Milestone Completion for Implementation Milestone 3 of the Distributed Namespace Project of contract SFS-DEV-001.

Revision History

Date	Revision	Author
2012-12-17	Original	R. Henwood
2012-12-18	Detail on regression	R. Henwood
2012-12-21	Updated performance measurements	R. Henwood
2012-12-21	Date for included Manual snapshot.	R. Henwood

Contents

Introduction

The following milestone completion document applies to Subproject 2.1 – Remote Directories subproject within the OpenSFS Lustre Development contract SFS-DEV-001 signed 7/30/2011.

Subproject Description

Per the contract, DNE 1: Remote Directories is described as follows:

> *This subproject distributes the Lustre namespace over multiple metadata targets (MDTs) under administrative control using a Lustre-specific mkdir command. Whereas normal users are only able to create child directories and files on the same MDT as the parent directory, administrators can use this command to create a directory on a different MDT. The contents of any directory remain limited to a single MDT. Rename and hardlink operations between files and directories on different MDTs return EXDEV, forcing applications and utilities to treat them as if they are on different file systems. This limits the complexity of the implementation of this subproject while delivering capacity and performance scaling benefits for the entire namespace in aggregate.*
>
> *Metadata update operations that span multiple MDTs are sequenced and synchronized to create and/or increment the link count on a MDT object before it is referenced by the remote directory entry and to update the remote directory entry before decrementing the link count and/or destroying the MDT object it referenced. Although this may result in an orphan MDT object under some failure conditions, it ensures that the Lustre namespace remains intact under any and all failure scenarios. All the other metadata operations avoid synchronous I/O and execute with full performance.*
>
> *This project includes the implementation of OST FIDs (File Identifiers). These are required to overcome a limitation in the current 2.x Lustre protocol that would otherwise prevent a single file system from having more than 8 MDTs. Addressing this technical debt in the first subproject of DNE avoids protocol compatibility issues that would arise if this feature were implemented after Remote Directories were used in production.*

Milestone Completion Criteria

Per the contract, three Implementation milestones have been defined by Intel. This document is concerned with completing the third Implementation milestone which is agreed as:

> *Performance and scaling testing will be run on available testing resources. The Lustre Manual will be updated to include DNE Documentation.*

These requirements are demonstrated below.

Performance of DNE code on OpenSFS Cluster

Performance tests were run with the 'mdsrate' MPI program on 24 clients on the OpenSFS Functional Test Cluster. Each client has 8 mount points giving a total of 192 threads to drive load to the file system. Each thread operates on 20K files for a given test resulting in a total test load of 3840K files per run. Each test run was completed once. The OpenSFS Cluster hardware is described in Appendix A.

Performance measurements on 2012-12-17

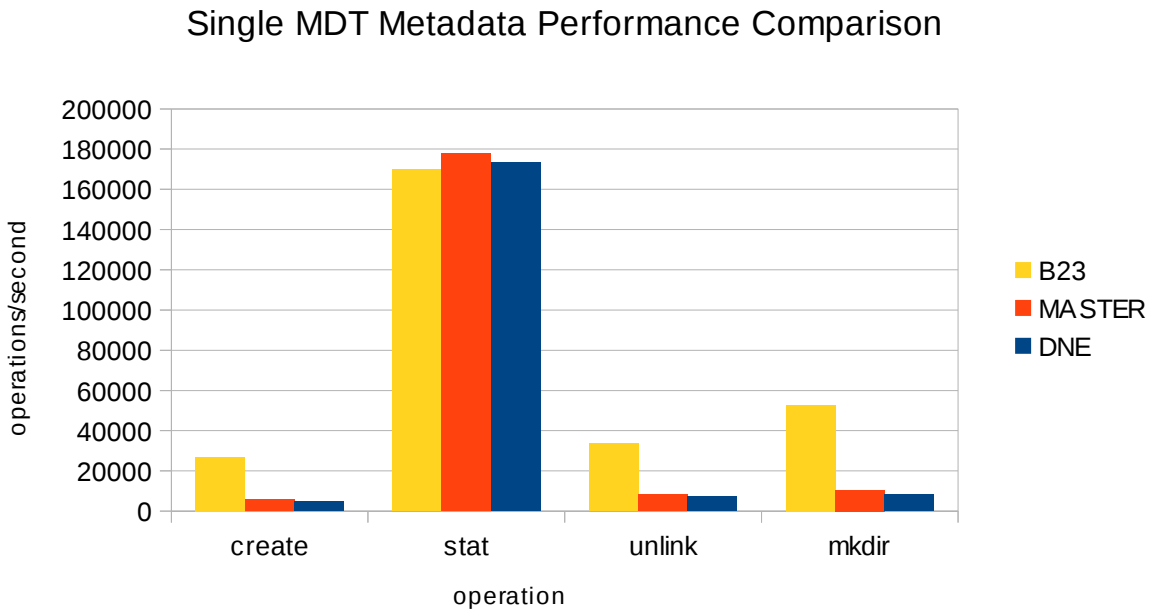


Figure 1: Performance test on OpenSFS Funcational Test Cluster from 2012-12-17

The performance of create unlink and mkdir on the current Master branch is significantly below the level of Lustre 2.3. DNE code has not yet been landed to Master so it is not responsible for this regression. The performance of the DNE-enabled code is comparable to the current Master branch.

After further investigation into the regression, it appears to be caused by quota accounting (LU-2442), which is now enabled by default in 2.4, but not in Lustre 2.3 or earlier. The quota code serializes the metadata operations, which defeats the SMP scaling introduced in 2.3. DNE scaling will be re-tested with quota disabled as part of the next DNE milestone.

