whamcloud

IEEE MSST
Asilomar Tutorial
April 2012

Lustre Metadata Scaling

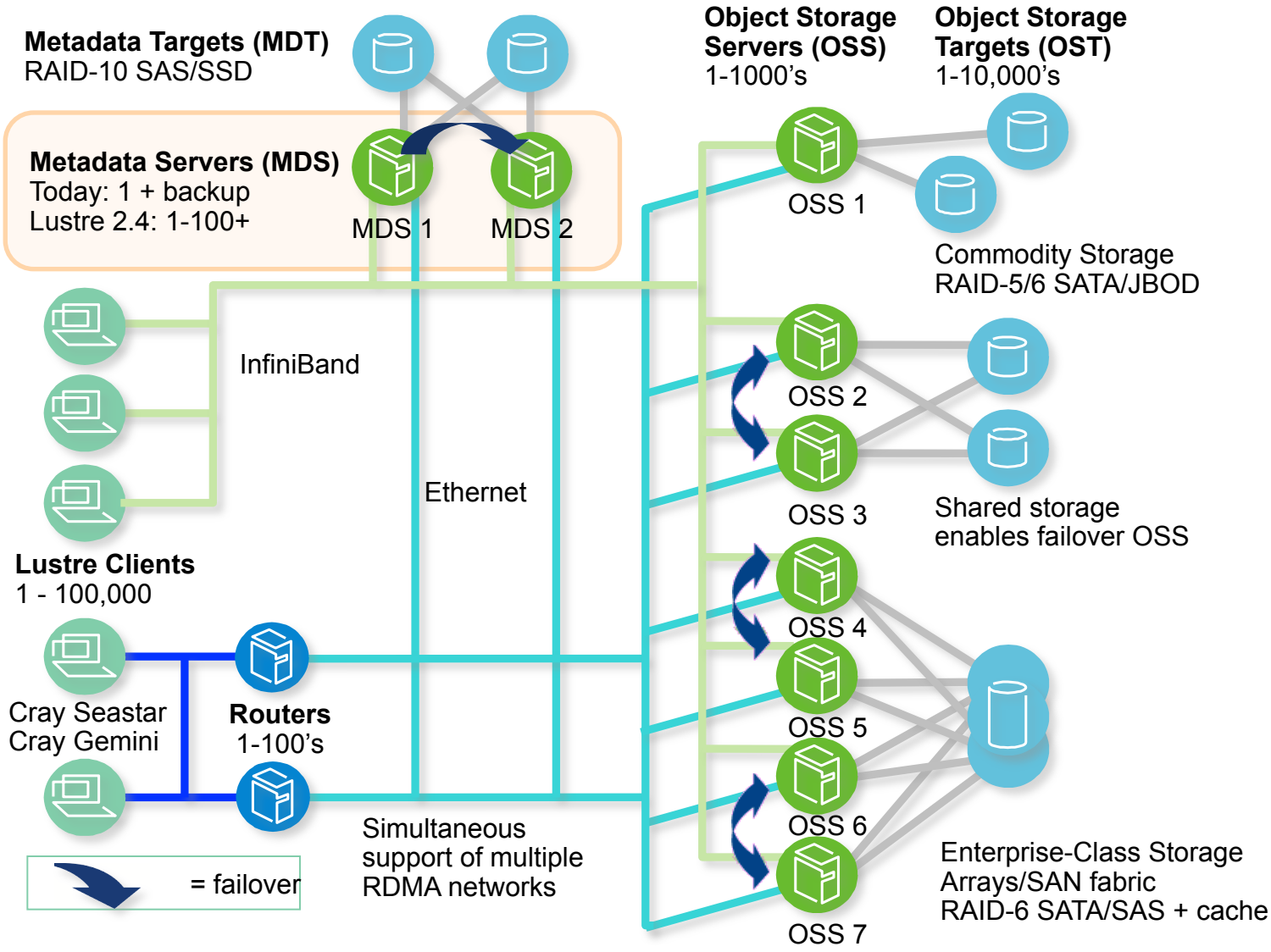
- Andreas Dilger
  Principal Lustre Engineer
  Whamcloud, Inc.
  adilger@whamcloud.com

Agenda

- Lustre overview
- Lustre architecture
- Recent metadata improvements
- In progress metadata improvements

What is Lustre?

- A scalable distributed parallel filesystem
- Hardware agnostic
  - Can use commodity servers, storage, and networks
  - Many vendors also integrate with their hardware/tools
- Open source software (GPL v2)
  - Ensures no single company controls Lustre
  - Protects users and their storage investments
  - Large, active, motivated development community
- POSIX compliant
  - What applications expect today…
    … though Lustre is flexible for future demands
- The most widely used filesystem in HPC
  - 7 or 8 of top 10 supercomputers for many years
  - ~70 of top 100 systems in most recent Top-500



Lustre Architecture Overview

- Client config from mgmt server at mount
- Client operations split into metadata/data
  - Each operation class goes to a dedicated server
  - RPCs to servers are independent of each other
- Metadata Server (MDS = node)
  - One (or more soon) Metadata Targets (MDT = filesystem)
  - Stores dirs, filenames, mode, permissions, xattrs, times
  - Allocates data object(s) layout for each file

Lustre Architecture Overview cont.

- Object Storage Servers (OSS = node)
  - One or more Object Storage Targets (OST = filesystem)
  - Objects store file data, size, block count, timestamps
  - Files may be striped across N objects/storage targets
  - Each object has its own extent lock/server per OST
  - IO/locking to OSTs is completely independent
- Client merges meta/data on read/stat
  - File size and timestamps remain distributed
    - File size is computed based on layout and object size(s)
    - Visible timestamp is based on most recent ctime
  - POSIX is an attribute of the client, not server or protocol

How Lustre MDS Is Different

- Does NOT manage block/chunk allocation
  - OST objects are variable sized
  - Block allocation handled internally by OST filesystem
- NOT needed for file IO after open
  - Only needed again at file close time
- Does not manage file size/write timestamps
  - Handled by client and OSTs
- Does not lock or manage file data
  - Data locking distributed across OSTs for each object
- Performs modifications asynchronously
  - Clients participate in RPC replay on MDS failure

