



Remote Directories Solution Architecture

 This document has been submitted for acceptance on 2011-12-09. This document was accepted on 2011-12-23

Introduction

 Unknown macro: 'html'

Today's HPC systems routinely configure Lustre with tens of thousands of clients. Each client draws resources from a single metadata server (MDS). The efforts of the Lustre community continue to improve the performance of the MDS, however, as client numbers continue to increase the single MDS represents a fundamental limit to filesystem scalability.

The goal of the Distributed Namespace (DNE) project is to deliver a documented and tested implementation of Lustre that addresses this scaling limit by distributing the filesystem metadata over multiple metadata servers. This is an ambitious engineering project that will take place over a period of two years. It requires considerable engineering and testing resource and will therefore be performed in the two phases described below.

Phase 1: Remote Directories

This phase introduces a useful minimum of distributed metadata functionality. The purpose primarily to ensure that efforts concentrate on clean code restructuring for DNE. The phase focuses on extensive testing to shake out bugs and oversights not only in the implementation but also in administrative procedures. DNE brings new usage patterns that must necessarily adapt to manage multiple metadata servers.

The Lustre namespace will be distributed by allowing directory entries to reference sub-directories on different metadata targets (MDTs). Individual directories will remain bound to a single MDT, therefore metadata throughput on single directories will stay limited by single MDS performance, but metadata throughput aggregated over distributed directories will scale.

The creation of non-local subdirectories will initially be restricted to administrators with a Lustre-specific `mkdir` command. This will ensure that administrators retain control over namespace distribution to guarantee performance isolation.

Phase 2: Striped Directories

This phase will introduce sharded directories in which directory entries in a single directory are hashed over multiple MDTs. This will allow metadata throughput within a single directory to scale.

This proposal describes the first phase: Remote Directories. A separate document will describe the Striped Directories phase.

Requirements

Performance

Scalability

DNE must provide the ability to distribute a single Lustre filesystem over multiple MDTs under administrator control.

Metadata throughput aggregated over multiple MDSs should scale as MDSs are added until OSSs reach saturation.

The throughput of metadata operations against any single directory will be limited by the performance of a single MDS during Phase 1. This restriction will be addressed during Phase 2 when a single directory may be sharded over multiple MDTs.



Single MDS performance

Single MDS performance is addressed in a separate OpenSFS project: [Single Server Metadata Performance](#)

Isolation

DNE must demonstrate performance isolation between metadata workloads on different subtrees of the namespace located on different MDSs, consistent with the extent to which these subtrees contend for the same OSTs.

Cross-MDT Operations

Non-local rename and hardlink

Rename and hardlink operations on inodes and directory entries all on the same MDT will function as normal. Otherwise these operations will be a no-op and return EXDEV, as if the caller attempted to use them on different filesystems. This will force utilities to try alternatives - e.g. "mv" will copy the source to the target and remove the source on success.

Non-local mkdir & rmdir

The administrative command to create a directory on a non-local MDT (i.e. where the child directory resides on a different MDT from its parent), and rmdir run on such directories will update state on both child and parent MDTs. Node failures while these operations are being performed must ensure that the child directory remains accessible and intact while the parent directory reference exists. This will require sequenced synchronous writes to persistent storage which will make these operations relatively slow. All other metadata modifying operations will execute as normal with no performance regressions.

Compatibility

All metadata operations on a single-MDT Lustre filesystem must also work in a Remote Directories environment. When adding a new MDT to a single-MDT filesystem, the contents of the existing MDT must not be affected.

Upgrading to DNE

A pre-DNE Lustre 2.x filesystem must be upgradable to a Lustre version that supports DNE.

Adding an MDT

Within the constraints of this initial development, addition of MDTs must be performed off-line. However, the implementation will allow online MDT addition as more experience is gained with its usage.

Disabling an MDT

An administrator will be able to disable any MDT. Any file operations that access a disabled MDT will receive an IO

error. Disabling should only be performed if an MDT is to be made permanently unavailable. If MDT0 (the primary MDT) is disabled the entire filesystem will be inaccessible.

Removing a MDT

Removing a MDT will not be supported.

Resilience

Operations affecting multiple MDTs

The majority of metadata operations will only affect a single MDT. Their implementation and performance will not be changed. Operations that span multiple MDTs will be subject to additional constraints as described above to prevent "dangling" links (i.e. a directory entry for which there is no inode) under all possible node failures or combinations of node failures. It is permissible for such failures to leave an orphan inode which can be cleaned up by a filesystem scan.

