CMD Server API HLD*

Huang Hua, Mike Pershin

2006/01/25

Contents

*(Extracted From Lustre Book & CMD2 Code)

1 Introduction

This section is not mandatory. It may contain some introduction points, like scope of HLD, some pre-discussed terms, etc.

This HLD describes the design of clustered metadata server environment in Lustre. Till now, production release of Lustre only has one active metadata server in a cluster, and all metadata is managed by this server. And this configuration has many limitations in several aspects, such as performance, scalability. So the landing of clustered metadata server (CMD) into Luster is necessary and urgent.

2 Requirements

This is mandatory section. It should contain "what management wants" we do in this work.

Lustre now faces many challenges in scalability, performance, availability and others. In order to provide enhanced scalability and performance, Lustre offers clustered metadata (CMD) servers. This document will give an outline of the architecture of clustered metadata in Lustre.

The main requirements for CMD functionality are:

- to provide a substantial gain in scalability of the metadata performance of Lustre through great parallelism of common operations. This involves finding mechanisms which distribute operations evenly over the metadata cluster, while avoiding a more complex protocol involving further RPC's. The current trend in distributed file system design is to do such clustering by allowing clients to pre-compute the location of the correct services;
- to provide good load balancing and resource allocation properties both for large installations where the metadata cluster acts in effect as a metadata server and in the case of small clusters in which the metadata cluster itself will access metadata on other nodes in the cluster;
- Organize the MDS in accordance with new approaches in layering and API.

3 Functional specification

This is mandatory section, it should contain "what should we do (and may be why to do) to meet requirements"

3.1 Definitions

CMD - clustered metadata. Common name for Luste environment with metadata handler but cluster of servers not one MDS.

MDS - metadata server.

FID - object identification in Lustre. It consist of sequence, id and version. FID is invariant in Lustre for the object.

FLD - FID location database. The functionality by which FID is translated to the object. Due to a possible object migration we need such DB to map FID to the object.

3.2 Architecture Overview

MDC/MDT: transport level. It contains all network functionality, handles all LDLM actions and provide network part of recovery. MDC/MDT can pack/unpack fs operations in optimal way to improve network performance, but this should be transparent for upper/bottom layers.

LMV/CMM: Logical Metadata Volume and Cluster Metadata Manager provides connections with metadata servers. LMV is similar to LOV, except that LMV handles metadata, while LOV handles data. Each MDC driver manages the metadata traffic to one remote MDT. MDD driver is used for local operations directly. The object distribution is left to a LMV/CMM driver, invoked by the llite or MDT through the defined API. LMV/CMM redirects operation to the selected MDC or/and to the local device - MDD after applying the placement policy.

MDD: metadata layer. MDD API can be used by LMV on both MD server and client to provide local operations with metadata,and MDD is only layer that operates with metadata. MDD uses OSD as storage device.

OSD: objects storage layer. It provides API for object operations, exports transaction API, locking API, index API and access to the FLD - FID location database.

3.3 CMD functionality basics

Overall the clustered metadata handling consists of the following principles:

1. Multiple MDSes manage all the metadata of the file system:
   - MDSes inter-operate with help of **CMM** device and **FLD** functionality;
   - MDS-MDS relationship are based on the same protocols as Client-MDS;
     - There is a single metadata protocol that is used by the client file system to make updates on the MDSes and by the MDSes to make updates involving other MDSes;
     - There is a single recovery protocol that is used by the clients - MDS and MDS-MDS service;
2. Metadata can be distributed across the MD cluster:
   - the name of object and object itself can be stored on different servers. See 3.5.
   - Directory can be split across multiple MDS like striped files on OSS. See 3.6;
3. **FID** is only invariant over whole cluster and should be used to get other attributes;
4. **FLD** is used to get location of an object by **FID;**
5. Objects can migrate from one MDS to another.

3.4 Metadata placement policy

In order to balance the load of all MDSes, and conform to the locality features of file system, the metadata should be placed in accordance with some policy. That policy should be applicable as on client as on server.

The example of resource location is managed as follows:

File inodes:

Create the file inode on the MDS where the parent directory inode is placed.

Directory inodes:

- Create on an MDS selected by some policy, e.g. round robin based on client node id, hash from name, FID sequence, free space, etc.;

  CMD2: Only new directories in root directory are distributed. Target MDS is choosen by the next rule: NID % (number of MDS)

Directory data:

- While the directory is small, keep it with the inode;

  CMD2: directory is counted as small if its size is smaller than 256K;
- When it grows fan it out. See 3.6

Placement policy is used by **LMV/CMM** device to choose MD servers involved in operation and can differ on client and server sides.