Backend Filesystems

- Lustre uses existing filesystems
  - High cost/effort to develop and maintain disk filesystem
  - Takes 5+ years to get a stable and robust filesystem
  - Able to leverage ongoing improvements in filesystems
  - Lustre improvements can be returned upstream
- Provide object storage to network
  - Block allocation, attribute storage are abstracted from Lustre
  - Clients do not interact with disk or filesystem structures
  - Filesystem is protected from misbehaving clients
  - Do not need to micro-manage on-disk resources

Backend Filesystems In Use

- Currently use *ldiskfs* in production
  - Based on Linux ext4/e2fsprogs with additional features, APIs
  - Also possible to say ext4 is based on ldiskfs ☺
- ZFS will be an option for Lustre 2.4 (2013)
  - Improve Lustre storage abstraction to integrate with ZFS
  - Immediately leverage major ZFS benefits on Linux
    - robust code with 10+ years maturity
    - data checksums on disk
    - online check/scrub/repair
    - pooled disk management
    - integrated with flash storage cache
  - http://zfsonlinux.org/lustre.html

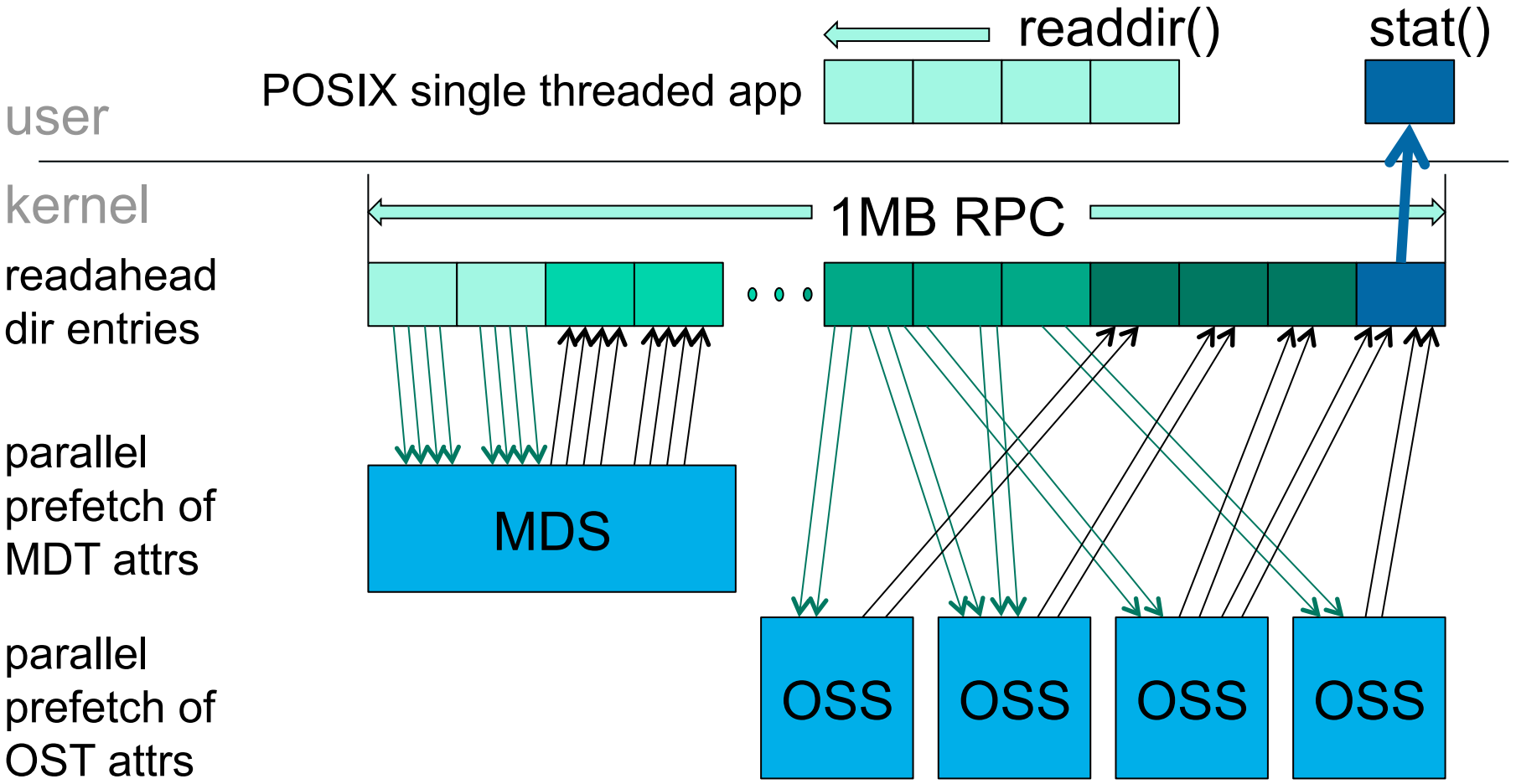
Improving Single Client Performance

- Not a primary HPC performance target
  - Most HPC jobs saturate server capacity
- Some tasks not easily parallelized (users :-)
  - *ls -l* on a large directory (or always with *ls --color* alias)
  - *find* on a large directory tree
- Can impact distributed performance
  - Lock contention on shared single file
  - Client flushes dirty data before it can release lock
  - Blocks other clients needing this lock until it is released

