CMD MDS Recovery HLD

Mike Pershin

February 7, 2008

1 Introduction

Introduction of CMD leads to many changes on MD server and on clients sides. The recovery is one of several areas where changes are quite big. The changes will occur due to rollback functionality, more strict layering on server and multi-MDS environment. This document describes basic recovery changes in CMD.

2 Requirements

The CMD environment requires the reviewed recovery due to major changes in functionality. The new recovery design should cover the following issues:

- recovery cases occurred due to cross-ref situations;
- land all recovery fixes from CMD2 if they are applicable still;
- changes in recovery logic due to FID introducing;
- client and server changes due to rollback;
- changes due to moving network things to the MDT level;
- changes in recovery API to make it explicit and clear;
- recovery and inter-server communications;
- proper orphan handling;
- multi-threaded recovery approach investigation.

3 Functional specification

Changes in MDS layering imply that all networks functionality should be placed in MDT layer but MDD is quite simple and do only local stuff. Therefore recovery cases will be handled in MDT or CMM level with assistance from OSD possibly.

3.1 Recovery changes due to an new layered MDS

3.1.1 LAST_RCVD file handling

MDT receives the request from client and the LAST_RCVD file should be updated also. Moreover this should be done in one transaction with operation itself. The transaction callback in MDT is used to update the LAST_RCVD.

3.1.2 Last committed transaction

MDT should supply the clients with last_committed_transaction data, so client can purge committed requests. This will be changed due to rollback functionality - the last_committed_epoch will be used instead of this. The details will be in rollback DLD.

3.1.3 Reconstruction

The MDT should contains the methods for reconstruction. All reconstructions can be done inside the MDT using the normal MD API. The reconstruction will be easier in some cases due to FID.

3.2 Replay changes

3.2.1 Rollback changes

Client cares about pending requests using the last_committed epoch value instead of transaction number. It is enough while normal operation, but in case of MDS failure the information is needed from LAST_RCVD file also.

3.2.2 FID and replay

With fids the replay becomes simpler. Replay functionality on the MDS looks similar to the usual operations in the most cases and reuse the usual methods greatly because of FID design. The open/create replay becomes simpler due to the fact that client always knows FID for open/create operation;

Server part for replaying is also changed because it may apply only part of replays by checking the undo log.

3.3 Cross-ref recovery issues

3.3.1 Timeout on the client

The delay of cross-ref command execution can invoke the timeout at the client:

1. Client send the request
2. MDS1 received it and ask MDS2 for it's part of action. MDS1 send the request later than client obviously
3. MDS2 didn't answer for a long time so timeout on client will occur. There are two cases:
   (a) MDS2 is failed
   (b) network delay
4. While MDS2 was the point of failure the MDS1 looks guilty for the client.

The solution were implemented in CMD2. It implies the following:

1. no reconnection is accepted while there is requests in progress from the same client
2. MDS failure are tracked by clients and timeouts will be postponed until MDS will return back. See the *cascading-timeouts-hld* for details.

Another solution are the periodic keep-alive messages to the client that MDS1 sends until MDS2 recovers.

3.3.2 New requests and partial requests

Due to cross-ref operations there are many cases occur when requests should be done in different manner or additional checks are needed - for example using FID instead of name, partial operations like creating the object but not name, etc.

3.3.3 MDS-MDS recovery

The MDS acts as client while talking with another one. Therefore the resend and replay will be done while MDS-MDS recovery. The only exception is case of rollback - in that case the initial MDS should drop all queues related to other MDS because these requests will be replayed/resent by clients after rollback will be done.

4 Use cases

All use cases are intended to 11/17 tests. The cluster is started up and clients runs several applications. One of the node is failed and other nodes/clients shouldn't be affected. Due to requirements of test 17 the recovery time should be not more than 5 minutes in each case.

4.1 Client failure

Client is failed and normal operations should continue. Other clients and servers should receive no errors.

4.2 Singe MDS failure in CMD

One MDS is failed, recovered and normal operations continue.

4.3 OST failure

OST is failed and recovery is started, clients should get no errors.

5 Logic specification

5.1 Recovery and new MDS API

5.1.1 Updating the LAST_RCVD

The LAST_RCVD is updated by using callback, registered by MDT. There is corresponding API in new layered MDS. Also, another callback has to be used to reserve enough space in transaction handle to do LAST_RCVD update.

5.1.2 Resent and reconstruction

Resent should affect only recovered MDS, all cross-ref cases should be handled properly.

Reconstruction methods are the same as in old MDS but they are using new API.

5.1.3 Replay and FID simplifications

Replay request uses the same methods like ordinary command. The FID of object is defined by client in both cases, therefore the only difference is the MSG_REPLAY flag

5.2 Cross-ref recovery handling

Let's suppose that client requests the MDS1 which in turn ask MDS2 for part of operation. MDS-MDS recovery happens when MDS2 didn't answer in time.

5.2.1 Timeouts dependencies

After MDS2 failure the timeout on client will happens at first with start of recovery for MDS1 which is not failed actually. To avoid such situation the timeouts for MDS-MDS request can be smaller than one for client-MDS requests. This doesn't guarantee that client will gets no timeout, but reduce the probability of such event.

If this situation will occur the client will wait for MDS1-MDS2 recovery

5.2.2 MDS1 failure

MDS1 can fail after the remote operation is done. In this case only future rollback functionality will helps to undo partial changes in the cluster.

6 State management

6.1 State invariants

FID is invariant so replay will use the same FID as ordinary operations did.

6.2 Scalability & performance

Multiple request ability can affect the performance and scalability greatly.

6.3 Recovery changes

The case of client failure is not affected by the new recovery approaches.

6.4 Protocol changes

PtlRpc functionality should use the last_committed epoch value in reply to the clients when rollback will be introduced.