Performance measurements on 2012-12-20

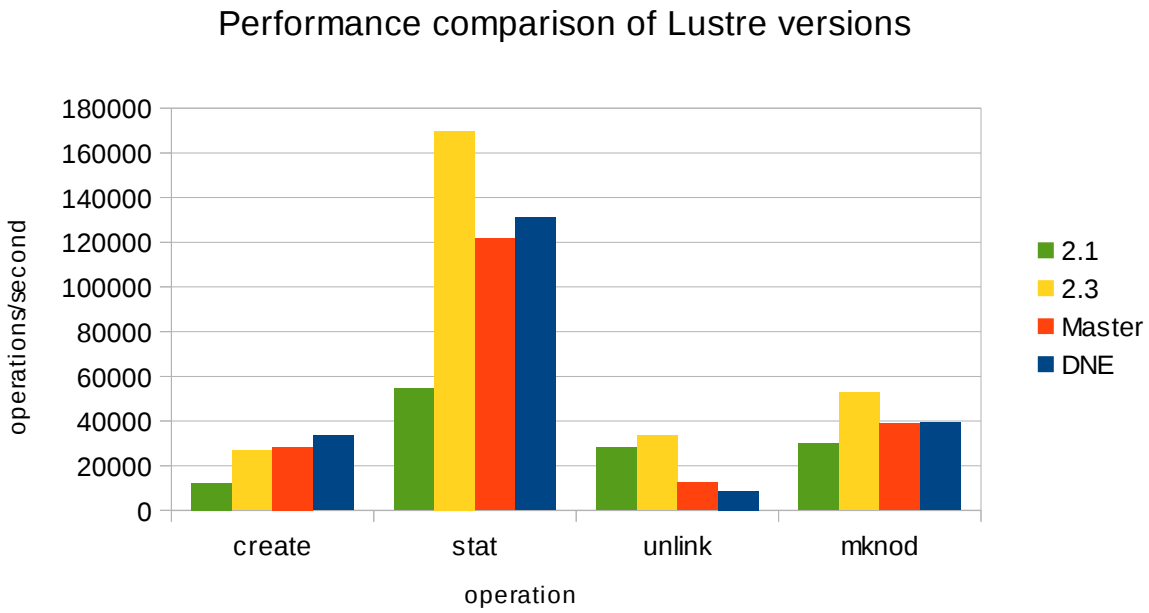


Figure 2: Performance tests on OpenSFS Functional Test Cluster from 2012-12-20.

Results from ongoing analysis are shown in the figure above. These results indicate illustrate that after an initial analysis, fixed build of Master and DNE are able to be rapidly tested and results made available. Performance overall for Master and DNE is improved compared with results from 2012-12-17. Work continues on the analysis of the performance differences which suggest a significant regression.

Scaling of DNE code on OpenSFS Cluster

Scaling test were performed on 2012-12-17 using Master and DNE code from the same date. This code displays poor performance compared to code tested on 2012-12-20, shown on Figure 2.

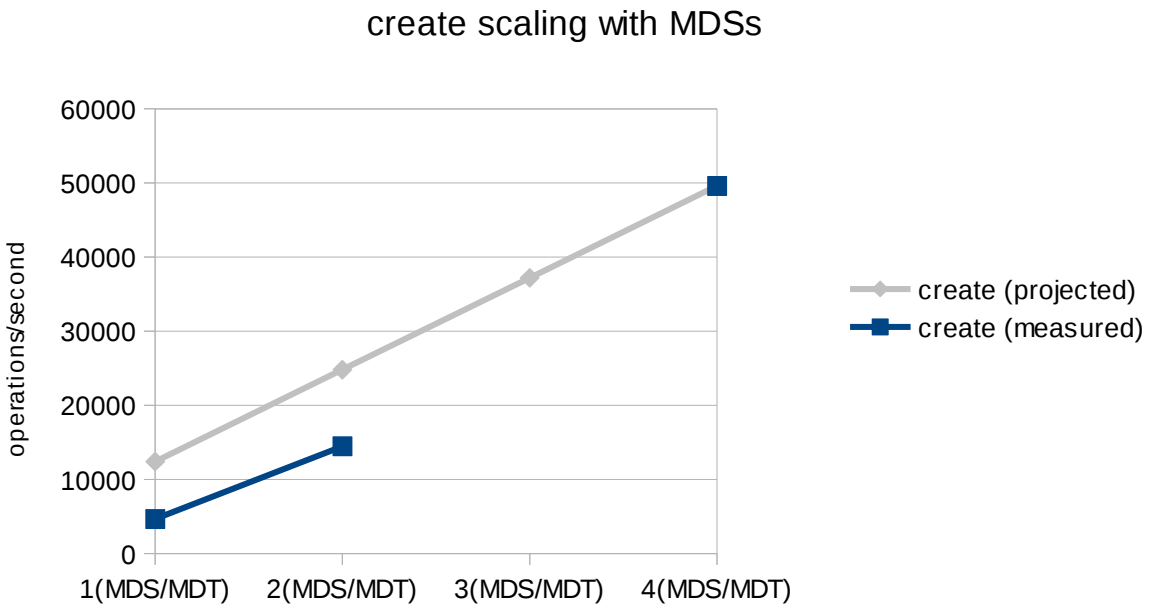
Four tests of typical operations have been completed. These include:

- create
- stat
- unlink
- mknod

Scale tests were run with the 'mdsrate' MPI program on 24 clients on the OpenSFS Functional Test Cluster. Each client has 8 mount points giving a total of 192 threads to drive load to the file system. Each thread operates on 20K files for a given test resulting in a total test load of 3840K files per run. Each test run was