3.5 Distributed entries

Distributed entries - (also **cross-ref** naming can be used) special entry's type in CMD which means that entry with object name is stored on one MDS but directory object itself (inode) - on another one. Such entries can occur as the result of metadata placement policy for directories or splitting of parent directory. Therefore request that need access to directory object should be redirected to the remote MDS. To do this the following should be done:

- file system on MDS should provide a way to store cluster-wide information along with name which will get object location, e.g. **FID** as only unique identifier across the cluster;
- get location of the corresponding object even on remote MDS by using the **FLD**;

  CMD2: directory entry contained MDS number right after entry name, manipulating by rec_len value. So while lookup operation the inode number and mds number were available as result. FID was stored in inode EA.

- for any operation with cross-ref objects MDS do:
  - do additional request to the remote MDS transparently;
  - for lookup request answer to the client with available info, and client can do additional getattr request if needed.

  CMD2: MDS do the same thing

3.6 Clustered directories.

When directories grow we will split them up by creating directory data objects on multiple MDS servers and populating it with directory entries. This is quite analogous to striped files, which are placed in data objects on multiple servers and involves a new RPC in the MDS service.

New directory types are occurred due to split:

1. **master directory** - with an EA in inode pointing to objects on other MD servers;
2. **slave directory** - directories on remote MD servers populated with entries copied from master directory;

Splitting operation and policy should care about:

- splitting policy - split is useful on really big directories;
- small impact of performance - split operation can freeze all MD cluster;
- locking changes due to entries replacement to remote MDS;
- recovery issues;
- rollback compatibility. Some of rollback basics can be broken by split operation.

To distribute all entries across the cluster the hash from entry name can be used so both client and MDS can know the target MDS for selected name.

Split operation itself do:

- scan directory;
- get hash for entry name;
- move entries to the remote MDS based on hash. Optimization is possible by collecting many entries and sending them to the MDS at once.

Note: *There is a fairly heavy penalty associated with splitting the directory and also with renames in within split directories. Moreover, at the point of splitting, inodes become remote and will incur a penalty upon unlink. Splitting can also breaks rollback functionality due to occurrence big amount of inter-MDS operations. Therefore* ***directory split design will be extracted into separate HLD****.*

Problems in CMD2 with split:

- lookup lock on client protects dentry but name can be moved in the same time to another MDS and became non-protected.
  - Bad on small dirs (more RPC for cross-ref entries but there can be a lot of small dirs), but can slow down cluster operations while splitting on really big directories.
    Currently htree is good on dirs (up to 2M entries and up to 9M with Alex's improvements) so split makes sense on directories bigger than such
    * no recovery at all for split operation
    * while doing lookup and then getattr client keep lookup lock that can produce cascading timeouts. There is HLD for that case.

3.7 Locking.

Some detailed works remains to be done for the design to avoid cyclic lock dependencies. The MDS takes locks on the resources it modifies. And these locks are canceled once ACKs are received. In the clustered MDS scenario, it is important to ensure that a deadlock is not caused as a result of the various systems waiting for ACKs from each other.

Locking can be done in FID order as it is currently done on the MDS. In order to obtain cluster wide ordering of resources, clients must choose the correct coordinating MDS, so that locks taken there initiate the lock ordering sequence to be followed. This is particularly important for rename, which has to be started at the target or source directory, depending on which the highest order resource occurs.

There are also situations that one MDS has taken some locks against objects, and need other MDSes to do something on these objects. The master MDS will send RPCs with special flags to other MDSes not to take lock again.

3.8 Clustered MDS recovery.

3.8.1 Rollback

Rollback is a recovery mechanism for a cluster of metadata servers. If the system crashes due to a power failure or due to multiple MDS failures, the problem that we face is that the state of the cluster may not represent a valid file system. The reason it may not is that transactions on different nodes may be related to

a single operation at the file system level. Some of these transactions may be lost in the crash, while others may have committed to disk.

In order to address this problem, the nodes will engage in a distributed algorithm that restores the disk state to the some previous state about what we are sure it is valid. The key feature of rollback is that it is possible to do undo for some operation until we will return to the some valid state. More details can be found in rollback HLD.

3.8.2 Client - MDS replay protocol

The clustered MDS - client recovery protocol is very similar to the single MDS - client protocol. The only one difference is that not all replays should be done due to rollback. All replays related to undo operations will be dropped.

3.8.3 Failover rings.

The configuration data can designate a standby MDS that will take over from a failed MDS. By organizing the servers in one or more rings, the nearest working left neighbor MDS can be the failover node. This leads to a simple scheme with multiple failover nodes, avoiding quorum and other complications beyond what is needed for two node clusters. This topic is described in "Cascading Timeout HLD".