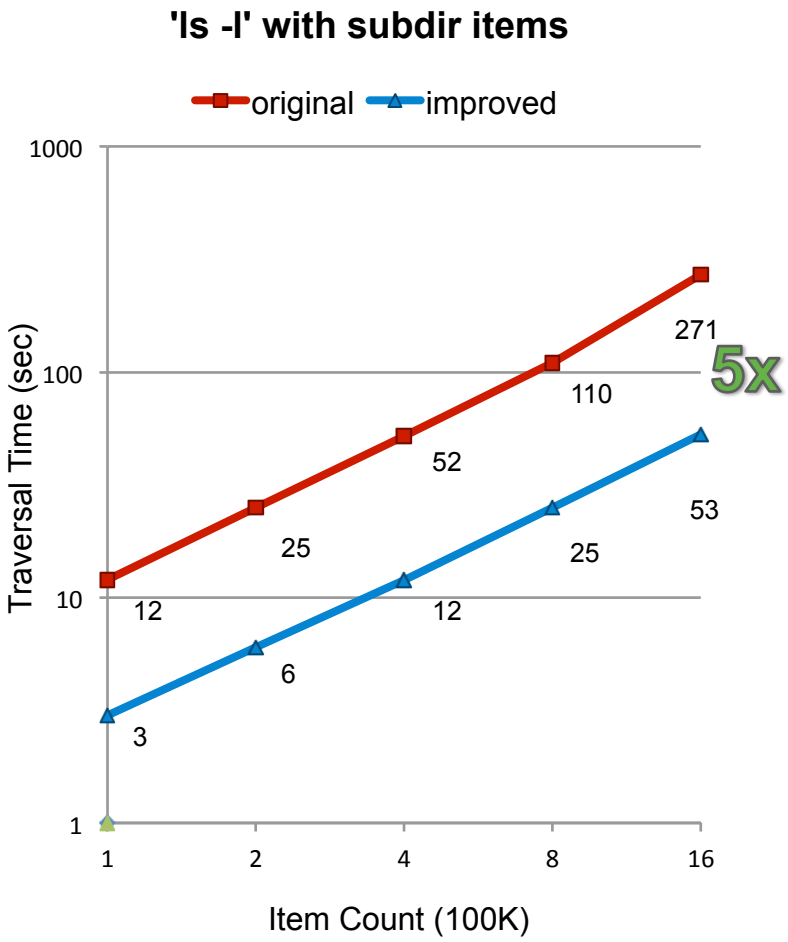
Accelerated Client Directory Traversal

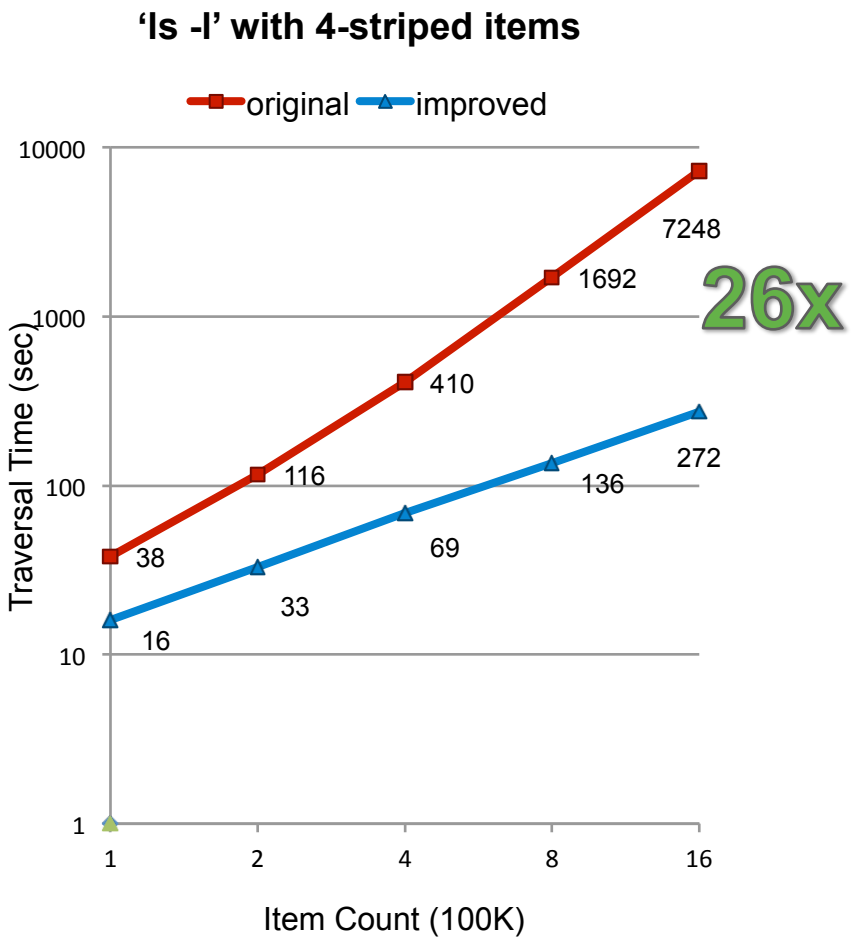
- Common to *ls*, *du*, *find*, etc.
- Large readdir RPCs
  - Formerly 1 page (typically 4096 bytes) per RPC
  - Now up to 1 MB RDMA per RPC to reduce overhead/latency
- readdir() + stat() is a common use pattern
  - POSIX APIs only operate on a single directory entry
  - readdirplus() not in POSIX yet
  - Need file size from OSTs
  - Kernel detects sequential access, starts *statahead*
  - Multi-threaded RPC engine active even for a single user thread
  - Significant performance improvement for WAN usage
  - MDS locks and fetches UID, GID, nlink on inodes in parallel
  - OST locks and fetches object size, blocks, mtime, ctime