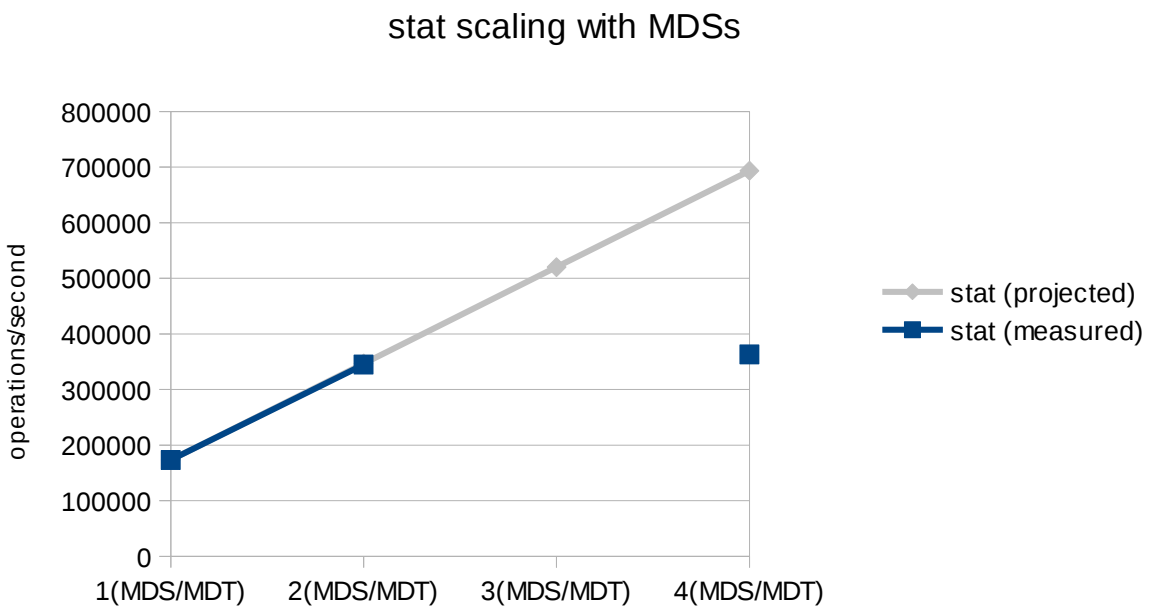
completed once. The OpenSFS Cluster hardware is described in Appendix A. In all cases: projected values are estimated as a linear extrapolation from the best performing MDS/MDT configuration.

create scaling



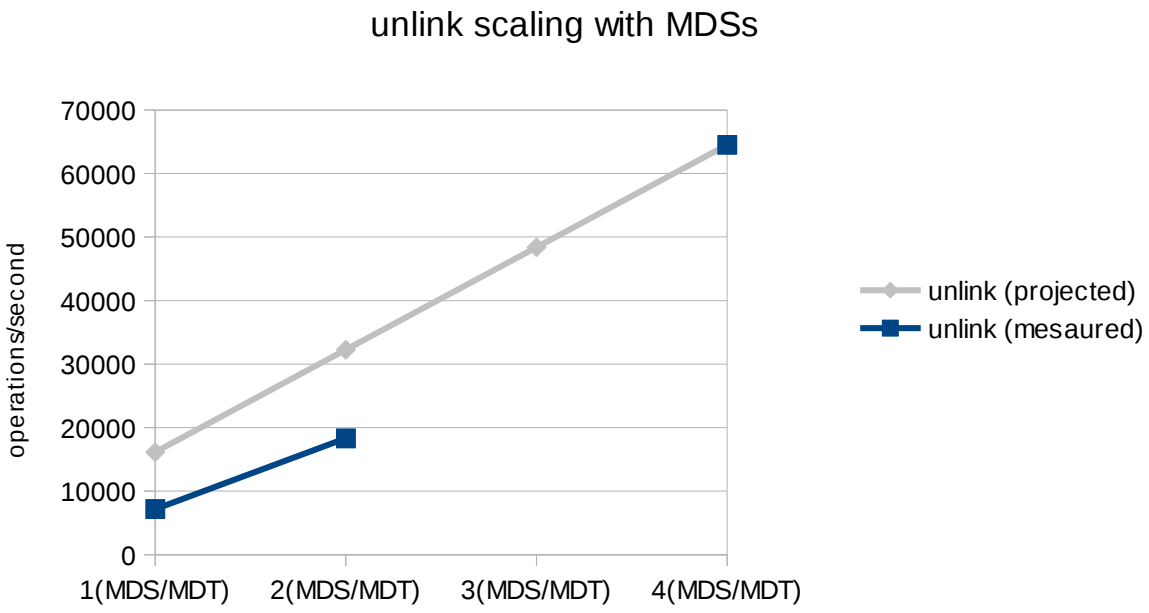
Projected values are calculated by making a linear projection from the best performing configuration. In the case of 'create' 4(MDS/MDT) perform best.

