Milestone Completion for the Striped Directories subproject of the Distributed Namespace Project of the SFS-DEV-001 contract.

Revision History

Date	Revision	Author
2014-12-10	Original	R. Henwood
2014-12-15	Update intro, description	A. Dilger
2014-12-16	Unit/functional test list	R. Henwood
2014-12-17	Corrected use of COS	R. Henwood

Introduction

The Lustre* Distributed Namespace (DNE) Project seeks to increase the metadata performance and capacity of a Lustre filesystem, allowing multiple metadata targets (MDT) to be used on multiple metadata server (MDS) nodes concurrently. The previously completed DNE Phase 1 Remote Directories project allows scaling aggregate filesystem metadata performance by allowing administrators to split the namespace handling and storage at a directory sub-tree level onto one or more additional MDTs and/or MDSes, while maintaining in a single visible namespace on the client.

The current DNE Phase 2 Striped Directories project continues this performance scaling effort by allowing a single Lustre directory to be stored on multiple MDTs, allowing the performance and scalability of a single directory to increase beyond that of a single server. In addition to implementing striped directories, functionality was added to allow migrating parts of the namespace (inodes and directories) from one MDT to another to allow existing filesystems to take advantage of multiple MDTs. Infrastructure was also developed for an efficient distributed transaction mechanism that allows operations that span multiple MDTs to be safe in the face of server or network failures.

Milestone Completion Criteria

The following milestone completion document applies to Subproject 2.2 – Striped Directories of the Lustre Distributed Namespace Project of the OpenSFS Lustre Development contract SFS-DEV-001 signed July 30th, 2011.

Per the contract, Implementation milestone is described as follows:

> *Contractor shall complete implementation and unit testing for the approved solution. Contractor shall regularly report feature development progress including progress metrics at project meetings and engineers shall share interim unit testing results as they are available. OpenSFS at its discretion may request a code review. Completion of the implementation phase shall occur when the agreed to solution has been completed up to and including unit testing and this functionality can be demonstrated on a test cluster. Code Reviews shall include:*
>
> a. *Discussion led by Contractor engineer providing an overview of Lustre source code changes*
> b. *Review of any new unit test cases that were developed to test changes*

* *Other names and brands maybe the property of others.

Components of Completed Implementation

The DNE Phase 2 Striped Directories implementation consists of several components that address the scalability and usability limitations of the DNE Phase 1 project. The new components implemented for DNE Phase 2 are described more fully in DNE Phase 2 High Level Design but are summarized here.

Striped Directories

In order to improve performance and capacity scaling of individual directories, new directories may be created with a layout that distributes it across two or more MDTs. The directory layout can be different for each directory, and is stored on a *master* MDT object normally residing on the same MDT where its parent directory is. The directory layout contains the number of stripes (or *shards*) over which the directory is distributed, File Identifiers (FIDs) for each directory shard, as well as the hash function for the directory.

Each client doing an operation in the directory uses the layout to hash the filename to determine which MDT object will contain that filename, and then does operations such as create, open, close, unlink, mkdir, etc. directly to the *slave* MDT for that object. No user-visible objects are stored directly in the master MDT object. Since multiple MDTs act largely independently for each file object, all MDTs in that directory can contribute to performance, removing the single-MDT performance limit for large directories.

Attributes for striped directories are aggregated from all of the directory shards at access time, much as they are for file objects striped across OSTs. During directory traversal (readdir()) operations, the directory entries are returned to userspace interleaved in hash offset order so that the readdir operation can be restarted if necessary, and to ensure that all MDTs are active at all times.

Remote Operation Consistency

New with DNE Phase 2 is the ability to do remote MDT rename and link operations that involve two or more MDTs. Unlike remote mkdir and rmdir operations that can be ordered to avoid user-visible inconsistencies in the namespace, the link and rename operations require compound operations to commit atomically and durably across multiple MDTs. Remote operations are split into updates for each MDT that would result in a user-visible inconsistency if committed separately at the time of system failure (e.g. link count too high or low, directory without a name, or multiple names referencing the same directory).

In order to achieve consistency for remote operations, a redo log for each such operation is sent along with the update request to all MDTs involved *in that operation* and is stored on

each MDT. The redo log contains all of the updates for all of the MDTs involved in the operation. If an update commits on at least one of the MDTs before a system failure, then the redo log will also be committed atomically. During recovery, the redo logs are examined for incomplete operations, and contain enough information to redo the missing updates for on any other MDT involved in that operation.

The redo log is only required in case at least one MDT update was committed to disk, since if all involved MDTs crash before any updates are committed they will also be in a consistent state. As a result, independent remote updates and redo logs can be sent and written to disk asynchronously. However, dependent remote operations such as creating a set of nested remote directories DO require the previous operations to be persistent. That is achieved using an existing dependency-tracking mechanism called Commit on Share (CoS).

