CROW (CReate On Write) HLD

Yury Umanets

10th February 2006

Contents

1 Introduction

This HLD describes CROW (CReate On Write). In few words, in current approach, in file create time, client sends RPC to MDT and then MDT is supposed to create object on OST server using pre-creation pools approach.

This approach has few downsides:

- pre-creation pools is quite big piece of complicated code which needs to be maintained;
- object created on OST is not really needed before any data gets written to it;
- CROW proposes better scalability in big clusters and does not need to adjust pre-creation pools for particular Lustre application.

This HLD describes changes needed to implement CROW as well as tries to foresee all possible issues and work them out.

2 Requirements

There are the following requirements:

- OST objects are created upon first write or setattr;
- OST saves objects metadata FID and striping info with inode, what is used later for reconstruction;
- OST saves objects ownership in regular attributes. This is going to be used for quota.

3 Functional specification

To meet requirements, the following should be done:

- objects on OST are no longer pre-created in time of creating inode on MDT;
- in create time, MDT maintains objects fids (one fid per stripe) and saves them transactionally;
- OST creates objects upon first write or setattr from client;
- OST stores metadata FID and striping into objects EA. This is used for reconstruction;
- OST stores object ownership in inode regular attributes. This is used for quotas;
- there are few nasty cases of creating object by CROW wrongly. They should be handled out especially. See section 5.2 for details.

4 Use cases

Using CROW:

1. write() and setattr()
   (a) in create, when client sends create RPC to MDS, MDS does not create object on OST. Instead it allocates new FID for OST object and returns it to client;
   (b) in first write or setattr time, client sends OST object FID obtained from MDS in create time to OST along with write RPC;
   (c) OST looks up the object using by FID and creates it if not yet created;
   (d) OST stores FID along with striping info to EA.
2. read()
   (a) we consider that client created file already;
   (b) in read time, client sends read RPC to OST along with OST object FID obtained in create time from MDS;
   (c) OST looks up for object by FID and does the following:
      i. if object already exists - OST behaves as usually;
      ii. if not yet - emulates empty OST object, that is, does nothing.
3. lvbo_update()

(a) OST lvbo stuff emulates empty object just like in read() case. That is does the following:

i. if object already exists - OST behaves as usually, that is, takes object size and attributes from it;

ii. if not yet - emulates empty OST object, that is, returns zero size and zero mtime. Zero mtime makes client use last one from MDS, that is POSIX is not violated.

5 Logic specification

To implement functional specification, the following changes to be done.

5.1 Overall changes picture

- MDT does not perform pre-creating objects on OST;
- in file create time, MDT obtains DT (data) object FIDs (one per stripe) from client along with its MD (metadata) FID. Then MDT saves DT FIDs transactionally for created file;
- client saves both FIDs in its inode info and uses them for talking to MDT (MD FID) and OST (DT FIDs);
- in first write, client sends both FIDs (DT and MD) to OST;
- OST creates object with accordingly to DT FID, saves them both into EAs and sets local object index up to be able to find objects later by FIDs;
- in recovery time, OST removes objects beyond last used one. See section 6.3 for details.

5.2 Wrong objects create

There are possible wrong object creating cases performed by CROW as result of race or another issue. They are the following:

5.2.1 Asynchronous settattr from MDS

1. CROW tries to create OST object upon asynchronous setattr (possibly) from MDT;
2. in the meanwhile, client sends unlink to OST;
3. OST now have race CROW vs. unlink. This leads to the following issues:

(a) OST re-creates unlinked objects in the case unlink path wins the race. That is, CROW re-creates objects which has no presentation on MDT already what consumes space on OST needlessly and causes later issues about space management;

(b) if setattr path wins, nothing bad is happening from operations order point of view, but in this case CROW performs not really needed actions - creating object which is already deleted on MDT and going to be deleted on OST as well.

5.2.2 Two clients case

1. client A makes setattr on a file and setattr on metadata object is done, setattr for data object is queued;
2. client B unlinks the same file, unlink of metadata object is done, unlink of data object is queued;
3. unlink data objects RPC arrives first and data object is deleted;
4. setattr RPC for data object comes to OST and creates OST object mistakenly again.

5.3 Solutions for wrong objects create issue

There is possible solution for wrong objects create issue. It is the following:

- llog approach.

5.3.1 Using llog approach

Unlink operation in OST should be logged using llog. Then later, in create time, when OST sees that object is not created yet, llog could be inspected to see if object was already unlinked. As this operation is not performed oftent it should not cause substential slowdown.

6 State management

6.1 State invariants

The following state invariants should be taken into account:

- object existing on OST should have presentation on MDT;

- object existing on MDT not mandatory has object presentation on OST. This is what CROW does;
- existing OST objects should have EAs and correct ownership attributes.

6.2 Scalability & performance

Scalability should be improved in compare to former pre-creation approach. Also with CROW one does not have to adjust pre-creation pools size, etc.

As for performance, create performace should not change much.

6.3 Recovery changes

There are many possible scenarios when creating orphans on OST is possible. OST orphan is an object which has no presentation on MDT and thus, cannot be accessed, removed, etc. Apparently OST should constantly take care of them. Orphans should be removed.

Recovery will see the following changes:

- clear orphans stuff should delete all objects beyond last known on MDT. This is going to be done the following way:
  - after recovery is finished on MDS, it sends clear orphans RPC to OST (in fact create with special flag);
  - with clear orphans RPC, MDT sends FID of its last known OST object;
  - OST knowing its last real created object and last MDT known object, using Object Index API iterates over objects between real created and last known and removes them. Thus, all OST objects beyond last known on MDT will be removed and thus, MDT synchronizes its objects vision with OST;
- clear orphans stuff should make sure that deleted objects will not be re-created by CROW after that.

6.4 Locking changes

Locking changes are the following:

- OST needs one more lock - create lock;
- its purpose is to synchronize object create vs. unlink and orphans delete paths.

6.5 Disk format changes

As OST now has to create objects EA and save DT and MD FIDs there, OST store should be formatted such a way to take this into account, that is, EAs should be stored in same block on disk as inode to have good performance.

6.6 Wire format changes

Implementing DLM approach for checking object's state may need to pass locks over wire in write RPCs. This will change size of RPC, will need to take care of endiang , etc.

6.7 Protocol changes

The following protocol changes are expected:

- implementing DLM approach for checking object's state may need to pass locks from MDS create RPC to client and then client should pass it to OST;
- MDT does not have to pre-create objects on OST in create time.

6.8 API changes

API changes are not substantial. OST should add one method - entry point for CROW stuff. It should be used from few places in OST to create object.

OSC will have a lot of pre-creation pools stuff removed.

6.9 RPCs order changes

RPCs order and RPCs origins is changed in create time. Former approach, in create time had RPCs order:

1. client->MDT;
2. MDT->OST.

CROW changes it to:

1. client->MDT;
2. client->OST.

7 Alternatives

The following alternatives are possible:

- there are possible alternatives about tracking objects state;
- should setattr make CROW to create OST objects? Seems no, because consequent first write will set correct ownership info anyway.

8 Focus for inspections

I think main focus in inspection should be given onto objects state tracking approaches.