stat scaling



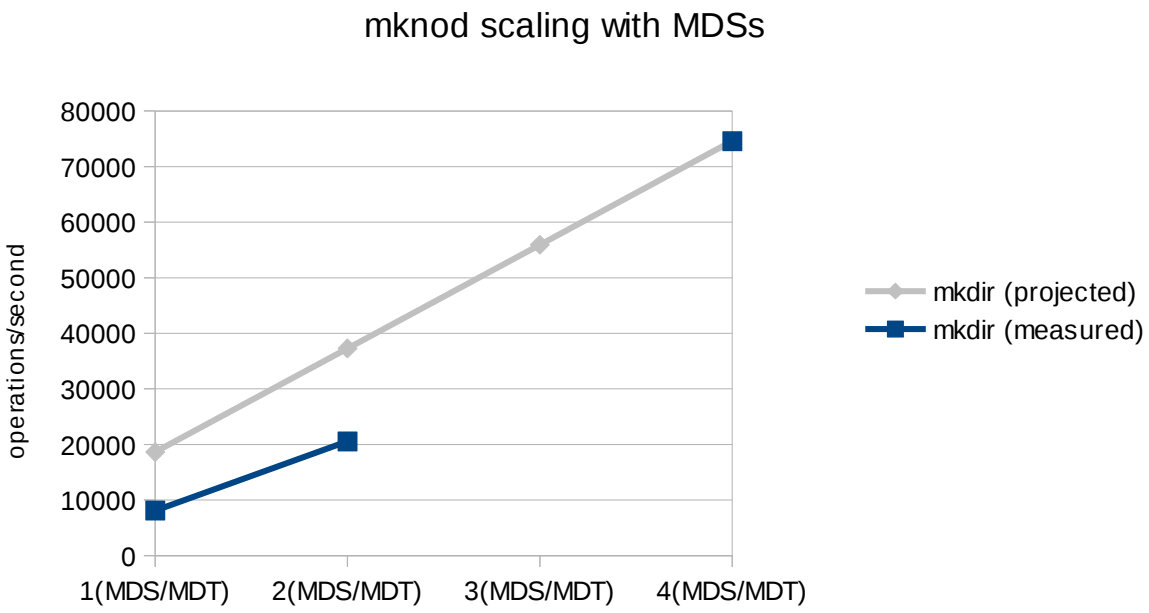
Projected values are calculated by making a linear projection from the best performing configuration. In the case of 'stat', 1(MDS/MDT) performs best.

unlink scaling



Projected values are calculated by making a linear projection from the best performing configuration. In the case of 'unlink' 4(MDS/MDT) perform best.

mknod scaling



Projected values are calculated by making a linear projection from the best performing configuration. In the case of 'mknod' 4(MDS/MDT) perform best.

In the case of the *stat* operation, the scaling between 1 and 2 MDTs increases linearly. An initial analysis suggests that the client load may reach a maximum at ~350K stat/second and be unable to drive the higher order availability. Further performance testing, analysis and improvements will be performed as part of the next milestone.

Create, unlink and mknod illustrate that a single MDT under load performs worse than two MDTs under the same load. An initial analysis suggests that as MDTs are added additional memory and disk cache become available with the new MDS nodes. As the load is constant, fewer inodes per MDS are handled.

Update Lustre Manual

The Lustre manual update is currently in review at:

http://review.whamcloud.com/4773

It includes the following topics:

- add an MDT.
- remove an MDT.
- upgrade to multiple MDT configurations.

- designing active-active MDS configurations.
- warns against having chained remote directories.

The changes from a snapshot of the Manual on 2012-12-17 are presented in Appendix B, below.

Conclusion

Measurements performed on 2012-12-17 indicated Master branch exhibited a performance regression compared to Lustre 2.3, DNE performance on 2012-12-17 was found to be on-par with Master performance on the same date. An initial analysis was conducted. New builds of both Master and DNE were prepared. On 2012-12-20, performance of Master and DNE were repeated. These results show performance of Master and DNE has significantly improved but some measurements (stat, unlink) remain behind 2.3. Once the performance regression in Master is understood and resolved, the DNE performance will be re-tested against the improved Master performance.

The stat operation scales well between one and two MDTs but does not continue to show linear performance scaling up to four MDTs (the only other measurement.) More investigation will be performed for the next milestone, including testing with three MDTs.

Implementation phase 3 has been completed according to the agreed criteria.

Appendix A: OpenSFS Functional Test Cluster configuration and specification

MDS server

- (2) Intel E5620 2.4GHz Westmere (Total 8 Cores)
- (1) 64GB DDRIII 1333MHz ECC/REG - (8x8GB Modules Installed) * (1) On Board Dual 10/100/1000T Ports
- (1) 500GB SATA Enterprises 24x7
- (1) 40GB SSD OCZ SATA
- (8) Hot Swap Drive Bays for SATA/SAS
- (6) PCi-e Slots 8X
- (3) QDR 40GB QSFP to QSFP iB Cables
- (3) Mellanox QDR 40GB QSFP Single Port

each pair of MDS are sharing one storage as below

- (1) Intel E5620 2.4GHz Westmere (Total 4 Cores)
- (1) 12GB DDRIII 1333MHz ECC/REG - (3x4GB Modules Installed)
- (1) On Board Dual 10/100/1000T Ports
- (1) On Board IPMI 2.0 Via 3rd. Lan
- (6) PCi-e Slots 8X
- (2) Mellanox QDR 40GB QSFP Single Port (Connected to MDS Server) * (1) LSI/3Ware 9750SA-4i with BBU installed (raid 0/1/5/6...)
- (1) SM826E16-R920LPB 12 Bays 2U Case with Dual 920W PS,
- (10) 2TB Enterprises HDDs. 24x7 SATA II
- (2) 120GB SSD (Raid 1)

The MDT are configured with external journal from a local SSD with size = 7GB

OSS server

- (2) Intel E5620 2.4GHz Westmere (Total 8 Cores)
- (2) Copper Base CP0217 CPU Cooler 1U with Heat Pipe
- (1) 64GB DDRIII 1333MHz ECC/REG - (8x8GB Modules Installed) * (1) On Board Dual 10/100/1000T Ports
- (1) On Board VGA

