MDS Layering HLD

Mike Pershin

2006-02-27

Contents

1 Introduction

The requirements of many new approaches in design of metadata server demand a clear understanding of server layers and their functionality. Current document describes this and should be used as basis for all depended designs.

2 Requirements

Requirements to the MDS layers are taken from management document and listed below:

- the MD target or transport layer is needed;
- the MDD (local metadata driver) shouldn't directly call LVFS but instead layer on OSD;
- the OSD object API should be extended with some metadata elements, such as the manipulation of trees, locks and transactions;
- the MDD API should be the same as the Lustre file system requires;
- move all elements of networking into MDT layer. Exports and imports, network recovery, DLM are strictly belong in the MDC/MDT layers;
- MDD and OSD should not contain per client data structures;
- llite and LMV could run directly on CMM and/or MDD w/o any use of LNET;
- the new locking functionality should be introduced in MDD due to moving all DLM stuff into MDT level;
- expose transaction API for use by MDD.

3 Functional specification

3.1 General notes:

According to new MDS layering different layers export quite different interfaces to the upper layers. With this in mind, it stand to reason to question the necessity of having global obd interface that we have now. For another thing, mds layers are not *object* devices. They deal with meta-data. Probably now, when meta-data components undergo major rewrite it's a right time to cleanup interface core:

- introduce `struct mdX` (not sure what X should be used) common for all meta-data layers and separate from `struct obd`;
- introduce some structure for common parts of `struct obd` and `struct mdX` (like reference counting, initialization/finalization, etc.);
- implement every MD layer as `struct mdX` plus some layer specific methods (e.g., abandon current practice of stuffing each and every possible method directory into common interface).

3.2 Definitions

FID - object identifier 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 mdsnum. Due to a possible object migration we need such DB to map FID to the server number.

3.3 Architecture Overview

There is new layer on MDS named **'CMM'** - Cluster Metadata Manager. This layer will make all functionality related to MD cluster - MDS-MDS interaction, epoch control, placement policy.

3.4 Functionality overview

The proposed layering is featured by CMM (Clustered Metadata Manager) on top of MDD. CMM layer takes care about placement policy and uses MDD for local operations and a set of MDC - for remote calls. CMM can be removed so MDT will works on top of MDD (for non-CMD lustre. Do we need this?). So the request handling algorithm looks like followed:

1. MDT receives the request from client (MDC):
   (a) unpack request information;
   (b) handle resends;
   (c) take needed DLM locks;
   (d) call below OBD and send reply to the client.
2. CMM chooses all servers involved in operation and send depended request if needed:
   (a) apply metadata policy using request parameters and some other info from MDT if needed, e.g. client NID;
   (b) for local operation just call MDD;
   (c) for complex operation:
      i. split operation to several parts;
      ii. do epoch negotiation;
      iii. call remote operation on selected MDC;
      iv. call local operation on MDD
   (d) get all results and return with combined one;

(e) queries and updates FLD.

3. MDD is local metadata device that receives all local calls from CMM

   (a) take all needed locks;

   (b) start transaction if needed;

   (c) execute operation via OSD.

4. MDC acts in the same way as on client:

   (a) prepare request;

   (b) send it to the corresponding MDS.

3.5 Server Layers

3.5.1 MDT

This is transport level. It contains:

- all network functionality, so other layers (except for MDC) know nothing about network;
- **LDLM** functionality. Other layers shouldn't use LDLM at all;
- **resent** functionality;
- intent handling.

This layer takes care about all networking components. The ideal case is that MDC/MDT are removed so llite/LMV on the client could run on top of CMM or MDD directly. To provide this MDC and MDD should have similar API. MDC/MDT can pack/unpack fs operations in optimal way to improve network performance, but this should be transparent for upper/bottom layers.