Statahead in Kernel



Directory Traversal Improvement



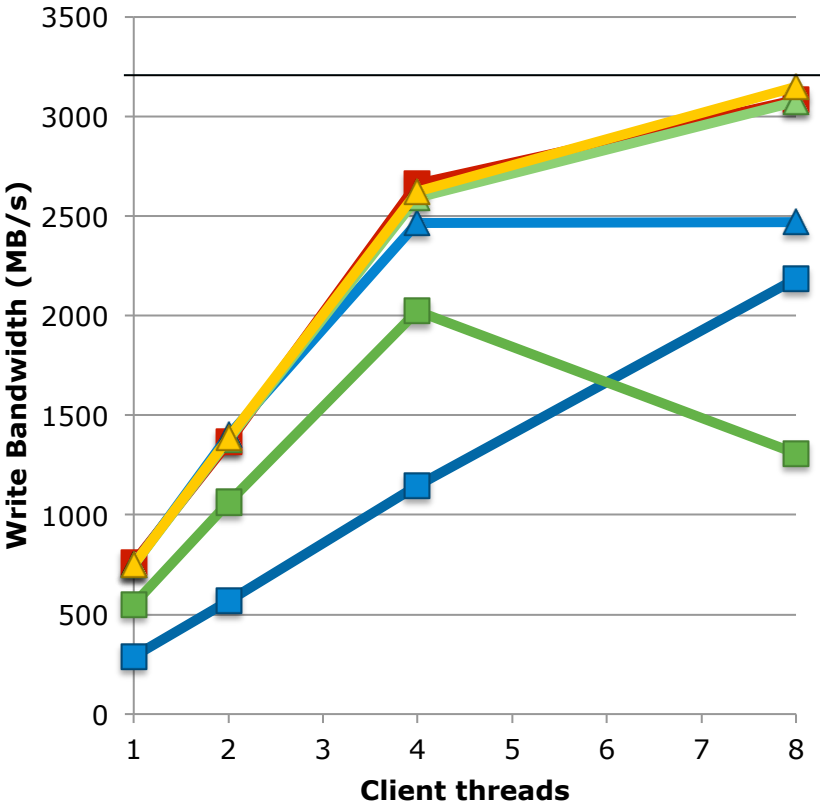


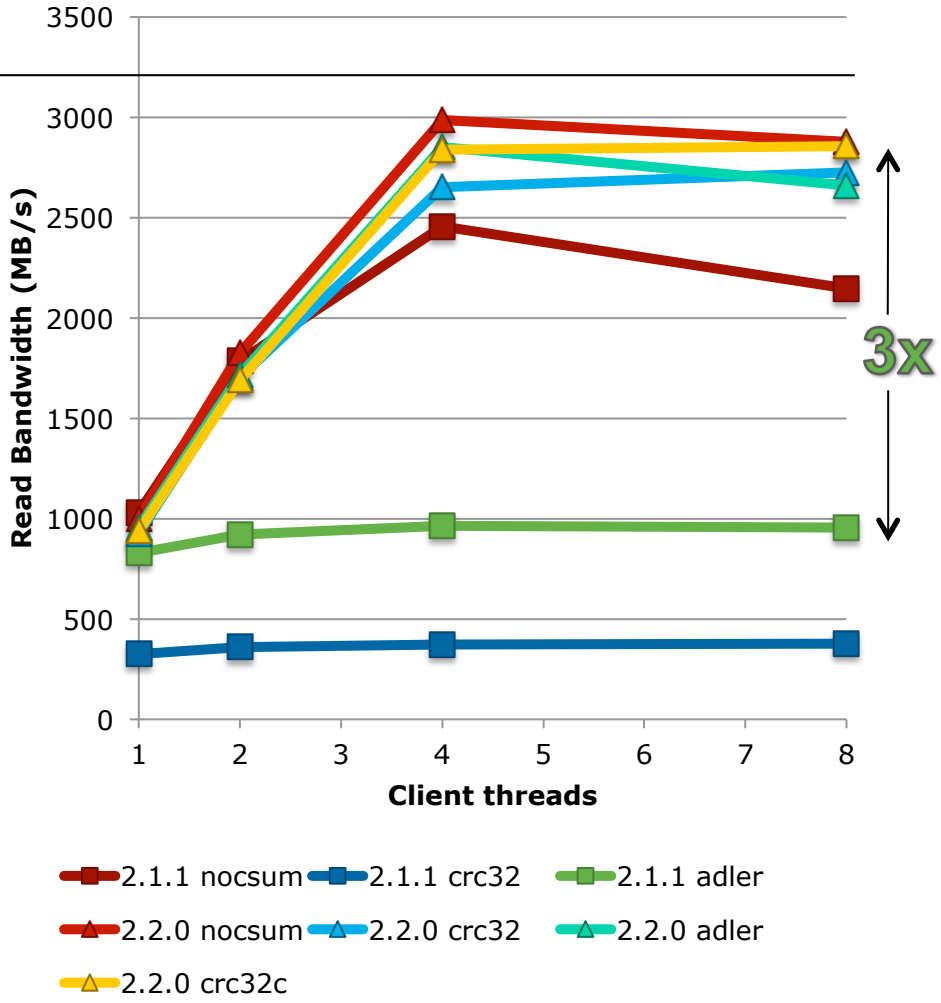
Improved Client Data Checksums

- End-to-end data integrity becoming critical
  - Petascale filesystems have 100-1000's of servers, 10000s of disks
- Lustre has data RPC checksums for 4+ years
  - Fine for many-client IO, but a bottleneck for single-client performance
  - Checksum also competes with data copy from userspace
  - Single-threaded IO is CPU bound, cores are not getting faster
- Use multiple RPC threads for doing checksums
  - Checksum does not need to be done in application process context
  - Increases parallelism for single-threaded userspace IO
- Leverage newer hardware support for checksums
  - Nehalem+ CPUs contain CRC32c checksum engine