Remote directories

The administrative command to create non-local sub-directories will initially only allow this from parent directories on MDT0. This is to ensure that the failure of any MDT other than MDT0 (the primary MDT) does not disconnect the namespace. For experimental purposes, an administrative override will be available that allows specifying non-MDT0 as the parent directory. Normal Lustre recovery should be unaffected.

MDT Failover

Multiple MDTs sharing a single MDS must be supported so that MDSs can be configured for active-active failover just like OSSs.

Solution Proposal

Distributed Metadata Operations

Operations that require updates on multiple MDTs will be executed using a master/slave model. The client will send the request to the master MDT (e.g. the MDT holding the parent directory for the operation) which will then distribute updates to the other slave MDT(s). These updates will be ordered and partially synchronous to ensure that the name space remains intact in the face of all possible failures. In the case of multi-node failures at worst a single reference on the directory will be leaked. This may lead to an empty directory not being deleted, but the namespace will remain usable after recovery. The cleanup of such empty directories will occur in Phase III of the Lfsck project. In DNE Phase I, only remote directory creation and destroy are cross-MDT operations.

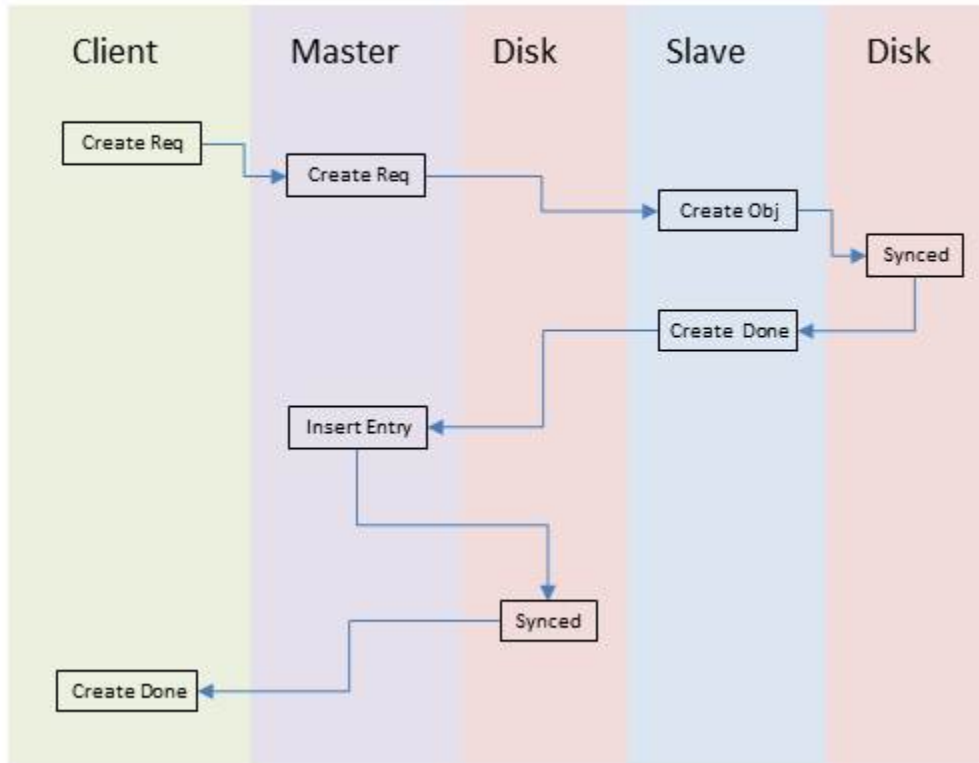
Remote directory creation

During remote directory creation,

1. An administrator calling `lfs mkdir` must specify which MDT the directory will be created on (this is the slave MDT).
2. The client will allocate a fid on the slave MDT for the remote directory and send a request to the master MDT to create the directory.
3. The master MDT will send the create request (along with the fid) to the slave MDT where the remote object resides.

4. The slave MDT will create the object from the fid synchronously and reply to the master MDT.
5. The master MDT will insert the name entry synchronously. By using a synchronous insert, the risk of files and directories being disconnected from the namespace can be avoided for a number of complicated recovery scenarios.

Remote Create



Remote create recovery

Given that the entire process is synchronous, replay will not occur during recovery. However, resend may occur under two conditions:

Resend between the Master MDT and client

If the master MDT restarts before it sends the reply to the client, the client will resend the request to the master MDT. Since it is the client that allocated the FID for the remote directory, it is guaranteed that the same FID will be used during the recovery.

After the master MDT receives the request, the existence of the name entry will be verified. If the entry exists, the master MDT will send a reply to the client with the existing directory FID. If the name does not exist the master MDT will redo the entire operation.

