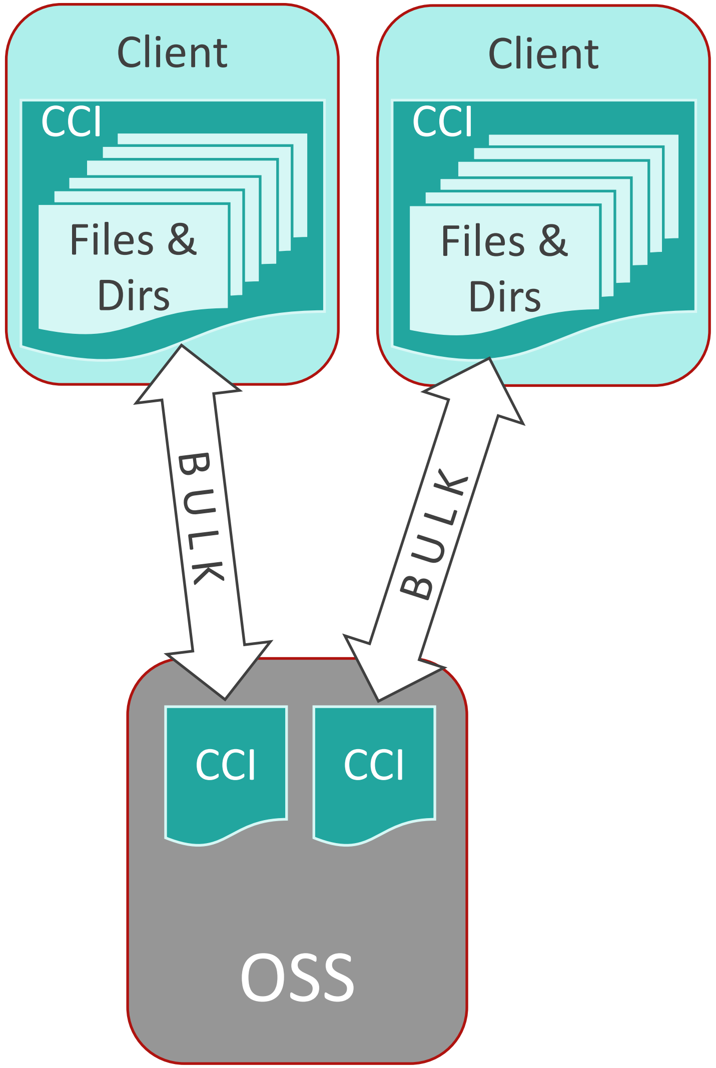
▶ LMR1b: DNE transaction performance
- Need to optimize if all `mkdir` are mirrored transactions
- Better distributed transaction logging format

2.17
- Improves **all** DNE operation performance

2.18 ▶ LMR1c: Replicate top-level dirs for availability
- ROOT/ directory (rarely changed) mirrored to 2+ MDTs
- No replication for regular file inodes in this phase

2.19+ ▶ LMR2/3 phases needed for full MDT redundancy
- Full directory tree replication, file inode replication
- Configurable mirror setting per directory/tree (`lfs setdirstripe`)
- Recovery, LFSCK, rebuild replicated directories if full MDT loss

Client Container Image (CCI)   (2.18+)