Single Client Checksum Performance

IOR file per process, one stripe/file, 8 OSTs, QDR InfiniBand



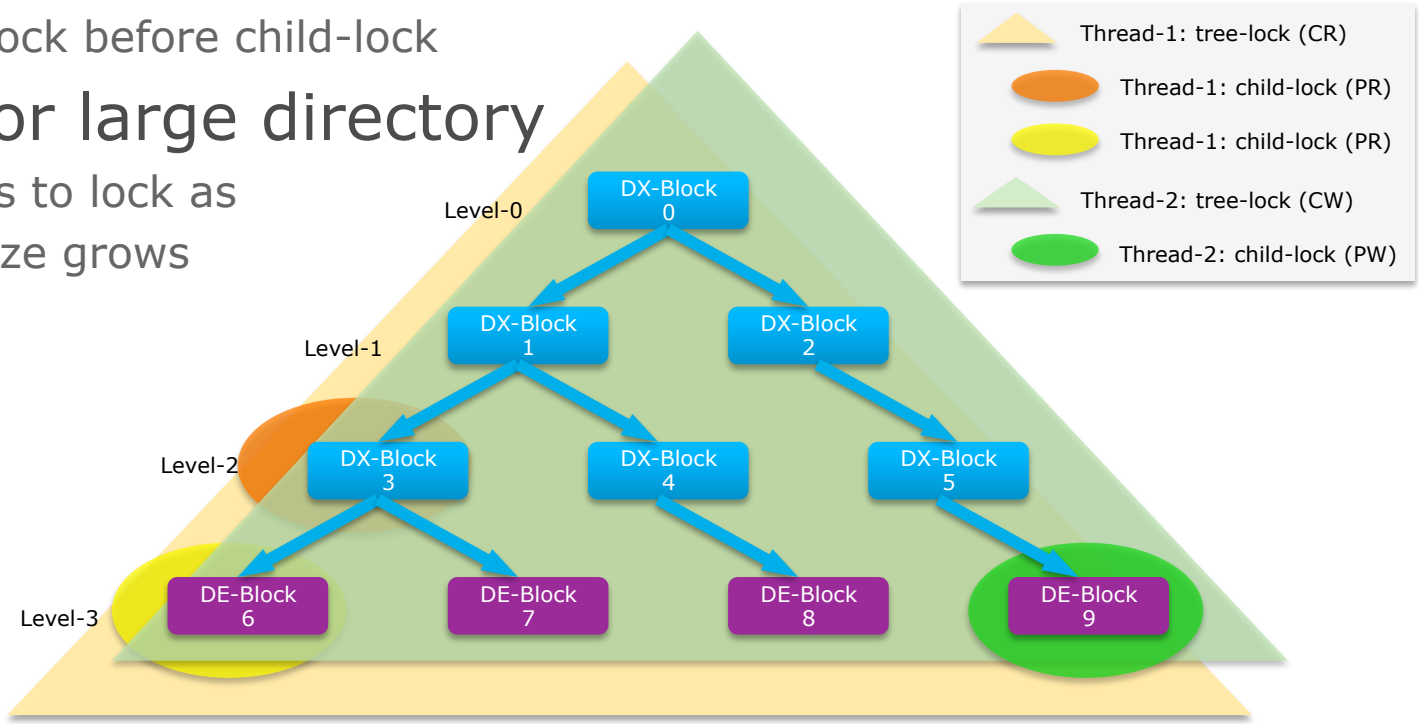


Improving Metadata Server Throughput

- Network/RPC/thread efficiency
  - Better CPU affinity (less cacheline/thread pinging)
  - Service threads awoken in MRU order
  - Multiple RPC request arrival/queues for network
  - More balanced internal hashing functions
- Parallel directory locking
  - Parallel directory DLM locking was in Lustre 2.0
  - Testing showed bottleneck in ext4 directory code
  - Add parallel locking for directory operations
  - Concurrent lookup/create/unlink in one directory
  - Improves most common use case for applications

Parallel Directory Locking

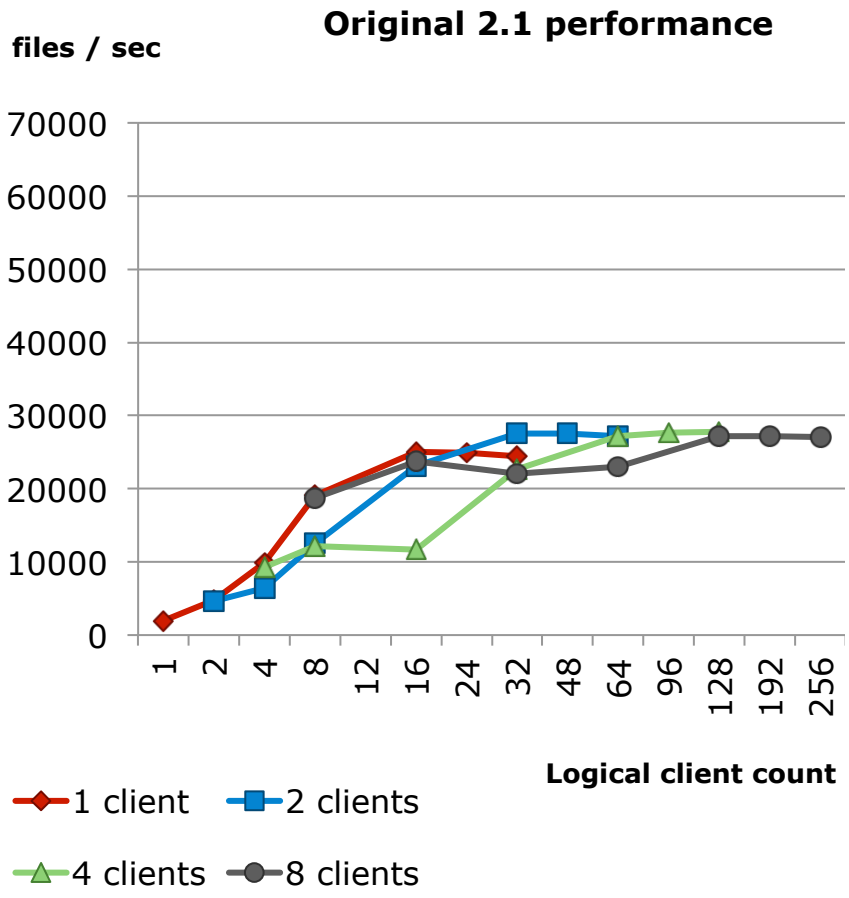
- Protect tree topology
  - Optimistically lock top levels of ext4 directory tree in shared mode
  - Lock bottom level(s) as needed for operation (read/write)
  - Backout and retry if leaf/node split is needed
  - Take tree-lock before child-lock
- Scalable for large directory
  - More leaves to lock as directory size grows