4 Use cases

This is mandatory section, it should contain "how to use or how to check new functionality". It is naturally to use it as design for unit or sanity tests.

4.1 Typical metadata operations

For the most part, operations are similar or identical to what they were before. In some cases multiple MDS servers are involved in updates and partial operations are needed.

4.1.1 Create/mkdir

Client: can only produce *md_create*(pfid, cfid, name) on MD servers;

MDS: may produce *remote_create_obj*(cfid) on remote MD server to create object only. Under the current placement policy, MDS will only remotely create directories under root directory.

4.1.2 Link

Client-MDS: *md_link*(sorce_fid, parent_fid, name, child_fid);

MDS-MDS: *remote_link_obj*(cfid) - to increase nlink counter.

4.1.3 Unlink/rmdir

Client: *md_unlink*(pfid, name);

MDS-MDS: before doing operation local lookup by 'name' is needed (*e.g.* *name2fid*() exported from **MDD/OSD**), then *remote_unlink_obj*(cfid) may be called if remote object;

4.1.4 Rename

Client: *md_rename*(oldpfid, oldname, pfid, name); Let's suppose that this is sent to the pfid home;

MDS: the most complex operation. May calls *remote_create_obj*(), *remote_delete_obj*() as well as ordinary *remote_create*().

4.1.5 Readdir

Client: md_readpage(fid,offset,page,...);

MDS: Can the server readpage at the offset and send it back without additional packing? Consider about the directory entry format.

4.1.6 Directory splitting

All operations making updates to directories can cause a directory split. See 3.6

Note: in split directory all operations except lookup may do request to several remote MDS.

4.2 LMV/CMM route selection

The MDS returns -ERESTART to client when it considers that the request should be restart again. This is mostly because MDS thinks the requested objects should not be handled by itself, or the internal state has changed and the operation should restart again. For example, when a directory is splitted, the create request should be sent again. Before send this create request again, client should predict which MDS will handle this request, and send it to the proper MDS. Please refer to the "client metadata api HLD".

5 Logic specification

Mandatory section, moreover, it is very important. It should contain "how to implement new functionality to meet requirements"

5.1 Locks in CMD

In order to increase the scalability and performance, we will try to take locks optimally. Take PW lock instead of EX lock when we want to modify an object, eg. we only take the PW lock against parent directory in most cases.

5.2 Metadata placement policy

We now propose two methods of placement policy:

1. **Name Hashing**. We may hash the name of the object into MDS mds_num. This policy concerns with the name (only last character or all characters), and neglects the client's influence. All the objects with the same hash value will be stored on the same MDS no matter there objects have some relations or not;
2. **NID Hashing**. We may hash the client NID into MDS mds_num. By this policy, the objects created by the same client will be stored on the same MDS.

Policy 1 may distribute the whole tree more evenly, in statical form. Policy 2 may achieve better locality, and hence produce better performance.

5.3 CMD protocol

In CMD environment several metadata operations may involve more than one MDS. Therefore several MD operations are changed a lot. The MDT APIs not only handles the requests from client, but also handles the requests from other MDSes. The requests from other MDSes will be handled in the same RPC function and number. So pay attention to these "partial operations" (or called "dependent operations").

5.3.1 Create

The create algorithms are as following:

1. check that entry should be on current MDS, return -ERESTART if not;
2. try to split parent, return -ERESTART if it was split;
3. continue as usual for regular files;
4. for a directory:
   (a) apply metadata placement policy and get target MDS;
   (b) if it is local MDS then continue as usual;
   (c) otherwise:
      i. call CMM to create directory object on remote MDS;
      ii. add a special dentry under parent directory which store FID for remote directory object.

5.3.2 Link

The Link algorithms are as following:

1. if the src lives on another MDS (ie. cross-ref):
   (a) get and lock the target directory dentry;
   (b) call md_link of CMM to link the src object(add the src object's i_nlink) on remote MDS,

i. when the remote MDS receives this request,
   A. it gets and locks the src dentry;
   B. adds the src object's i_nlink;
   C. release the lock, put the dentry, finishes this request;

(c) add a special target dentry under the local target parent directory. The target dentry stores FID for remote src object;

(d) release the lock, put the target directory dentry;

(e) finish this request.