- (1) On Board IPMI 2.0 Via 3rd. Lan
- (1) 500GB SATA Enterprises 24x7
- (1) 40GB SSD OCZ SATA
- (8) Hot Swap Drive Bays for SATA/SAS
- (6) PCi-e Slots 8X
- (3) QDR 40GB QSFP to QSFP iB Cables
- (3) Mellanox QDR 40GB QSFP Single Port

each pair of OSS are sharing one storage as below

- (1) Intel E5620 2.4GHz Westmere (Total 4 Cores)
- (1) Copper Base CP0217 CPU Cooler 1U with Heat Pipe
- (1) 12GB DDRIII 1333MHz ECC/REG - (3x4GB Modules Installed)
- (1) On Board Dual 10/100/1000T Ports
- (6) PCi-e Slots 8X
- (2) Mellanox QDR 40GB QSFP Single Port (Connected to OSS Server) * (1) LSI/3Ware 9750SA-4i with BBU installed (raid 0/1/5/6...)
- (1) SM846E16-R1200B 24 Bays 4U Case with Dual 1200W PS,
- (20) 2TB Enterprises HDDs. 24x7 SATA II (2 x 8+2 Raid 6)
- (4) 120GB SSD (Raid 1, 2+2)

ALL OS version are Rhel 6.3/x86_64

DNE: http://review.whamcloud.com/4414

Master: http://review.whamcloud.com/4614

Test CMD:

```
/usr/lib64/openmpi/bin/mpirun -mca plm_ssh_agent rsh -np 192 -machinefile
/home/minh.diep/bin/machinefile /home/minh.diep/bin/mdsrate --create|stat|
unlink|mknod --mdtcount ## --mntcount 8 --mntfmt='/mnt/lustre%d'
--dirfmt='dnedir%d' --nfiles 20000 --ndirs 192 --filefmt 'g%%d'
```

Appendix B: Lustre Manual changes for DNE as of 2012-12-17

The up-to-date changes for DNE are available at:

http://review.whamcloud.com/4773

Table 1.1. Lustre Scalability and Performance

Feature	Current Practical Range	Tested in Production
Client Scalability	100-100000	50000+ clients, many in the 10000 to 20000 range
Client Performance	*Single client:* I/O 90% of network bandwidth *Aggregate:* 2.5 TB/sec I/O	*Single client:* 2 GB/sec I/O, 1000 metadata ops/sec *Aggregate:* 240 GB/sec I/O
OSS Scalability	*Single OSS:* 1-32 OSTs per OSS, 128TB per OST *OSS count:* 500 OSSs, with up to 4000 OSTs	*Single OSS:* 8 OSTs per OSS, 16TB per OST *OSS count:* 450 OSSs with 1000 4TB OSTs 192 OSSs with 1344 8TB OSTs
OSS Performance	*Single OSS:* 5 GB/sec *Aggregate:* 2.5 TB/sec	*Single OSS:* 2.0+ GB/sec *Aggregate:* 240 GB/sec
MDS Scalability	*Single MDS:* 4 billion files *MDS count:* 1 primary + 1 backup Since Lustre 2.4: up to 4096 MDSs and up to 4096 MDTs.	*Single MDS:* 750 million files *MDS count:* 1 primary + 1 backup
MDS Performance	35000/s create operations, 100000/s metadata stat operations	15000/s create operations, 35000/s metadata stat operations
Filesystem Scalability	*Single File:* 2.5 PB max file size *Aggregate:* 512 PB space, 4 billion files	*Single File:* multi-TB max file size *Aggregate:* 10 PB space, 750 million files

Other Lustre features are:

- **Performance-enhanced ext4 file system**: Lustre uses an improved version of the ext4 journaling file system to store data and metadata. This version, called `ldiskfs`, has been enhanced to improve performance and provide additional functionality needed by Lustre.
- **POSIX compliance**: The full POSIX test suite passes in an identical manner to a local ext4 filesystem, with limited exceptions on Lustre clients. In a cluster, most operations are atomic so that clients never see stale data or metadata. Lustre supports mmap() file I/O.
- **High-performance heterogeneous networking**: Lustre supports a variety of high performance, low latency networks and permits Remote Direct Memory Access (RDMA) for Infiniband (OFED) and other advanced networks for fast and efficient network transport. Multiple RDMA networks can be bridged using Lustre routing for maximum performance. Lustre also provides integrated network diagnostics.
- **High-availability**: Lustre offers active/active failover using shared storage partitions for OSS targets (OSTs). Lustre 2.3 and earlier offers active/passive failover using a shared storage partition for the MDS target (MDT).