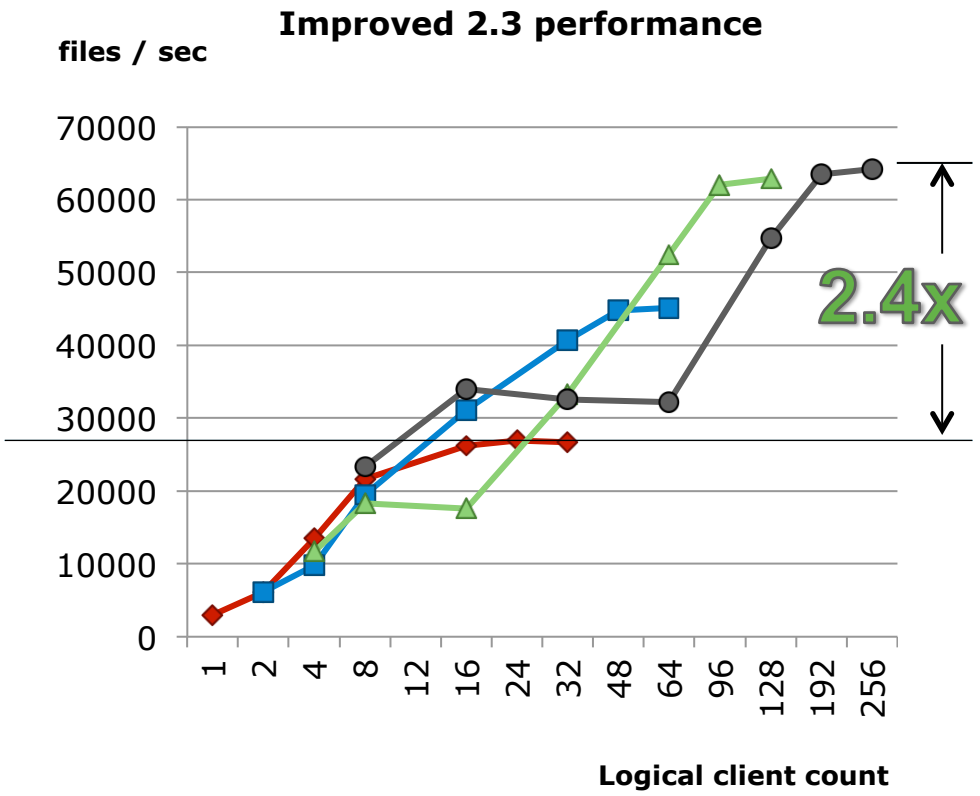


Graph-2 : htree and htree-lock

Single Shared Directory Open+Create

1M files, up to 32 mounts/client





Single File Wide-striping Support

- Now up to 2000 stripes per file
  - Layout stored in extended attribute (xattr)
  - Can allocate single file over all OSTs
- Add large xattr support to ldiskfs
  - Allocate new xattr inode
  - Store large xattr data as file body of that inode
  - Original file inode points to this new xattr inode
  - Not backward compatible with older ext4 code
- Require larger network buffers
  - Return –EFBIG for old clients with smaller buffers
  - Old clients can still unlink such files (done by MDS)

Distributed Namespace

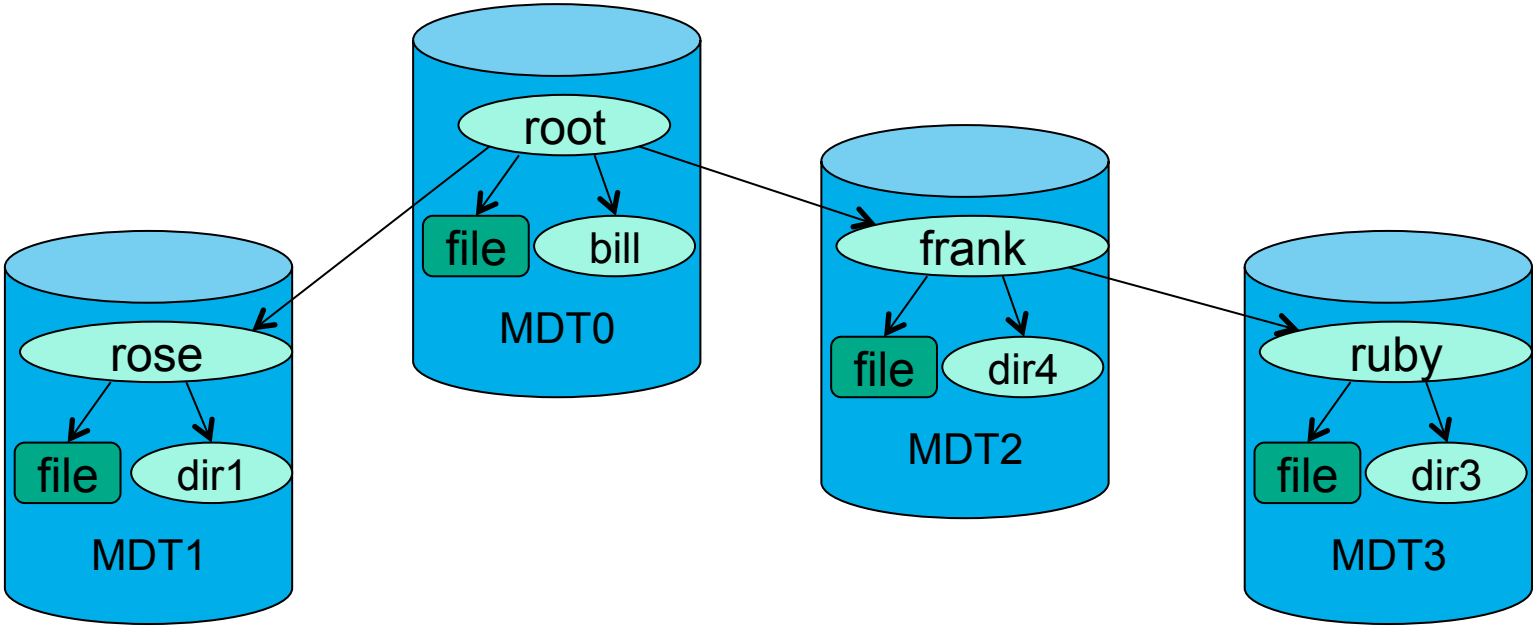
- Inodes normally on same MDT as parent directory
  - Create scalable namespace using distributed (slower) operations
  - Use scalable namespace with non-distributed (fast) operations
  - Scale aggregate throughput
- Funded by OpenSFS
  - Target availability in Lustre 2.4 (March 2013)

Distributed Namespace, cont.