Inode Migration Tool

With DNE Phase 1 remote directories, renaming a file or directory from one MDT to a directory on another MDT would return a *cross-device* link (EXDEV) error to the caller, which indicates that it needs to copy the whole file or directory to the new location, if possible, or return an error to the user. While this was functional, it may cause a large amount of data movement for large files. With DNE Phase 2, there is a new lfs mv user command that allows users to migrate only the metadata portion of the file or directory (inode, layout, and attributes) to a new MDT without copying any of the file data. The lfs mv command can move whole directory trees in order to allow administrators to load-balance the namespace onto new MDTs that are added to existing filesystems.

Component Patches

The patches containing this functionality are below, in order of planned landing:

Change #	Subject
e88992a*	LU-2430 mdt: Add global rename lock.
0209add*	LU-2430 mdd: add lfs mv to migrate inode.
370de92*	LU-3531 mdt: delete striped directory<
3c216b9*	LU-3531 llite: fix "lfs getdirstripe" to show stripe info
7117ff4*	LU-3531 mdc: release dir page cache after accessing
4e0c8ae*	LU-3531 llite: move dir cache to MDC layer
7f6f701	LU-5420 mgc: MGC should retry for invalid import
3e28034	LU-3536 lod: Separate thandle to different layers.
07c9244	LU-3534 osp: move RPC pack from declare to execution phase
67fe9ef	LU-3534 lod: record update for cross-MDT operation
31bb2c2	LU-3564 mdt: move last_rcvd obj update to LOD
9ec47fe	LU-3564 LOD: add distribution id to identify updates
a603212	LU-3534 lod: write updates to update log
548a70e	LU-3546 lod: cancel update log after all committed
9f71978	LU-3564 lod: update recovery thread
2e6dbe1	LU-3537 mdt: allow cross-MDT rename and link
59aa0b7	LU-3536 osp: send updates by separate thread
0fe99fb	LU-3536 update: change sync updates to async update

* Patches are landed in master.

Functional/Unit tests

suite	id	title
sanity.sh	all	Run test suite with striped directories
sanityn.sh	all	Run test suite with striped directories
sanity.sh	17n	Run e2fsck after migration
sanity.sh	230a	Create remote directory and files under the remote directory
sanity.sh	230b	Migrate directory
sanity.sh	230c	Check directory accessibility if migration is failed
sanity.sh	230d	Check migrate big directory
sanityn.sh	80	Migrate directory when contents are being opened

suite	id	title
replay-single.sh	80a	DNE: create remote dir, drop update rep from MDT0, fail MDT0
replay-single.sh	80b	DNE: create remote dir, drop update rep from MDT0, fail MDT1
replay-single.sh	80c	DNE: create remote dir, drop update rep from MDT1, fail MDT[0,1]
replay-single.sh	80d	DNE: create remote dir, drop update rep from MDT1, fail 2 MDTs
replay-single.sh	80e	DNE: create remote dir, drop MDT1 rep, fail MDT0
replay-single.sh	80f	DNE: create remote dir, drop MDT1 rep, fail MDT1
replay-single.sh	80g	DNE: create remote dir, drop MDT1 rep, fail MDT0, then MDT1
replay-single.sh	80h	DNE: create remote dir, drop MDT1 rep, fail 2 MDTs
replay-single.sh	81a	DNE: unlink remote dir, drop MDT0 update rep, fail MDT1
replay-single.sh	81b	DNE: unlink remote dir, drop MDT0 update reply, fail MDT0
replay-single.sh	81c	DNE: unlink remote dir, drop MDT0 update reply, fail MDT0,MDT1
replay-single.sh	81d	DNE: unlink remote dir, drop MDT0 update reply, fail 2 MDTs
replay-single.sh	81e	DNE: unlink remote dir, drop MDT1 req reply, fail MDT0
replay-single.sh	81f	DNE: unlink remote dir, drop MDT1 req reply, fail MDT1
replay-single.sh	81g	DNE: unlink remote dir, drop req reply, fail M0, then M1
replay-single.sh	81h	DNE: unlink remote dir, drop request reply, fail 2 MDTs
replay-single.sh	110a	DNE: create striped dir, fail MDT1 and client
replay-single.sh	110b	DNE: create striped dir, fail MDT2 and client
replay-single.sh	110c	DNE: create striped dir, uncommit on MDT1, fail MDT1/MDT2
replay-single.sh	110d	DNE: create striped dir, uncommit on MDT2, fail MDT1/MDT2
replay-single.sh	111a	DNE: unlink striped dir, uncommit on MDT1 fail MDT1 and client
replay-single.sh	111b	DNE: unlink striped dir, uncommit on MDT2 fail MDT2 and client
replay-single.sh	111c	DNE: unlink striped dir, uncommit on MDT1, fail MDT1/MDT2
replay-single.sh	111d	DNE: unlink striped dir, uncommit on MDT2, fail MDT1/MDT2
replay-single.sh	112a	DNE: cross MDT rename, fail MDT1 and client
replay-single.sh	112b	DNE: cross MDT rename, fail MDT2 and client

Conclusion

Implementation has been completed according to the agreed criteria. This new functionality allows both the capacity and performance of metadata operations to scale with the addition of metadata servers to an existing filesystem, as well as introducing some important functionality for administrators updating existing single-MDT filesystems to DNE.