  With Lustre 2.4 or later servers and clients it is possible to configure active/active failover of multiple MDTs

  This allows application transparent recovery. Lustre can work with a variety of high availability (HA) managers to allow automated failover and has no single point of failure (NSPF). Multiple mount protection (MMP) provides integrated protection from errors in highly-available systems that would otherwise cause file system corruption.
- **Security**: By default TCP connections are only allowed from privileged ports. Unix group membership is verified on the MDS.
- **Access control list (ACL), extended attributes**: the Lustre security model follows that of a UNIX file system, enhanced with POSIX ACLs. Noteworthy additional features include root squash.
- **Interoperability**: Lustre runs on a variety of CPU architectures and mixed-endian clusters and is interoperable between successive major Lustre software releases.
- **Object-based architecture**: Clients are isolated from the on-disk file structure enabling upgrading of the storage architecture without affecting the client.
- **Byte-granular file and fine-grained metadata locking**: Many clients can read and modify the same file or directory concurrently. The Lustre distributed lock manager (LDLM) ensures that files are coherent between all clients and servers in the filesystem. The MDT LDLM manages locks on inode permissions and pathnames. Each OST has its own LDLM for locks on file stripes stored thereon, which scales the locking performance as the filesystem grows.
- **Quotas**: User and group quotas are available for Lustre.
- **Capacity growth**: The size of a Lustre file system and aggregate cluster bandwidth can be increased without interruption by adding a new OSS with OSTs to the cluster.
- **Controlled striping**: The layout of files across OSTs can be configured on a per file, per directory, or per file system basis. This allows file I/O to be tuned to specific application requirements within a single filesystem. Lustre uses RAID-0 striping and balances space usage across OSTs.

1.2.2. Lustre File System Components

Each Lustre file system consists of the following components:

- **Metadata Server (MDS)** - The MDS makes metadata stored in one or more MDTs available to Lustre clients. Each MDS manages the names and directories in the Lustre file system(s) and provides network request handling for one or more local MDTs.
- **Metadata Target (MDT**) - The MDT stores metadata (such as filenames, directories, permissions and file layout) on storage attached to an MDS. Each file system has one MDT. An MDT on a shared storage target can be available to multiple MDSs, although only one can access it at a time. If an active MDS fails, a standby MDS can serve the MDT and make it available to clients. This is referred to as MDS failover.

  Since Lustre 2.4, multiple MDTs are supported. Each filesystem has at least one MDT. An MDT on shared sotrage target can be available via multiple MDSs, although only one MDS can export the MDT to the clients at one time. Two MDS machines share storage for two MDTs. After the failure of one MDS, the remaining MDS begins serving the MDT of the failed MDS.
- **Object Storage Servers (OSS)** : The OSS provides file I/O service, and network request handling for one or more local OSTs. Typically, an OSS serves between 2 and 8 OSTs, up to 16 TB each. A typical configuration is an MDT on a dedicated node, two or more OSTs on each OSS node, and a client on each of a large number of compute nodes.
- **Object Storage Target (OST)** : User file data is stored in one or more objects, each object on a separate OST in a Lustre file system. The number of objects per file is configurable by the user and can be tuned to optimize performance for a given workload.
- **Lustre clients** : Lustre clients are computational, visualization or desktop nodes that are running Lustre client software, allowing them to mount the Lustre file system.

The Lustre client software provides an interface between the Linux virtual file system and the Lustre servers. The client software includes a Management Client (MGC), a Metadata Client (MDC), and multiple Object Storage Clients (OSCs), one corresponding to each OST in the file system.

A logical object volume (LOV) aggregates the OSCs to provide transparent access across all the OSTs. Thus, a client with the Lustre file system mounted sees a single, coherent, synchronized namespace. Several clients can write to different parts of the same file simultaneously, while, at the same time, other clients can read from the file.

Table 1.2, "Storage and hardware requirements for Lustre components" provides the requirements for attached storage for each Lustre file system component and describes desirable characteristics of the hardware used.

3.1.2. Types of Failover Configurations

Nodes in a cluster can be configured for failover in several ways. They are often configured in pairs (for example, two OSTs attached to a shared storage device), but other failover configurations are also possible. Failover configurations include:

- **Active/passive** pair - In this configuration, the active node provides resources and serves data, while the passive node is usually standing by idle. If the active node fails, the passive node takes over and becomes active.
- **Active/active** pair - In this configuration, both nodes are active, each providing a subset of resources. In case of a failure, the second node takes over resources from the failed node.

Typically, Lustre MDSs are configured as an active/passive pair, while OSSs are deployed in an active/active configuration that provides redundancy without extra overhead. Often the standby MDS is the active MDS for another Lustre file system or the MGS, so no nodes are idle in the cluster.

Lustre 2.4 introduces metadata targets for individual sub-directories. Active-active failover configurations are available for MDSs that serve MDTs on shared storage.

3.2. Failover Functionality in Lustre

The failover functionality provided in Lustre can be used for the following failover scenario. When a client attempts to do I/O to a failed Lustre target, it continues to try until it receives an answer from any of the configured failover nodes for the Lustre target. A user-space application does not detect anything unusual, except that the I/O may take longer to complete.

Lustre failover requires two nodes configured as a failover pair, which must share one or more storage devices. Lustre can be configured to provide MDT or OST failover.

- For MDT failover, two MDSs are configured to serve the same MDT. Only one MDS node can serve an MDT at a time.

  Lustre 2.4 allows multiple MDTs. By placing two MDT partitions on storage shared by two MDSs, one MDS can fail and the remaining MDS can begin serving the unserved MDT. This is described as an active/active failover pair.

- For OST failover, multiple OSS nodes are configured to be able to serve the same OST. However, only one OSS node can serve the OST at a time. An OST can be moved between OSS nodes that have access to the same storage device using `umount/mount` commands.

To add a failover partner to a Lustre configuration, the `--failnode` or `--servicenode` option is used. This can be done at creation time (using `mkfs.lustre`) or later when the Lustre system is active (using `tunefs.lustre`). For explanations of these utilities, see Section 36.14, " mkfs.lustre" and Section 36.18, " tunefs.lustre".

Lustre failover capability can be used to upgrade the Lustre software between successive minor versions without cluster downtime. For more information, see Chapter 16, *Upgrading Lustre*.

For information about configuring failover, see Chapter 11, *Configuring Lustre Failover*.