3.5.2 CMM

Clustered Metadata Manager provides inter-server communitcation related to CMD environment and decides how data will be/should be distributed across the cluster. CMM does the following:

- apply placement policy if needed;
- handles **FLD** - FID Location Database;

- redirects operation to the **MDD** directly or do complex operation with requesting the remote **MDS;**
- support the array of **MDC** drivers that manage the metadata traffic to the remote **MDT**;
- in addition to the standard set of fs operations **CMM** supports partial operations for cross-ref objects support;
- rollback functionality (just because it is needed for multi-MDS environment only and not needed w/o CMD) - snapshot control, epoch negotiation, undo log

3.5.3 MDD

Metadata layer. There all metadata operations are processed:

- MDD API can be used by CMM/LMV on both MD server and client to provide local operations with metadata
- MDD is local driver and know nothing about CMD.
- MDD uses OSD as a storage device.

3.5.4 OSD

Objects storage layer:

- provides API for object operations
- exports transaction API
- exports locking API
- exports index API
- provides access to the FLD

3.5.5 LOV

LOV layer is need on MDS to provide connection with OSSes. It cannot be placed nor in MDT or CMM - because they don't exists in several configurations. It seems MDD should be connected to the LOV

3.6 Layers API

The ordinary filesystem methods:

- `lookup()`
- `create()`
- `link()`
- `unlink()`
- `rename()`
- `symlink()`
- `open()`
- `close()`
- `setattr/getattr()`

The partial methods due to CMD functionality:

- `create_object()`
- `delete_object()`
- `link_obj()`
- `unlink_obj()`

4 Use cases

4.1 Possible configurations

4.1.1 Single MDS server:

MDT => MDD => OSD: no CMM layer and therefore no placement policy, no roll-back, no partial operations calling.

4.1.2 Cacheing MDS

The idea is MDD is started locally on client and works as cache MD server. This implies that **llite** should be able to run on top of MDD. While this looks simple there is one issue - all files will belong to one server so after flushing they will be placed also on one MD server. For Lustre setup with only one MDS server this is ok, but it becomes the problem for CMD. It is possible to change fid_sequence rapidly, but in absence of LMV this will be done w/o any policy.

To solve this issue the following scheme can be used:

The LMV is used on top of several local MDD. In that case it does usual things but instead array of MDC it has array of MDD. It can be actually one MDD that used in all LMV targets. Therefore the *create*() can looks like the following:

1. **llite** calls `md_create(lmv, ...);`
2. LMV choose the server (fid_seq) and call `md_create()` on selected target MDS - MDD actually;
3. MDD do the usual operations with locking, transactions, etc. and call own OSD (that could be the one OSD for all)

It is the one of possible approach and additional inspection/review is needed while DLD/PROTO stage. Another way is listed in 'Alternatives' section 7

4.2 Recovery

The existence of CMM will not break recovery things.

4.2.1 Resent

The resent functionality can be done in MDT, possibly with assistance from MDD/OSD. CMM policy for resent helper is just passthrough it directly to the local MDD.

4.2.2 Replay

Replay operations can be done as original operation. Rollback and FID remove the difference with ordinary commands. MDT know the epoch of last committed snapshot and will pass to the CMM only needed replay operations. Due to rollback all servers in cluster will undo depended operations as well, so CMM will do the same things as for ordinary request with depended replays to the remote MD server if needed. FLD is updated also as needed.

4.2.3 Rollback

CMM can interoperate with servers and can be used for epoch controlling. This will works with help of MDC/MDT or by direct request to the MDD. As epoch is not network-related value but number of local snapshot so there is no network part in CMM - it just calls `md_get_info()` from one of server.

Neither MDT nor CMM have access to the transaction API, so undo log cannot be recorded in these layers. MDD will do undo log for each operation transactionally.

5 Logic specification

5.1 MDT

MDT layer should hide all network parts from CMM/MDD. Therefore neither CMM nor MDD will know whether are they working under MDT or directly under llite/LMV. This implies that MDT should use the same operation like llite/LMV while working with CMM or MDD.