2. otherwise continue as usual;

5.3.3 Unlink

The unlink algorithms are as following:

1. if the child object is stored on another MDS (cross-ref):

   (a) get and lock the parent and child dentry;

   (b) call md_unlink of CMM to unlink the child object on remote MDS;

      i. When the remote MDS receives this request:
         A. get and lock the parent and chind dentry; (remember the child's entry name is on another MDS)
         B. it drops the child object's i_nlink;
         C. If this i_nlink drops to zero, handles the orphan;
         D. unlock, put ref, and finish this request;

   (c) delete the child dentry from the parent directory;

   (d) handle orphan if necessary;

   (e) unlock, put ref, and finish this request;

2. lock the slave objects of this splitted directory if necessary;

3. do the local entry's unlink as usual;

4. if directory is splitted, unlink the slaves;

   (a) CMM will dispatch this slave unlink request to all slaves;

   (b) when the remote MDS receives this unlink request:

      i. get and lock parent and child dentry;(Do not lock the master directory because it is already locked)

      ii. unlink the slave object as usual;

      iii. release the locks, put ref, and finish request;

5. unlock the slave objects of this splitted directory if necessary;

6. finish this request.

5.3.4 Rename

The rename operation involves four object: source dir, old dentry, dst dir, new dentry. Theoretically, these fours objects may be stored on four different MDSes. So this is the most complicated case for CMD operations.

For rename, We will follow these rules:

1. We will create new dentry on the MDS where dst dir resides on. This can give us simplicity.
2. How to lock these objects is very important. Please refer to "cmd rename locking HLD" for detail.

We have these algorithms for the rename:

1. If the dst dir is on another MDS:
   (a) We will create the new dentry on that MDS. lock the src dir and old dentry;
   (b) Call md_rename of CMM to rename it on remote MDS;
      i. When the remote MDS receive this request:
         A. get and lock the dst dir and new dentry;
         B. create a special new dentry under dst dir, store FID information for this dentry to reference remote old dentry inode;
         C. release the locks, put ref, and finish this request;
   (c) remove the old dentry from source dir locally;
   (d) finish this request;
2. If the old dentry is stored on another MDS (CROSS_REF):
   (a) get and lock the four objects in order;
   (b) do various sanity checks against the inodes;
   (c) add special new dentry to dst dir, storing old dentry's FID information for the new dentry. If the new dentry exists, check permission at first;
   (d) delete old dentry from source dir;
   (e) release the lock, put ref, finish this request;
3. otherwise (all objects are located on the same MDS), perform this operation as usual;
   (a) get and lock the four objects pair;
   (b) do rename as usual; handle the orphan properly; destroy OST object if necessary;
   (c) release locks. put ref, finish this request.

5.3.5 Open

In Lustre with CMD, there are some differences comparing to Lustre without CMD. The most important thing is to handle splitted directory and check for cross reference.

1. If the parent is splitted and this child dentry should not be placed on this MDS according to dentry placement policy, finish this request with -ERESTART error;
2. If the child dentry is stored on another MDS (cross-ref), return this child's FID information to client, and the client will open it from the proper MDS;
3. If the child does not exist, we should create it. Handle the dir split properly;
4. Open it as usual.

5.4 Split in CMD2

When a directory grows to a pre-defined size, it will spilt. When a directory splits, some entries of this directory will be moved to other MDSes. This MDS becomes the master of this directory, and others will be the slaves of this directory. A new EA is added to the master directory inode which contains information about all remote objects. The slave directories have such EA also to prevent its splitting.

5.4.1 Split rules

There are some additional rules for directory split:

1. root directory will be never split;
2. the splitted directory will be never split again;
3. directory can be split only across all MDS;
4. If some more MDSes are added into system, the splitted dir can not use these MDSes.

Actually adding/removing the MDS can break placement policy based on round-robin. Maybe we should describe such situation and possible solutions.

5.4.2 Split operation

1. create directory objects on remote servers;
2. read dentry's entries into memory;
3. apply chosen policy for metadata placement;
4. collect entries for the same MDS into one list of pages;
5. move lists of pages to appropriate MDS;
6. delete these entries from original dir.

5.4.3 Placement policy

How to placement all the items in the splitted directory among MDSes is an important issue, because it has close relationship with how to implement POSIX readdir operations in Lustre. The split and readdir operations should use the same policy to distribute and find the entries for a directory. Please refer to the "POSIX readdir HLD".

6 State management

Mandatory and very important section, it should contain recovery changes, scalability, formats, protocols, state invariants, state sharing, etc.

6.1 State invariants

This contains all new state invariants, state sharing, etc.

In Lustre with CMD, there will be some new states which are introduced by distributed operations among MDSes, such as link, unlink, create, mkdir, and rename. These distributed operations may involve several MDSes, and some MDSes only do a partial metadata change. The rollback/recovery/failover should pay attention to this situation. Please refer to the "cmd rollback HLD".