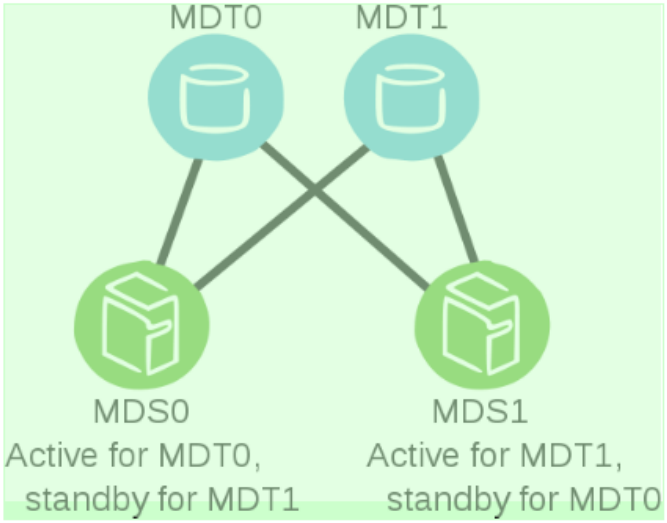
> **Note**
>
> Failover functionality in Lustre is provided only at the file system level. In a complete failover solution, failover functionality for system-level components, such as node failure detection or power control, must be provided by a third-party tool.

> **Caution**
>
> OST failover functionality does not protect against corruption caused by a disk failure. If the storage media (i.e., physical disk) used for an OST fails, Lustre cannot recover it. We strongly recommend that some form of RAID be used for OSTs. Lustre functionality assumes that the storage is reliable, so it adds no extra reliability features.

3.2.2. MDT Failover Configuration (Active/Active)

Multiple MDTs became available with the advent of Lustre 2.4. MDTs can be configured in a load-balanced, active/active failover configuration. A failover cluster is built from two MDSs as shown in Figure 3.2, " Lustre failover configuration for a active/active MDTs ".

Figure 3.2. Lustre failover configuration for a active/active MDTs



5.1.1. MDT Storage Hardware Considerations

The data access pattern for MDS storage is a database-like access pattern with many seeks and read-and-writes of small amounts of data. High throughput to MDS storage is not important. Storage types that provide much lower seek times, such as high-RPM SAS or SSD drives can be used for the MDT.

For maximum performance, the MDT should be configured as RAID1 with an internal journal and two disks from different controllers.

If you need a larger MDT, create multiple RAID1 devices from pairs of disks, and then make a RAID0 array of the RAID1 devices. This ensures maximum reliability because multiple disk failures only have a small chance of hitting both disks in the same RAID1 device.

Doing the opposite (RAID1 of a pair of RAID0 devices) has a 50% chance that even two disk failures can cause the loss of the whole MDT device. The first failure disables an entire half of the mirror and the second failure has a 50% chance of disabling the remaining mirror.

If multiple MDTs are going to be present in the system, each MDT should be specified for the anticipated usage and load.

Warning

MDT0 contains the root of the Lustre filesystem. If MDT0 is unavailable for any reason, the filesystem cannot be used.

Note

Additional MDTs can be dedicated to sub directories off the root filesystem provided by MDT0. Subsequent directories may also be configured to have their own MDT. If a MDT serving a subdirectory becomes unavailable this subdirectory and all directories beneath it will also become available. Configuring multiple levels of MDTs is an experimental feature for Lustre 2.4.

To configure a simple Lustre file system, complete these steps:

1. Create a combined MGS/MDT file system on a block device. On the MDS node, run:

   ```
   mkfs.lustre --fsname=<fsname> --mgs --mdt --index=0 <block device name>
   ```

   The default file system name (`fsname`) is `lustre`.

   > **Note**
   >
   > If you plan to create multiple file systems, the MGS should be created separately on its own dedicated block device, by running:
   >
   > ```
   > mkfs.lustre --fsname=<fsname> --mgs <block device name>
   > ```
   >
   > See Section 13.7, "Running Multiple Lustre File Systems" for more details.

2. Optional for Lustre 2.4. Add in additional MDTs.

   ```
   mkfs.lustre --fsname=<fsname> --mgs --mdt --index=1 <block device name>
   ```

   > **Note**
   >
   > Up to 4095 additional MDTs can be added.

3. Mount the combined MGS/MDT file system on the block device. On the MDS node, run:

   ```
   mount -t lustre <block device name> <mount point>
   ```

   > **Note**
   >
   > If you have created and MGS and an MDT on separate block devices, mount them both.

4. Create the OST. On the OSS node, run:

   ```
   mkfs.lustre --fsname=<fsname> --mgsnode=<NID> --ost --index=<OST index> <block device name>
   ```

   When you create an OST, you are formatting a `ldiskfs` file system on a block storage device like you would with any local file system.

13.8. Creating a sub-directory on a given MDT

Lustre 2.4 enables individual sub-directories to be serviced by unique MDTs. An administrator can allocate a sub-directory to a given MDT using the command:

```
lfs mkdir –i <mdtindex> <remote_dir>
```

This command will allocate the sub-directory `remote_dir` onto the MDT of index `mdtindex`. For more information on adding additional MDTs and `mdtindex` see 2.

Warning

An administrator can allocate remote sub-directories to separate MDTs. Creating remote sub-directories in parent directories not hosted on MDT0 is not recommended. This is because the failure of the parent MDT will leave the namespace below it inaccessible. For this reason, by default it is only possible to create remote sub-directories off MDT0. To relax this restriction and enable remote sub-directories off any MDT, an administrator must issue the command `lctl set_param mdd.*.enable_remote_dir=1`.