- Phase 1 - remote directories
  - Home/project directories balanced over all MDTs
  - Home/project subdirectories/files constrained to parent MDT
  - Huge filesystems with tens of billions of files
  - Performance requirements beyond a single MDS
- Phase 2 - striped directories
  - Directory entries hashed over directory stripes
  - Huge directories for file-per-process apps
  - O(n) speedup for shared directory ops
  - Complements shared directory improvements

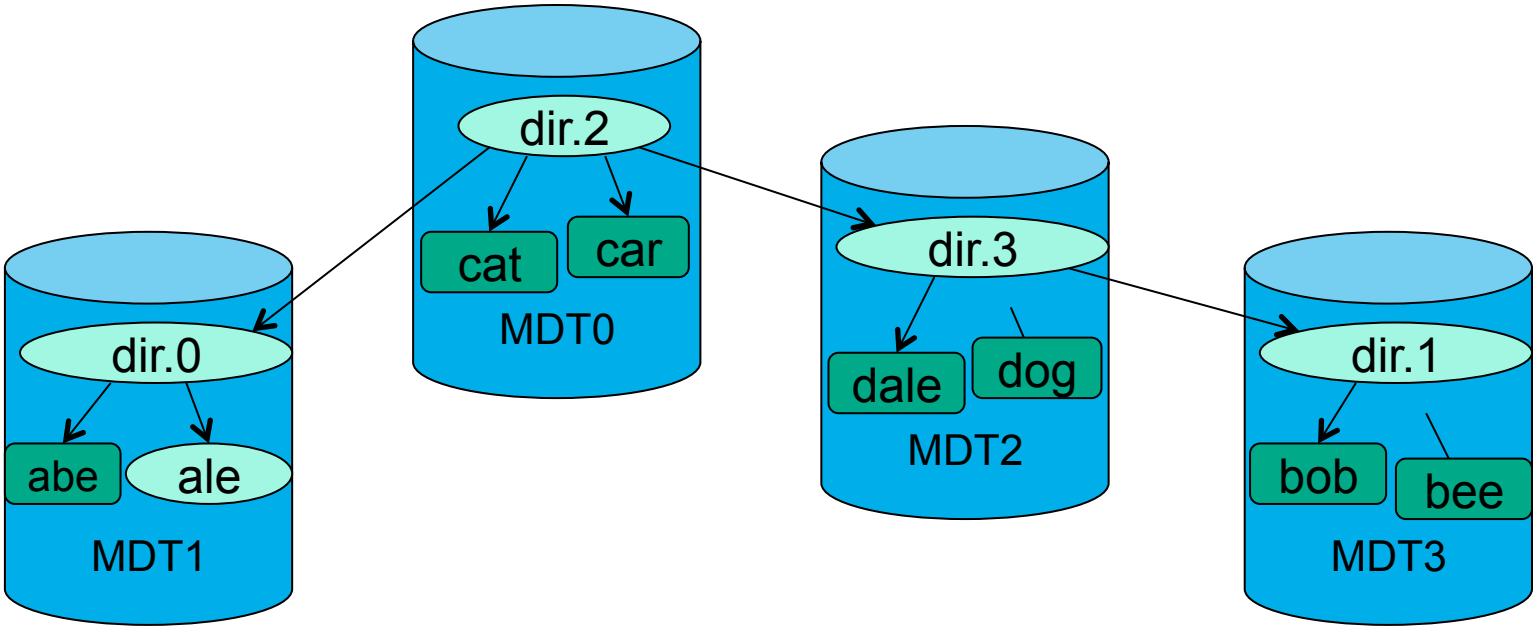
DNE Phase I - Remote Directory

- Subdirs on a remote metadata target
- Scales namespace as data servers can
- Dedicated performance for users/jobs
- Share IO bandwidth or segregate by OST pools



DNE Phase II - Shard/Stripe Directory

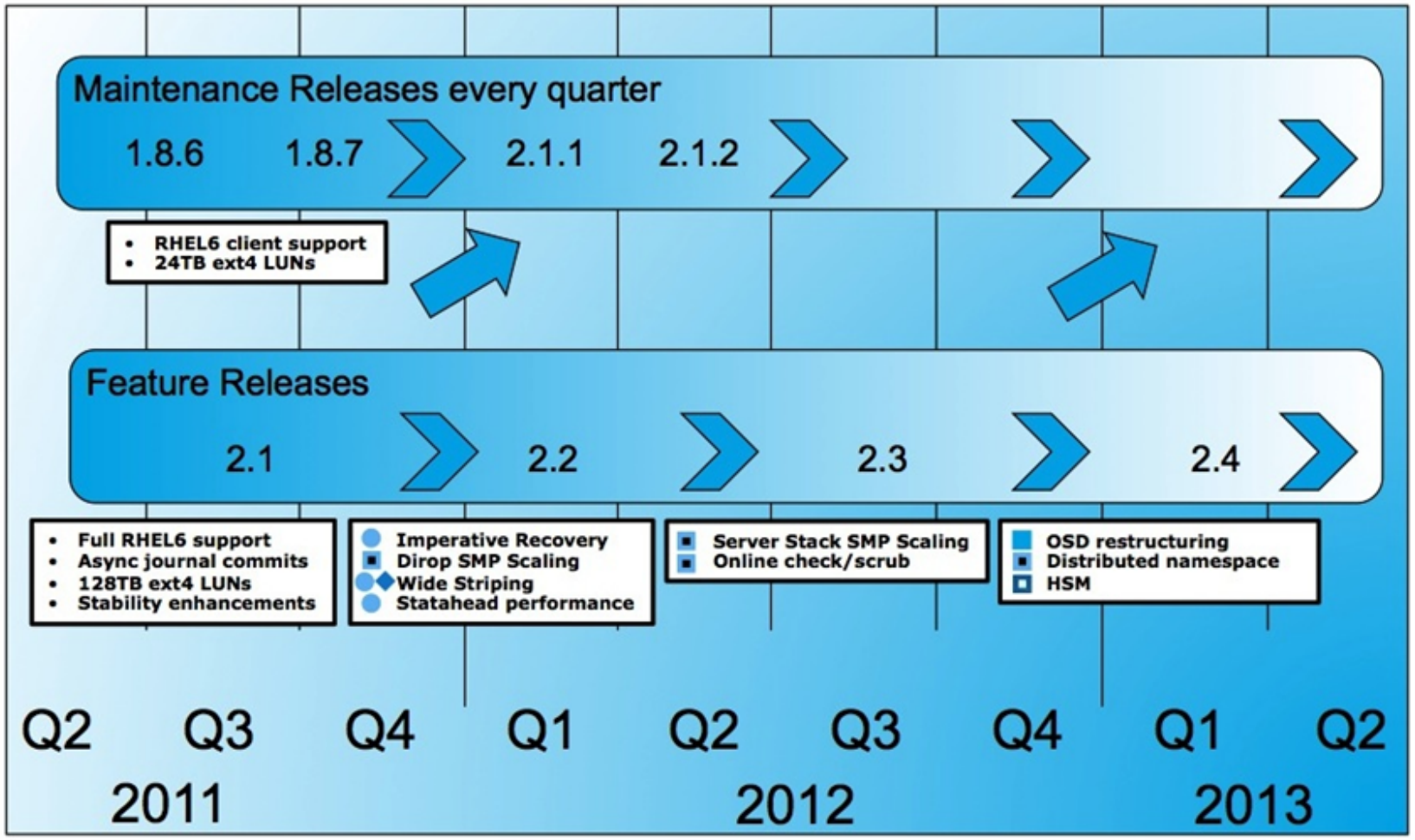
- Hash a single directory across multiple MDTs
- Multiple servers active for directory/inodes
- Improve performance for large directories