Need improved handling of aggregations of many files

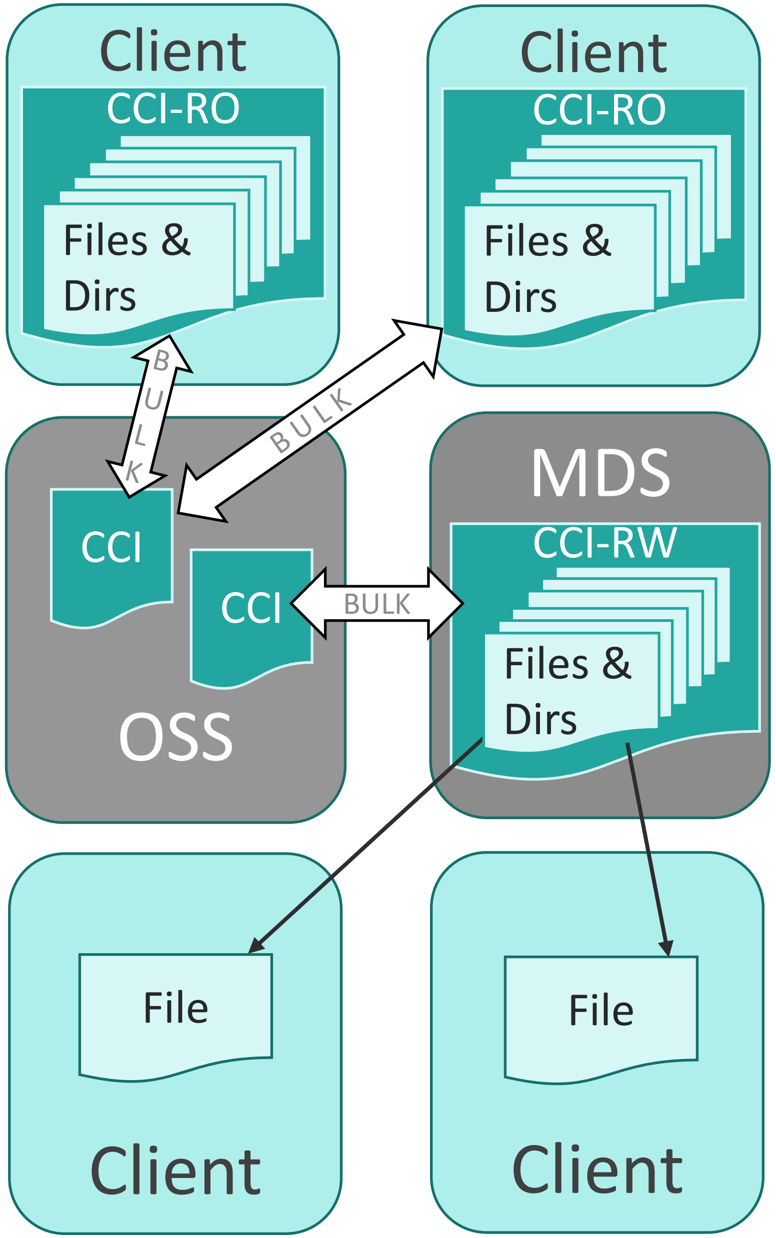
- Create, access, (modify?), delete many files without untar/unzip

▶ CCI allows efficiently accessing filesystem image many files

▶ **Low I/O overhead, few file lock(s), high IOPS per client**
  - Readahead and write merging for data and metadata
  - Client-local in-RAM filesystem operations with very low latency

▶ Access, migrate, replicate image with large bulk OSS RPCs
  - Thousands of files aggregated with MB-sized network transfers
  - Leverage existing high throughput OSS bulk transfer rates
  - 1GB/s OSS read/write is about 30,000 32KB files/sec

▶ Unregister+delete container to mass-delete **all** files within
  - Simplifies user data management, accounting, job cleanup
  - Avoid MDS overhead when dealing with groups of related files



Client Container Image (CCI) (2.18+)

- Ext4 filesystem images used *ad-hoc* with Lustre today
  - **Read-only cache** of many small files manually mounted on clients
  - **Root filesystem images** for diskless clients/VMs
- ***Container Image* is local ext4/ldiskfs image** mounted on client
  - Directory tree (maybe millions of files) stored in one CCI Lustre file
  - **Best for self-contained workloads** (AI, Genomics, ensemble runs)
  - Can configure with job preamble script today, for read-only data
- CCI automates container image handling into Lustre TODO
  - Image is registered to Lustre directory to control future access
  - **Transparently mount** registered image **at client** on directory access
  - Image data blocks read (written) from/to OST(s) and/or client cache
  - Automatically unmount from client when idle or under contention
- Internally mount and export from MDS for multi-client access
  - Will need some modifications to CCI file to *Lustre-ize* files