5.1.1 LDLM locks

MDT handles all LDLM locks and this requires lookup method to be exported from MDD to get FID from the name

5.1.2 Intents

The `lookup()` operation involves the intents. The MDT takes care about them and calls appropriate functions from bottom layers.

5.2 CMM

CMM layer is a key component of CMD. It contains all CMD functionality actually. Client-side CMD-related layer is called **LMV.**

5.2.1 Placement policy

When CMM receives the operation it should recognize all involved servers. For existent object CMM check fid_sequence and gets MDS number from FLD. For any new object CMM chooses the target server and updates FLD with new record. In this case the defined policy should be applied to choose MDS and that policy should be duplicated in LMV on client.

5.2.2 Rollback

Rollback is recovery mechanism in a cluster. The CMM will take care about all stuff related to the rollback - epoch control and undo logging. Epoch is controlled through negotiation with all MD servers periodically or during inter-server operation. CMM takes care that cross-ref operation will be done in one epoch. Another part of rollback functionality in CMM is undo log - it can be done by using llog functionality that all OBDs have already.

5.2.3 Multi-server operations

CMD is featured by cross-ref operations which contain two parts - remote and local, e.g. `create_object()`, `insert_name()` as parts of `create()`. These partial commands have to be added to the MDD/MDT API.

5.3 MDD

The main idea about MDD that it is 'local' layer and knows nothing about network, CMD, etc. MDD can be used by CMM, MDT directly or llite on client, so MDD API should contains all needed methods to provide such wide functionality. Logic of MDD is described in MDD_OSD HLD in details.

5.4 Logic of operations on MDS in CMD environment

The proposed solution implies that MDD should support as usual set of fs operations - create/link/unlink/etc. as partial operations appeared due to CMD - create_obj/insert_name/etc. The common set of operations is needed in MDD because it can be used w/o CMM. The set of partial operations is needed because the different MDS can do each part. Due to this the CMM should check is the operations local or distributed and use the appropriate method.

Alternate solution is described in 'Alternatives' section 7

5.4.1 Create

llite: invoke `md_create()` on LMV

LMV: apply placement policy and assign FID for the new child, call `md_create()` on parent's MDC

MDC: send request to the needed MDT

MDT: receive request, unpack and call `mdt_create()` :

```
mdt_create(obd, pfid, name, cfid, flags, ...){
        fid_lock(pfid, LCK_PW);
        md_create(obd, pfid, name, cfid, flags, ...);
        fid_unlock(pfid, LCK_PW);
}
```

CMM: called from MDT. Determine all servers involved, split operation if needed

```
int cmd_create(obd, pfid, name, cfid, ...) {
        /* check the child */
        mdsnum = fld_lookup(cfid);

        if (mdsnum == local) {
                /* call MDD operation */
                md_create(mdsnum, pfid, name, cfid, ...);
        } else {
                /* split operation to call it on different targets */
                error = md_create_obj(mdsnum, cfid, ...);
                if (!error) {
                        /* insert name locally */
                        md_create_name(local, pfid, name, cfid, ...);
                }
       }
}
cmm_create_obj(cfid, ...) {
       ...
}
```

MDD: called from CMM, care about locking/transaction and complete operations with calling OSD

5.4.2 Unlink

MDT:

```
mdt_unlink(obd, pfid, name, ...){
        fid_lock(pfid, ...);
        cfid = md_lookup(obd, name);
        fid_lock(cfid, ...)
        md_unlink(obd, pfid, name, cfid, ...);
        fid_unlock(cfid, ...);
        fid_unlock(pfid, ...);
}
```