During this process, the slave MDT will verify whether the operation has been executed. This can be resolved by adding the client->master request XID to the `last_rcvd` file for that client. When the master MDT sends its RPC to the slave MDT, the XID of the client->master request will be included in the slave RPC, and the slave MDT will add the XID to that MDT's export in the `last_rcvd` file. That will allow the slave MDT to determine whether the request from the master MDT was processed during recovery.

Resend between the Master MDT and the slave MDT

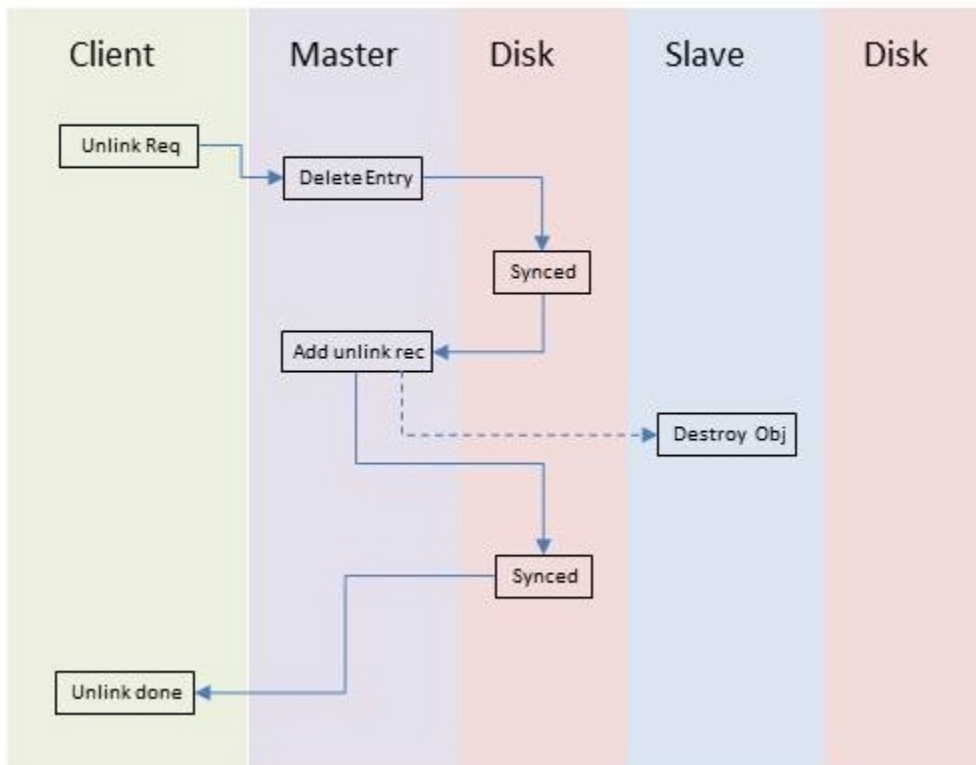
If the slave MDT restarts before it sends reply to the master MDT, the master MDT will resend the request to the slave MDT when it reconnects. This will recreate the object if it does not exist.

Remote rmdir

During remote directory unlink,

1. The client will send the unlink request to the master MDT, where the parent resides.
2. The master MDT will delete the name entry and add an unlink request to a local log in a single synchronous disk transaction.
3. The master MDT may then reply to the client.
4. A log sync thread will send an unlink request to the slave MDT to destroy the (now orphaned) directory inode. This request will be resent until it completes normally to confirm that the directory inode has been destroyed. The local log record will then be canceled.

Remote Unlink



⚠ Remote unlink

Remote unlink is similar to OST orphan handling.