Comparison and Summary of WBC vs. CCI

Metadata Writeback Cache

- Keep normal namespace
- **Transparent to users**
- Very low latency metadata operations
- **Faster single client**
- Network **batch RPCs** improves other ops
- **Lower total overhead** due to fewer layers

Client Container Image

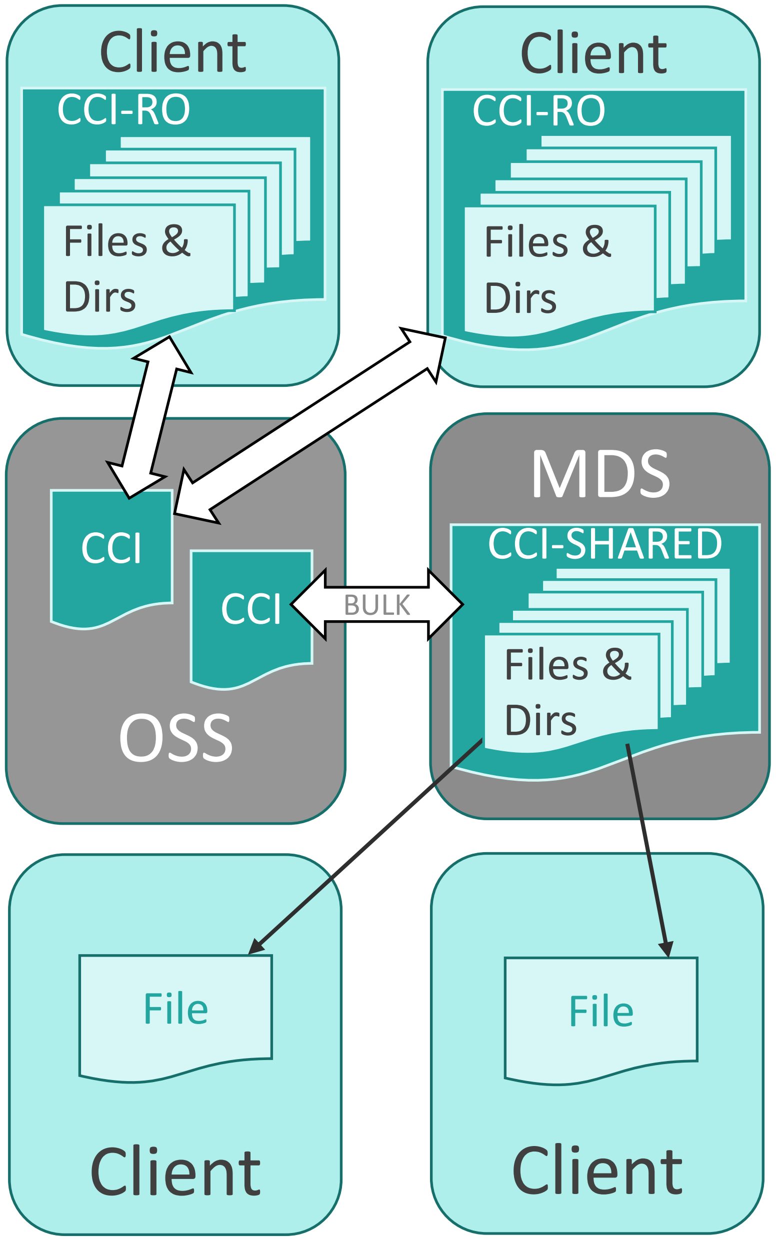
- **Segregated directory subtree**
- Needs coordination with user/job
- Not for all usage patterns
- **Faster total performance**
- Network **bulk IO** avoids MDS workload
- Bulk file/data management (e.g. fast unlink)
- **Metadata tiering/HSM aggregation**

- Significant improvements for evolving HPC workloads
- Leverages substantial functionality that already exists
- Needs supporting project to drive steady progress

Whamcloud

Thank You!