Metadata Performance Testing

- New mds-survey tool for metadata testing
- Run directly on the MDS
  - Doesn't require any Lustre clients or OSTs
  - Can optionally include OSTs for object creation
  - Similar to obdfilter-survey, but for metadata
- Allows bottom-up benchmarking
  - Disk IOPS testing
  - Metadata benchmark testing of local filesystem (mdtest, etc)
  - Lustre MDD/OSD stack (mds-survey)
  - Client metadata benchmark testing (mdtest, etc)
- Create/open/lookup/getattr/xattr/delete ops

Community Lustre Roadmap



Lustre 2.1 and 2.2 Releases

- Lustre 2.1.1 GA Q1 2012
  - Fixes from early 2.x evaluators, RHEL 6.2 and OFED 1.5.4
  - LLNL entering production with 2.1 - 3100+ **servers**, 25% of clients
  - LLNL deployment has been "tipping point" for past major upgrades
- Lustre 2.2.0 GA Q2 2012 - Major Features
  - **Parallel Directory Operations**
  - **Directory Statahead**
  - Wide Striping (a.k.a. large xattrs)
  - SMP Server Scaling
  - Imperative Recovery
- Lustre 2.2.0 Additional Enhancements
  - Client parallel data checksums
  - Multi-threaded client RPC engine
  - Metadata server benchmarking tool - mds-survey
  - SLES 11 SP1 servers (community maintained)

Lustre 2.3 and Beyond

- Lustre 2.3 (September 2012)
  - Server SMP metadata performance
  - LFSCK Online OSD check/scrub
  - **OSD Restructuring (ZFS on OSTs)**
- Lustre 2.4 (March 2013)
  - **OSD Restructuring (ZFS on MDTs)**
  - LFSCK Online check/scrub - Distributed repair
  - Distributed Namespace - Remote directories
  - HSM
- More is in the planning/funding stage
  - Working with OpenSFS to prioritize requirements
    - Object mirroring/migration
    - Storage tier management/quota/migration

- Andreas Dilger
  Principal Lustre Engineer
  Whamcloud, Inc.
  adilger@whamcloud.com

Lustre Installation

- Lustre distributed as RPMs
  - http://www.whamcloud.com/downloads/
  - Client .debs available but not officially supported
  - Source is also available via Git
- Servers require Lustre-patched kernel
  - Lustre kernel, ldiskfs modules
  - Lustre user tools, e2fsprogs
  - Lustre 2.4 + ZFS server does not need a patched kernel
- Clients do not need a patched kernel
  - But must run kernel that matches Lustre modules
  - Clients need Lustre kernel modules and user tools

Lustre Userspace Utilities

- mkfs.lustre
  - For initial format of Lustre targets (MDT/OST)
- tunefs.lustre
  - Read and modify some on-disk configuration
- mount.lustre
  - Called via *mount -t lustre* from fstab/HA scripts
- lctl
  - Administrator control tool (parameters, status)
- lfs
  - User helper (*lfs {df, setstripe, getstripe}*, etc)

LNET Configuration

- Not needed for simple TCP network
  - Will use eth0 and needed ethernet interfaces by default
- Needed for InfiniBand or interface selection
- Stored in client/server */etc/modprobe.d/lustre*
  - Skipping the admin network

    ```
    options lnet networks=tcp0(eth1)
    ```
  - InfiniBand network

    ```
    options lnet networks=o2ib0(ib0)
    ```
- Can cover virtually any multi-protocol network
- Can handle multiple RDMA networks efficiently

Formatting MDT/OST

- Need some Linux block device
  - */dev/sdX* (RAID LUN), */dev/md0* (md RAID), */dev/vgX/lvX* (LVM)
- MDT
  - Usually RAID-1+0 storage
  - Size driven by IOPS, and how many files will be created
  - Default 1 inode/2k of MDT device size

  n0# *mkfs.lustre --mgs --mdt --fsname=huge /dev/sdX*
- OST
  - Size driven by bandwidth, and much data will be created
  - Can format OSTs in parallel

  n1# *mkfs.lustre --fsname=huge --mgsnode=n1 --ost --index=N /dev/sdY*

Mounting MDT/OST/Clients

- Mount servers first, then clients

```
n0# mount -t lustre /dev/sdX /mnt/mdt0
n1# mount -t lustre /dev/sdY /mnt/ost0
    :
    :
cli# mount -t lustre n0:/huge /huge
```

- Check the filesystem is mounted

```
cli$ lfs df
UUID                       bytes        Used   Available Use% Mounted on
huge-MDT0000_UUID          8.0G       400.0M       7.6G   0% /huge[MDT:0]
huge-OST0000_UUID         32.0T       400.0M      32.0T   0% /huge[OST:0]
huge-OST0001_UUID         32.0T       400.0M      32.0T   0% /huge[OST:1]
filesystem summary:       64.0T       800.0M      64.0T   0% /huge
```