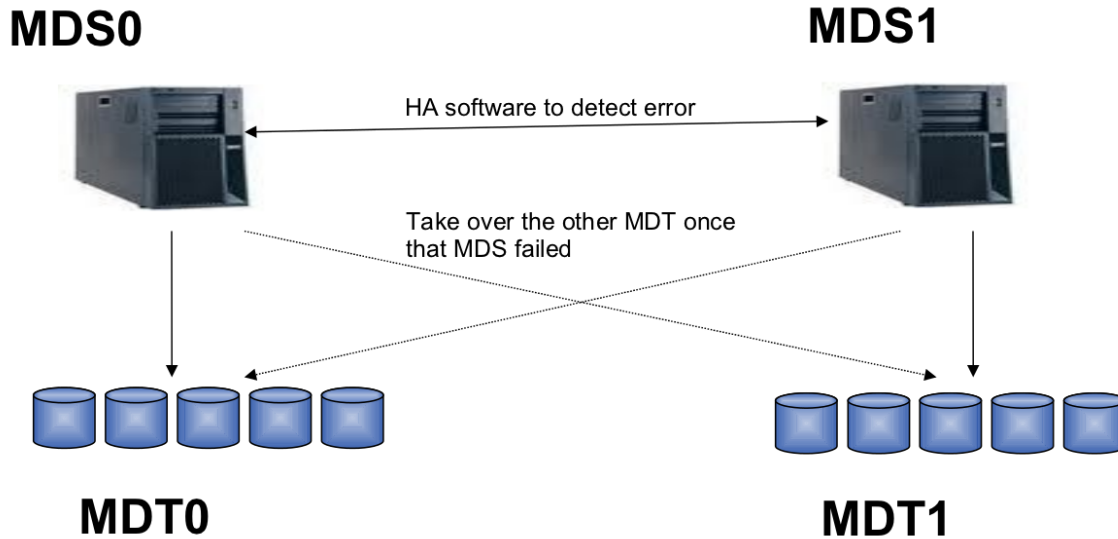
Remote unlink recovery

If the master MDT restarts before it sends the reply to the client, the client will resend the request to the master MDT. The entire operation will be repeated if the entry still exists, otherwise the master MDT will do nothing but reply to the client as if the operation has been processed correctly.

If the slave MDT restarts during the process, the log sync thread will ensure the remote directory will be finally destroyed on the slave MDT.

MDT Failover

With DNE, MDS nodes may be configured as active-active failover pairs using shared storage. If one MDS in the pair fails, the other MDS will take over the failed MDS's MDT to maintain the accessibility of the complete namespace. Such an active-active failover configuration is illustrated below:



The notification and failover mechanism is provided by an external High Availability (HA) daemon that monitors the status of the MDS nodes. This HA daemon implements a mechanism to power off (STONITH) a failing MDS, and to import the shared storage to the backup server.

Unit/Integration test plan

Normal (Single-MDT) operations

Functional testing will create a directory on every MDT and run the full set of standard non-failover tests within each directory concurrently and from multiple clients. All test must pass with the same behavior observed with a single MDT.

Cross-MDT operations

EXDEV tests

Create two directories on all MDTs except MDT0. Populate the directories with subtrees using depth and file counts randomly chosen from within specified limits. Run multiple threads on multiple clients, each randomly picking a pair of directories from all the subtrees, creating a file in one directory and attempting to rename(2) or link(2) between directories. These operations must fail with EXDEV if the directories chosen were on different MDTs and succeed if they were on the same MDT.

Soak tests

Create a directory on every MDT and run racer and lfs mkdir under these directories.

1. Setup the file system with four MDTs and four OSTs.
2. Create one directory on each MDT.

3. Run Lustre `racer` test under each directory.
4. At the same time, do `lfs mkdir --i $((RANDOM % MDTCOUNT)) $((RANDOM % MAX))` under the directory on MDT0 and no errors will be observed.

Remote mkdir

A remote directory must inherit default ACL and other necessary attributes of the parent directory correctly.

1. `mkdir /mnt/lustre/parent` on MDT0.
2. Set default ACL on `/mnt/lustre/parent`.
3. `lfs mkdir -i 1 /mnt/lustre/parent/dir_mdt1` # create a cross-ref dir on MDT1
4. Verify the default ACL of `/mnt/lustre/parent/dir_mdt1` is the same as the one in `/mnt/lustre/parent`.

rmdir remote directory

1. `rmdir /mnt/lustre/parent/dir_mdt1` # unlink a cross-ref dir on MDT1.
2. Verify entry `dir_mdt1` has been deleted on MDT0.
3. Verify `dir_mdt1` has been destroyed on MDT1 by checking whether `dir_mdt1` exists before and after `rmdir` on MDT1.

Recovery test

Normal Recovery tests

Normal recovery is defined as recovering to a functional filesystem after client or server failure.

1. Setup the filesystem with four MDSs (one MDT on each MDS) and four OSSs (two OSTs on each OSS), eight clients.
2. Each client will create its own directory by `lfs mkdir` (used for creating cross-ref directory in DNE) on one MDT, so that each MDT will have two directories.
3. Each client will run a test on its own directory. (bonnie++, dbench, or other benchmark to be advised). Note: The client whose directory resides on MDT0 only does `lfs mkdir` and `rmdir` during the test.
4. After a random number minutes with an agreed minimum, one node (MDS, OSS or client) will be rebooted, and the services will be restarted (restart tests if it is client).
 - a. If MDS or OSS is failed, the test on each client should still keep running and not return any error.
 - b. If the client is failed, the test program on other clients should still keep running and not return any error.
5. Wait until recovery is complete and repeat step 4.