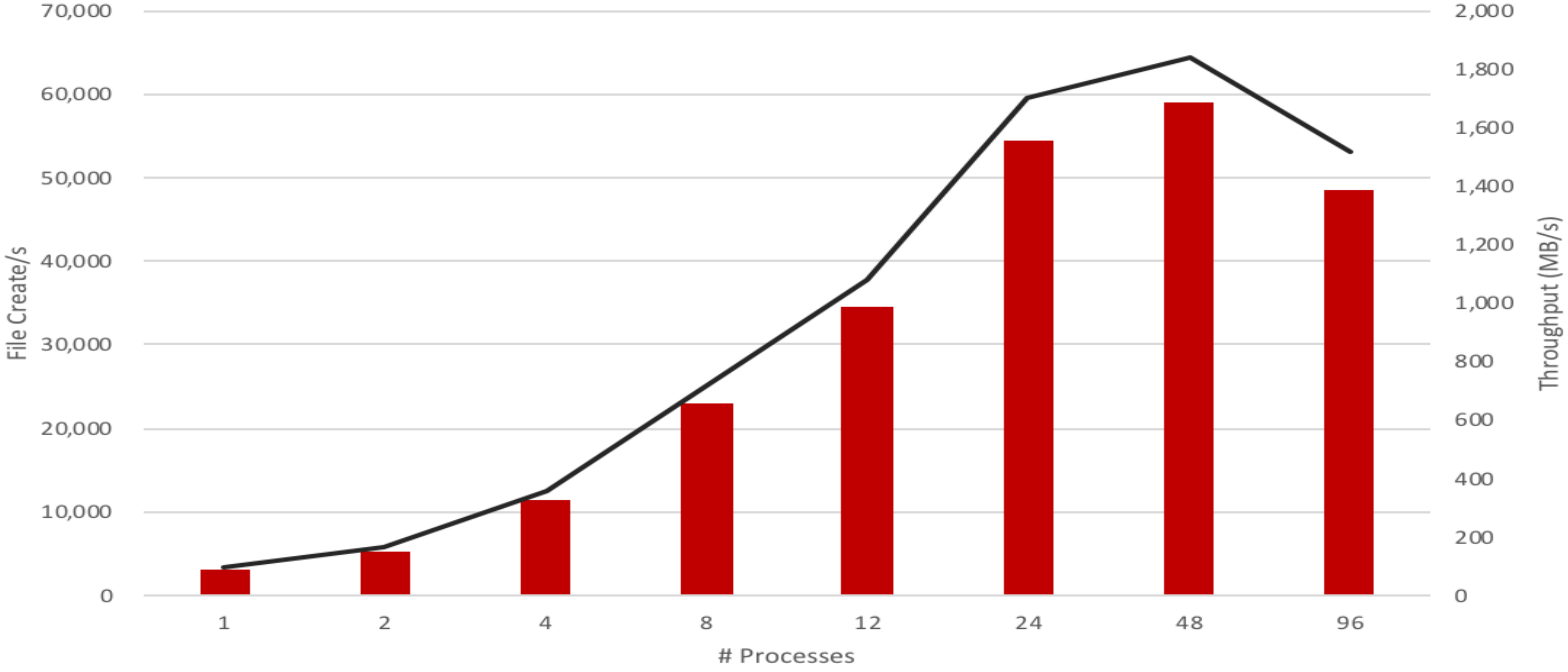
Questions?