6.2 Scalability & performance

This should contain scalability analysis, will it hurt scalability or not, if yes - how much and in what aspects. Some basic algorithm analysis should be done as well. Like if some algorithm linearly depends on input array or array it operates on, this algorithm definitely suffers from scalability issues. In this section also should be done client affecting analysis. This is will new functionality make one client health affect another client or not. Most obvious issue which may be good example is invalidating client's caches which was found by Nikita. In few words, taking not compatible extent lock on one client causes cache invalidation on another clients. If cache invalidation algorithm linearly depends on extent size, one hostile client may cause eviction of all other affected clients, which start invalidate all pages from extent [0 - 4GB] in blocking ast function.

With CMD, we expect Lustre to achieve better scalability and performance. Since the whole file system metadata load is distributed among multiple MDSes, the cluster should handle metadata operations more efficiently. So Luster can support more clients to access file system simultaneously and provide better IOPS and data throughput.

In CMD, the root directory of Lustre (not the root on client side) tries to place its sub-directories among all the MDSes if there is no directory spliting. Each MDS will store and manage these different sub-directories and their descendants respectively. So if these directories belongs to different applications and have few relations between them, almost all the operations will involve only one MDS, and these MDS will work in great parallelism. This will produce much better performance and scalability. In fact we usually do store data in this style. For example, we store database in one directory, and multimedia materials in another directory, and temporary results in yet another directory, and so on. So the metadata access to the data can be distributed and balanced. On the other hand, if we want to achieve better performance and scalability, we should do a better plan how to place the files.

The directory splitting will also bring us better performance and scalability then the directory size is large because several client may operate on a directory in parallel. But it also will bring us some side effect when doing splitting, such as recovery and freeze.

6.3 Recovery changes

This section should contain all recovery changes introduced by new feature or functionality. Most of recovery scenarios should be described.

The recovery protocol is the same as before, and the client-MDS and the MDS-MDS will use the same protocol.

6.4 Disk format changes

Any disk format changes should be described in here. This different EAs, some special directories or files (ROOT, OBJECTS, etc.), endiang, etc.

The back store file system will change a bit to contain some information required by CMD. Please refer to "Index API Module HLD".

6.5 Wire format changes

Wire structs changes, size, fields order, endiang, etc.

Some wire format need to be changed because of FIDs and others. Please refer to "FID HLD" and other HLD.

6.6 Protocol changes

Any protocol changes, changes in connecting, etc.

All the objects in Lustre will be identified by FID. Please refer to "FID HLD".

6.7 API changes

This section describes API changes, changes in existing functionality even from renames point of view.

The MDT will not only handle the requests from clients, but also handle the requests from other MDSes. The request from other MDS is a partial operation in a distributed operation. We will use the same APIs to handle all the requests, but they may have different roles and act differently.

6.8 RPCs order changes

This section contains RPC order changes (if any) which may affect existing functionality or recovery. Most obvious example is CROW. It changed RPCs order in create time by creating OST objects from client on first write or setattr. This caused later quite serious issues with concurrent create-by-CROW and unlink, etc.

There is no change in RPC order.

7 Alternatives

Not mandatory section. May contain alternative points of view on some aspects of HLD.

7.1 Lock order

The FID resource order locking is hard to implement, while the parent-child order is easier. Shall we use the FID resource order or the parent-child?

7.2 metadata placement policy

we only apply metadata placement policy against the sub-directory of the root (if we have no directory splitting). But will this rule limit the scalability and performance? For example, all the data files of some HPC is located in /HPC/. Under the current rule, all the metadata of these files will reside on a single MDS (no dir splitting). This HPC will not gain better performance if all the clients connect to this MDS in a cluster. But if we carry out the metadata placement policy on every directory (put the sub-directory on another MDS), it has some side effect also. For example, some data files have close relationship and will be access together. If these metadata are on different MDS, operations on the metadata will be expensive.

So I think we should have something with more complexity. (1) The system can decide the placement policy. (2) the user can also decide the placement policy for the whole file system or for a specific directory.

8 Focus for inspections

Not mandatory section. It may contain references to some important parts of HLD. Some complicated algorithms, disputable or opaque solutions, etc.

1. Locks. Locks for CMD is very important. The lock order, lock resource, and lock mode should be inspected.
2. what should the disk format for dir_entry be?
3. what should the struct format and flags of “struct dentry” in memory be?
4. What about the load balancing? How to do load balancing? automatically or manually? When to do it?