CMM:

```
int cmm_unlink(pfid, name, cfid, ...) {
        mdsnum = fld_lookup(cfid);
        if (mdsnum == local) {
                /* call MDD */
                md_unlink(local, pfid, name, cfid, ...);
        } else {
                /* remote operation */
                error = md_unlink_obj(mdsnum, cfid, ...);
                if (!error) {
                        /* remove the directory entry */
                        md_delete_name(local, pfid, name, cfid, ...);
                }
       }
}
cmm_unlink_obj(cfid, ...) {
       ...
}
```

5.4.3 Lookup and Intents

llite: call `md_lookup()` with needed intents, then call other operations - `md_open()` or `md_getattr()` as needed

LMV/MDC: after `md_lookup()` results can have results also for following `open/getattr` or do additional RPC for this

MDT: can receive the `lookup()` from client with IT_OPEN or IT_GETATTR. cfid is passed only in O_CREAT case

```
mdt_lookup(obd, pfid, name, cfid, ...){
        fid_lock(pfid,...);
        fid = md_lookup(obd, pfid, name, ...);
        if (!fid && !cfid) {
                 return -ENOENT;
        }

        if (IT_OPEN) {
                if (md_open(obd, fid, cfid, ...)) {
                        // fill openhandle;
                }
        }
        if (IT_GETATTR) {
                md_getattr(obd, fid, ...);
```

```
        }
        fid_unlock(pfid);
}
```

CMM:

```
cmm_lookup(pfid, name, ...) {
        md_lookup(pfid, name, ...);
}
cmm_open(fid, cfid, ...) {
        mdsnum = fld_lookup(fid);
        if (mdsnum == local) {
                /* call MDD */
                md_open(local, fid, cfid, ...);
        } else {
                return -ERESTART;
        }
}
cmm_getattr(cfid, ...) {
       ...
}
```

MDD: while `lookup()` is trivial, the `open()` need detailed view. We need open here because MDT/CMM layers are optional and llite can call `open()` directly on MDD

```
mdd_open (fid, cfid, ...) {
        if (cfid) { // O_CREATE case
                 osd_create_obj();
                 lov_create_obj();
        }
        //get refcount to the object
        osd_get_object(cfid, ...);
}
```

The comment is needed here about `-ERESTART` return code. When the current MDS cannot provide intent-related data it returns just lookup data with `-ERESTART` code. Therefore a client will do additional RPC to the other MDS.

6 State management

6.1 Scalability & performance

MDS layering changes are initiated in view of clear design and code. It shouldn't hurt performance and scalability

6.2 Recovery changes

Recovery API is changed due to new layering. Network parts are in MDT, the MDD should export API for replay and reply reconstruction and CMM should provide cluster-wide recovery - rollback. There are separated HLDs for this.

6.3 Protocol changes

Locking should be changes to the parent-child order instead of FID order. While children can be at remote server, the parent-child order becomes natural but FID order will be difficult to implement.

6.4 API changes

This design implies that MDD can be used by client directly w/o MDC/MDT. Therefore the API of MDC and MDD should be the same.

7 Alternatives

7.1 Cacheing MDS issue

Alternative solution to the 4.1.2

The problem is - there is only one cacheing MDS but logically llite should works like with several one. Therefore some new level is needed that will only apply the placement policy and select FID-sequence. That level can works on top of LMV - in that case all requests will be redirected to proper server - or on top of MDD directly - in that case all requests will go to the local storage. Possibly it is not even 'layer' between llite and LMV but some kind of library that can be used by llite directly.

7.2 Reduced command set in MDD

Alternative solution to the 5.4

This solution suppose that MDD provides only set of basic operations with name and object like insert/delete name, create/remove object, etc. Therefore CMM split any command into such sub-operations and call them. This will simplify CMM operations due to they will looks similar for cross-ref operations and for local one. But this means also that MDD cannot used by llite/LMV or MDT directly, because they works in terms of usual operations. Therefore the CMM should always be used on top of MDD to provide set of usual operations. There are several problems with this solution but proposed way should be reviewed while DLD/PROTO stages.

8 Focus for inspections