CCI Access Models

- Need to integrate image handling on Lustre client/MDS
  - Integrate CCI creation with job workflow is easiest
  - CCI layout type on parent directory creates CCI upon mkdir
  - Enhance ldiskfs online resize to manage image size
- Client ***exclusively*** mounts CCI(s) and modifies locally
  - For initial image creation/import from directory tree
  - For workloads that run independently per directory
- Multiple clients ***read-only*** mount single image
  - Shared input datasets (e.g. gene sequence, AI training)
- MDS exports ***shared read-write*** image to many clients
  - Internal mount at MDS attaches image to namespace
  - Use Data-on-MDT to transparently export image tree to clients
- Process whole tree of small files for HSM/tiering
  - Efficiently migrate tree to/from flash tier, to/from archive

Single Client **32KB** File Create Performance (MDS+OSS)

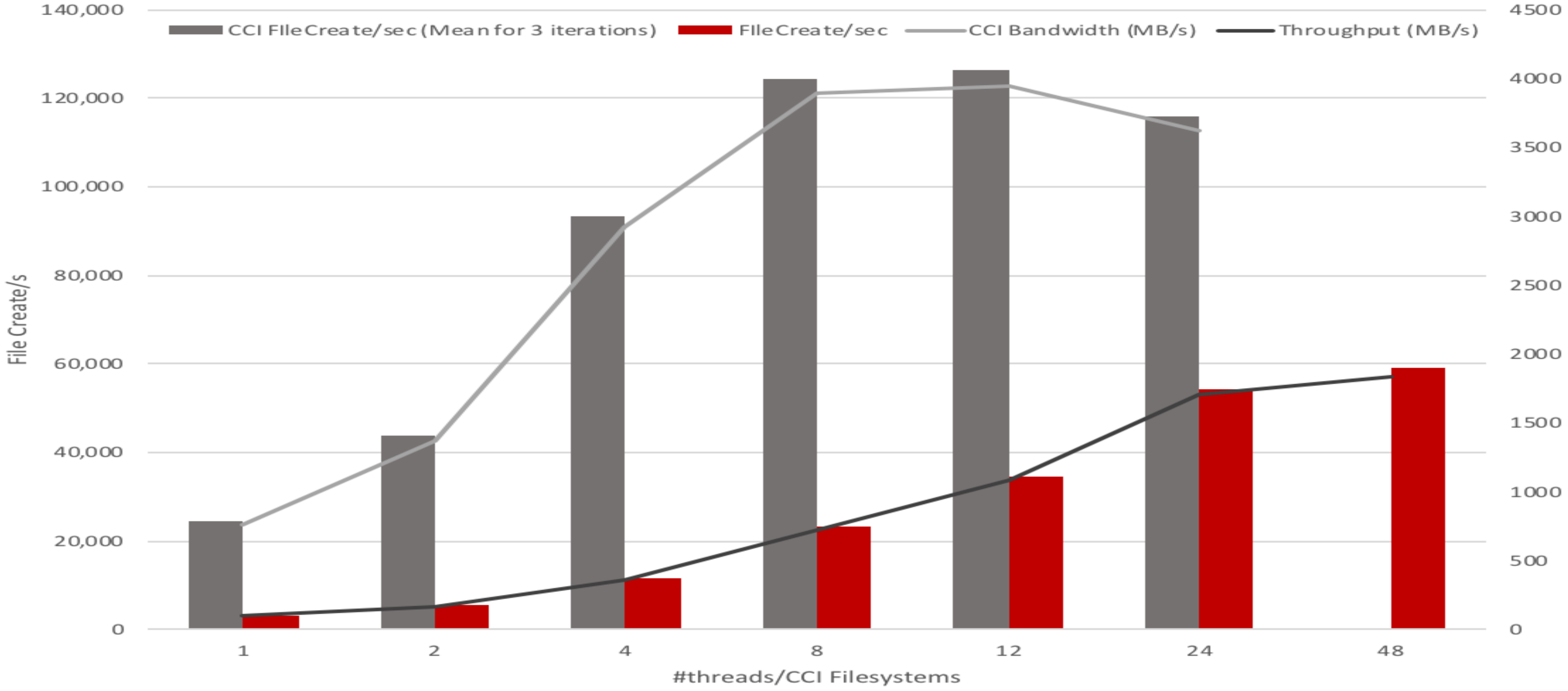


1 client, n processes, 12000 files/process

mdtest -n 12000 -d ${OUTDIR} -u -v -p 20 -w 32768 -e 32768 -F -i 3

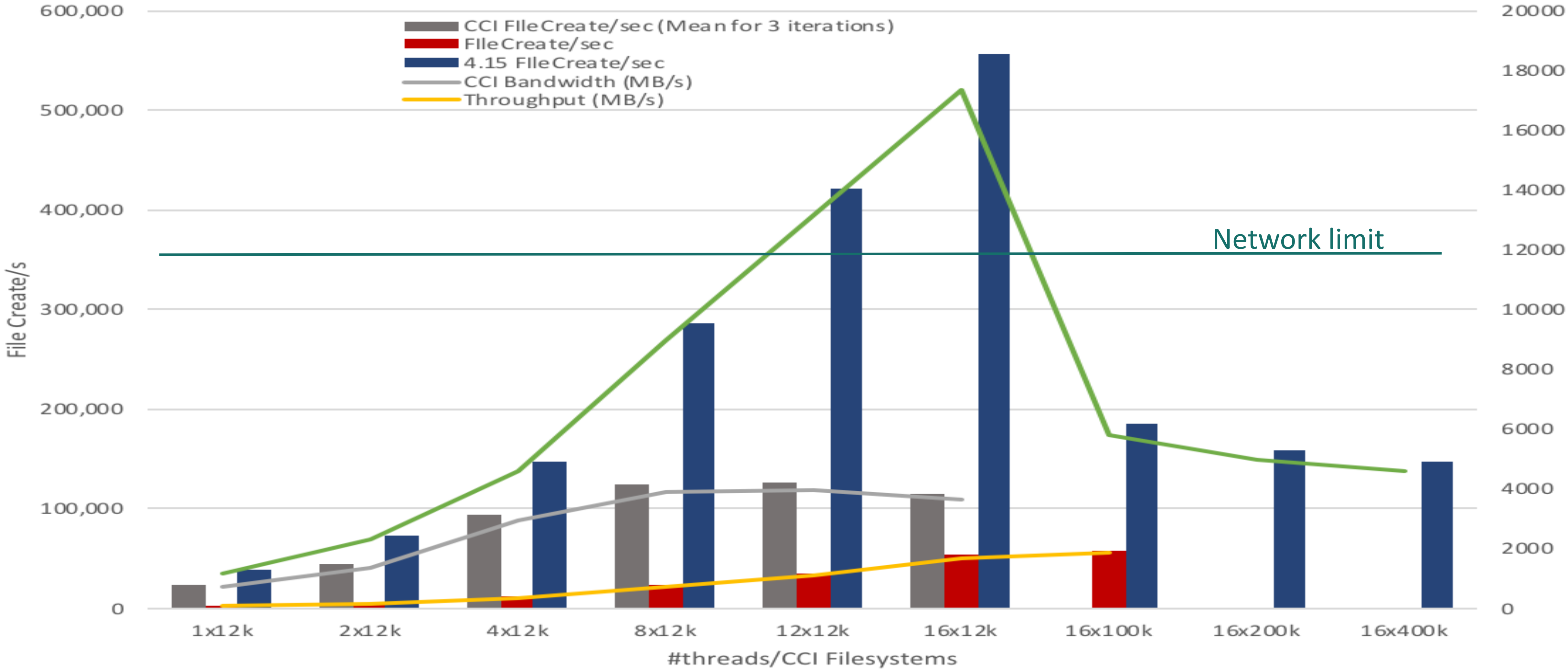
MDS, OSS: 6x 960GB NVMe, 2.7GHz 24-core 8186, 48GB RAM

Single Client **32KB** File Create Performance (MDS vs. CCI)



- Early testing of manually-configured CCI shows significant promise

Single Client **32KB** File Create Performance (MDS vs. CCI)



- 4.15 CCI improvement due to Ubuntu 4.15 kernel loopback driver
- Early testing of CCI prototype shows promise