HLD for Open Recovery

Huang Hua

February 7, 2008

1 Introduction

This document describes the handling of open recovery in CMD. Please refer to ../HLD/cmd-open.lyx, ../HLD/cmd-recovery.lyx, ../DLD/cmd-recovery-dld.lyx and ../book/recovery.lyx for some related information.

2 Requirements

The open recovery should provide the following features:

- handle open request replay;
- handle open request resent;
- orphan handling; (there is another separate hld/dld for orphan handling –cmd-orphan-handling.lyx)

3 Functional specification

3.1 Something about recovery

When some failure happens, recovery is needed. These failures include client failure, MDS failure, OST failure and network failure. Open recovery is mainly the recovery for MDS failure.

MDS has to handle two types of open recovery: open replay and open resent.

- Replay. Generally speaking, if a client has got reply, and the request change has not been committed by MDS, the client will send replay request to MDS when MDS does recovery. Client knows whether a request should be replayed or not by checking the transaction number and last committed transaction number. This is described in ../book/recovery.lyx. But open request is a little different from other requests. Open request is kept valid for replay purpose until the opened file is closed. Client keeps a reference count on the open request to keep it valid and keep it from destroyed.

- Resent. Resent is any request that was sent but no reply is received yet. In case of failure all such requests are not committed usually. If request was committed but not replied then reply was lost for some reason and reconstruct is needed. It is only one, rare case of resend.

3.2 Open request replay

In CMD, the child object fid is always provided by client and is always valid if client wants to create something. When MDS does open replay, it should the followings:

- check the child object to see if it exists on disk: as an orphan or as a common object.
- if the child object does _NOT_ exist, follow the normal open code sequence;
- if the child exists on disk, return attribute of that object back to client, and create an open handle for it.

After getting the reply, client should update child fid in open request. Therefore while replay there is correct FID of opened object. This will allow us to use the way above without lookup by name, but using open by fid. Because sometime, the child has been delete, and becomes an orphan. So, there is no name for an orphan, and MDS has to open it by fid.

3.3 Open request reconstruct

If the reply is lost, MDS should re-construct the reply for that open request. It should:

- restore request status from mdt_client_data which is stored on disk within the same transaction as the operation;
- restore locks for this request if necessary; (OPEN_LOCK is not supported in current branch);
- return attribute for child object to reply message;
- lookup open handle in hash table, or create an open handle for this request, and then return this handle to client.

3.4 Orphan handling

Orphan handling will be done by MDT, together with MDD, OST. MDD should keep open count for opened file, and handle deleted open file properly.

4 Use cases

There are two types of open recovery on MDS: open replay and re-construct open reply. This is described in ../book/recovery.lyx. Please refer to that document.

4.1 Life cycle of open and close request

we will describe the open and close requests life cycle when we touch a new file on client: touch /mnt/lustre/foo

1. The client mount point is /mnt/lustre. The client will first getattr() for "/mnt/lustre".
2. The client will call ll_lookup_nd(), and which will call ll_lookup_it() with proper intent. In this case, the intent is open|create. If this request has a create flag, client will first allocate fid for new child with the help from LMV. Then, client will call lmv_intent_open(). And LMV will call proper mdc_intent_lock(). MDC will call mdc_enqueue() to communicate with MDS. The open request now has three reference count.
3. ll_create_it() will be called with the same intent, and consequently the same open request. This function will call ll_create_node(), and open request reference count decreases to two.
4. ll_file_open() will be called with the same intent and consequently the same open request. Because the open has been created and opened on MDS, the client will call ll_local_open() to open this object. At this point, an obd_client_handle will be allocated to hold the opened handle, and in MDC, a mdc_open_data will be allocated to record the request, and set commit callback(mdc_commit_open()) and replay callback(mdc_replay_open()) for this open request. After this, the open request still has one reference count, and kept valid.
5. After the file is created and open, the file will be closed eventually. So, ll_file_release() will be called to close this file. Client will call ll_close_inode_openhandle() to close the opened handle, and which will call lmv_close(), and which will call mdc_close(). MDC will set commit callback for this close request, and sends out the close request to MDS. When MDC gets reply and knows the close request has been committed, the corresponding open request will be set "not replay". And the obd_client_handle will be deallocated.

6. After sometime, the open request will be cleared from replay queue, and mdc_commit_open() will be called. At that time, the mdc_open_data will be deallocated. At this moment, the open request will be destroyed.

4.2 Open recovery handle on MDT

MDT check the flag of request to see if this request is a normal request, or a replay request, or a resent request. MDT should re-construct the result for a resent request; and should replay all the changes for a replay request.

4.3 Single point of failure

Lustre promises to survive from single point of failure. If there are more than one failure at some moment, Lustre may lose some data and/or metadata. This is also true to open request.

5 Logic specification

5.1 Replay an open request

- Check to see if the request is a replay open request. If it is, then:
  - if create transaction was not committed and create was lost, we need to repeat open+create; This can be checked by looking up the child to see if it is already exists. we can handle this situation as the normal open.
  - transaction was committed and all changes are on disk, so only open by fid is needed. In this situation, we need to pack attributes of child back to client.
  - in this scenario, the original transaction number which was returned to client is used as this replay's transaction number. That is to say: in replay, the transaction number keeps the same.

5.2 reconstruct an open reply

- Check to see if the request is a resent open request. If it is, try to reconstruct it based on last_rcvd data. Otherwise, handle this as normal request.
- The last_rcvd data will be updated in the same transaction as the operations.

```
int mdt_open(struct mdt_thread_info *info) {
        struct ptlrpc_request * req = mdt_info_req(info);
        if (lustre_msg_get_flags(req->rq_reqmsg) & MSG_REPLAY) {
                child = mdt_find_object(child_fid);
                if (child exists) {
                        pack attribute of child back to client;
                        create open handle and do other things.
                        return 0;
                } else
                        goto normal_case;
        } else if (lustre_msg_get_flags(req->rq_reqmsg) & MSG_RESENT) {
                restore request from lasv_rcvd;
                check disposition to see how we ad executed this open.
                pack attribute of this child object back to client;
                search/create openhandle for this open.
                return;
        }
normal_case:
        parent = mdt_find_lock_object();
        ...
}
```

5.3 Orphan handling in open

Orphan handling in open recovery is mainly about what to do in handling of open replay request. An opened file may be deleted, and becomes an orphan. So, when replay open, MDS may want to open an orphan, and cleanup this orphan when file closed. The orphan handling is described in ../HLD/cmd-orphans-handling.lyx.

6 State management

6.1 FID

The child object fid is always provided by client and it is valid if this is an open | create request. This is the difference between CMD2 and CMD3.

6.2 Open Lock

Currently, we do not support open lock. So, open recovery need not care about lock fixing up.

6.3 Parallel close request

From now on, the MDS_CLOSE request can be issued concurrently with other request from a single client, and may be handled by MDS in parallel.

7 Focus for Inspection