14.6. Adding a new MDT to a Lustre file system

MDTs can be added to to serve individual remote sub-directories within the filesystem. However, the root directory will always be located on MDT0. To add a new MDT into the file system:

1. Discover the maximum MDT index. Each MDTs must have unique index.

```
client$ lctl df | grep mdc
36 UP mdc lustre-MDT0000-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063-8477566eb84e 5
37 UP mdc lustre-MDT0001-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063-8477566eb84e 5
38 UP mdc lustre-MDT0002-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063-8477566eb84e 5
39 UP mdc lustre-MDT0003-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063-8477566eb84e 5
```

2. Add the new block device as a new MDT at the next available index. In this example, the next available index is 4.

```
mkfs.lustre --reformat --mdt --mgsnode=<mgsnode> --index 4 <blockdevice>
```

3. Mount the MDTs.

```
mount –t lustre <blockdevice> /mnt/mdt4
```

8.1. Removing a MDT from the File System

If a sub-directory is being served by an individual MDT, it is possible to remove the MDT from the file system. The first step is to move all the files out off the sub-directory served by the MDT. Once the sub-directory is empty, an administrator can remove the MDT using the command issued from MDT0:

```
lfs rmdir <remote_dir>
```

This command will remove the sub-directory `remove_dir` from the namespace.

> **Warning**
>
> Once removed, a MDT serving a sub-directory must be reformatted before it is added back.

14.8.2. Working with Inactive MDTs

Files located on or below an inactive MDT are inaccessible until the MDT is activated again. Clients accessing an inactive MDT will receive an EIO error.

16.1. Lustre Interoperability

Lustre 2.x is built on a new architectural code base which is different than the one used with Lustre 1.8. These architectural changes require existing Lustre 1.8 users to follow a slightly different procedure to upgrade to Lustre 2.x - requiring clients to be unmounted and the file system be shut down. Once the servers are upgraded and restarted, then the clients can be remounted. After the upgrade, Lustre 2.x servers can interoperate with compatible 1.8 clients and servers. Lustre 2.x does *not* support 2.x clients interoperating with 1.8 servers.

> **Note**
>
> Lustre 1.8 clients can interoperate with 2.x servers, but the servers should all be upgraded at the same time.

> **Note**
>
> Lustre 2.x servers are compatible with clients 1.8.6 and later, though it is strongly recommended that the clients are upgraded to the latest version of Lustre 1.8 available. If you are planning a heterogeneous environment (mixed 1.8 and 2.x servers), make sure that version 1.8.6 or later is installed on the client nodes that are not upgraded to 2.x.

> **Warning**
>
> Lustre 2.4 allows remote sub-directories to have individual MDTs. Clients prior to 2.4 only have the namespace from MDT0 visible to them.

16.3. Upgrading to multiple metadata targets

Lustre 2.4 allows separate metadata servers to serve separate sub directories. To upgrade a filesystem to Lustre 2.4 that support multiple metadata servers:

1. Stop MGT/MDT/OST and upgrade to 2.4
2. Reformat new MDT according to Section 14.6, "Adding a new MDT to a Lustre file system".
3. Mount all of the targets according to Section 14.6, "Adding a new MDT to a Lustre file system".
4. After recovery is completed clients will be connected MDT0.

> **Note**
>
> Clients prior to 2.4 will only be have the namespace provide by MDT0 visable.

30.1.3. MDS Failure (Failover)

Highly-available (HA) Lustre operation requires that the metadata server have a peer configured for failover, including the use of a shared storage device for the MDT backing file system. The actual mechanism for detecting peer failure, power off (STONITH) of the failed peer (to prevent it from continuing to modify the shared disk), and takeover of the Lustre MDS service on the backup node depends on external HA software such as Heartbeat. It is also possible to have MDS recovery with a single MDS node. In this case, recovery will take as long as is needed for the single MDS to be restarted.

When Section 30.6, "Imperative Recovery" is enabled, clients are notified of an MDS restart (either the backup or a restored primary). Clients always may detect an MDS failure either by timeouts of in-flight requests or idle-time ping messages. In either case the clients then connect to the new backup MDS and use the Metadata Replay protocol. Metadata Replay is responsible for ensuring that the backup MDS re-acquires state resulting from transactions whose effects were made visible to clients, but which were not committed to the disk.

The reconnection to a new (or restarted) MDS is managed by the file system configuration loaded by the client when the file system is first mounted. If a failover MDS has been configured (using the `--failnode=` option to `mkfs.lustre` or `tunefs.lustre`), the client tries to reconnect to both the primary and backup MDS until one of them responds that the failed MDT is again available. At that point, the client begins recovery. For more information, see Section 30.2, "Metadata Replay".

Transaction numbers are used to ensure that operations are replayed in the order they were originally performed, so that they are guaranteed to succeed and present the same filesystem state as before the failure. In addition, clients inform the new server of their existing lock state (including locks that have not yet been granted). All metadata and lock replay must complete before new, non-recovery operations are permitted. In addition, only clients that were connected at the time of MDS failure are permitted to reconnect during the recovery window, to avoid the introduction of state changes that might conflict with what is being replayed by previously-connected clients.

Lustre 2.4 introduces multiple metadata targets. If multiple metadata targets are in use, active-active failover is possible. See Section 3.2.2, "MDT Failover Configuration (Active/Active)" for more information.

MDT0

The metadata target for the file system root. Since Lustre 2.4, multiple metadata targets are possible in the same filesystem. MDT0 is the root of the filesystem which must be available for the filesystem to be accessible.