Cross-MDT updates Recovery tests

Restart the slave MDT during cross-MDT updates,

1. `lfs mkdir` to create a remote directory. The master MDT will send a synchronous create request to the slave MDT.
2. Restart the slave MDT before or after it creates the directory inode.
3. After the recovery succeeds, the operation must succeed and the namespace must remain consistent.

Restart the Master MDT during cross-MDT updates,

1. `lfs mkdir` to create a remote directory. The master MDT will send synchronous create request to the slave MDT.
2. The slave MDT creates the object synchronously and replies to the master MDT.
3. Restart the master MDT before it inserts the name entry.

4. After recovery completes the operation must succeed and the namespace must remain consistent.

Active-Active Recovery

Demonstrate active-active failover with separate MDTs.

1. Setup active-active failover for 2 MDSs. This setup will be similar to configuring OSS active-active failover.
2. Simultaneously run a suitable, on-going test against both MDTs.
3. Fail one MDS. The request to the failed MDS should be sent to the other MDS.
4. Both tests must continue to run with no errors.

Compatibility test

Upgrade single MDT to DNE,

1. Create a new filesystem with single MDT. The Lustre version should be prior to DNE, but after 2.1.
2. Create files and directories on the filesystem.
3. Shut down the file system (MDS and OSSs).
4. Upgrade MDS and OSSs to DNE and restart the initial filesystem.
5. Non-DNE clients are able to reconnect to MDS and OSSs, files and directories created on step 2 can be accessed.
6. Format 3 additional MDTs with `mkfs.lustre` and add them to the filesystem, which now includes 4 MDTs.
7. The incompatible client is evicted.
8. Verify the files and directories can be accessed and they reside on MDT0.
9. Update client to DNE. Remount the filesystem.
10. Add an additional MDT. Create a directory with files on the newly added MDT.

Performance scaling

Performance tests should show acceptable scaling for workloads operating in different directories hosted on separate MDTs provided OSSs are not saturated.

Open/Create performance

1. Create a file system with n MDTs and $2n$ OSTs.
2. Create n directories. Each directory will reside on different MDTs.
3. Set the default file layout to be single-stripe.
4. Create 1 million files on a single directory. Measure the rate of creation f (files per second)
5. Create 1 million files under all n directories concurrently.
6. Observe the total rate of creation f' . f' must be $\sim 3n$.

Lookup/stat performance

1. Create a file system with 1 MDT and n OSTs.
2. Create n directories the single MDT and populate with 1 million files in each.
3. Umount and remount clients and MDS to clear the client/MDS cache.
4. Do `mdsrute (--lookup --stat)` on each directory parallel (`thread_count = n`) and measure the time taken for the entire process: $t_{lookup+stat}$
5. Umount and reformat the file system, and then remount.
6. Add $n-1$ MDTs to the file system. Repeat steps(1-3) ensuring n directories now reside on different MDTs.
7. Do `mdsrute (--lookup --stat)` on each directory in parallel (`thread_count = n`) and record the time taken: $t'_{lookup+stat}$ Assert $t'_{lookup+stat} < t_{lookup+stat}$

Unlink performance

1. Create a file system with 1 MDT and n OSTs.
2. Create n directories the single MDT and populate with 1 million files in each.
3. Umount and remount clients and MDS to clear the client/MDS cache.
4. Do `mdsrage (--unlink)` on each directory parallel (`thread_count = n`) and measure the time taken for the entire process: t_{unlink}
5. Umount and reformat the file system and then remount.
6. Add $n-1$ MDTs to the file system. Repeat steps(1-3) ensuring n directories now reside on different MDTs.
7. Do `mdsrage (--unlink)` on each directory in parallel (`thread_count = n`) and record the time taken: t'_{unlink}
Assert $t'_{unlink} < t_{unlink}$

Acceptance Criteria

1. All unit/integration tests will pass.
2. All performance scaling tests pass.
3. The filesystem is still available when any MDT other than MDT0 is down or disabled.

Glossary

MDS - MetaData Server (the node on which the metadata operations are done)

MDT - MetaData Target (the underlying storage and filesystem in which the metadata resides)

MDT0 - The MDT containing the root of the filesystem.

Master MDT - The MDT that coordinates a distributed MDT operation.

Slave MDT - The MDT controlled by the master MDT in a distributed MDT operation.

XID - The unique ID for a ptrlpc request within a ptrlpc connection.