High Level Design for CMD Rename Locking

Niu Yawei

2006/01/24

Contents

1 Requirements

- Correct locking: All components involved in rename operation should be locked in right order.
- Sane checking: Source & target of rename should be checked in case of loop creation.

2 Definition

- Ancestor first order: Linux VFS use this rule for metadata locking. *see Documentation/filesystem/directory-locking*
- Resource order: Lustre order the metadata locks by the pair (mds number, inode number).
- cross-mds rename: Source parent and target parent are _not_ on same mds.
- cross-dir rename: Source and target are _not_ in same directory.

3 Functional Specification

3.1 All operations follow ancestor first order

Lustre follow resource order locking currently, which has some disadvantages:

- After lock parent/child in resource order, we need relookup child to see if something changes during locking, if it does, we have to drop all locks and restart the loop of "lookup -> lock -> verify".
- In some rename case, we have to lock 4 locks orderly, which makes code complicated. (see enqueue_4ordered_locks())
- For cross-mds rename, we even need RPC to do child verification.
- It's hard to fit resource order locking into mdt/mdd/osd layering.

Linux VFS use ancestor first order for years, there aren't these disadvantages. So, ancestor first order is a proper choice for Lustre. If the two directories to be locked isn't ancestor-offspring relationship, resource order is applied.

Ancestor order locking must be followed strictly. If there are two parents need be locked in a rename, they are also need be locked in ancestor first order.

3.2 Serialize cross-dir rename

If we follow ancestor first order, cross-dir rename must be serialized in case of deadlock (*Documentation/filesystem/directory-locking*). To serialize them, we might need to invent some kind of cluster-wide global lock. For example, a DLM lock for a special resource id on mds0.

3.3 Ancestor checking

Some ancestor checking function is necessary for locking two parents: we need to find out which is ancestor and lock it first. Unfortunately, it's not so straightforward to find out the ancestor of a slave splitted directory and a cross-ref

directory. For slave directory, we need to keep the lustre id of master directory in it's ea, thus, we can find it's master by this id and go on ancestor searching from master directory. For cross-ref directory, we need to keep the real parent id in it's ea for parent lookup.
The master id in slave ea and parent id in cross-ref dir ea feature has been done in audit development, we can integrate it.

4 Use Cases

- Non-cross-dir rename, source and target has common parent:
  1. User rename /p1/d1 to /p1/d2; (d2 exists)
  2. Lock their common parent: p1;
  3. Lock children d1 and d2 in resource order;
  4. Do rename work;
  5. Unlock children d1 and d2;
  6. Unlock common parent: p1;
- Cross-dir rename, target parent is ancestor of source parent:
  1. User rename /p1/p2/p3/d1 to /p1/d2; (d2 exists)
  2. Lock the cluster wide global lock;
  3. Lock parents in ancestor first order: p1, then p3;
  4. Sanity check: d1 can't be ancesotr of d2, d2 can't be ancestor of d1;
  5. Lock children d1 and d2 in resource order;
  6. Do rename work;
  7. Unlock children d1 and d2;
  8. Unlock parents p3 and p1;
  9. Unlock global lock;
- Cross-dir rename, source parent and target parent is not ancestor-offspring relationship:
  1. User rename /p1/p2/d1 to /p1/p3/d2;
  2. Lock the cluster wide global lock;
  3. Lock parents p2 and p3 in resource order;
  4. Lock children d1 and d2 in resource order;
  5. Do rename work;
  6. Unlock children d1 and d2;
  7. Unlock parents p2 and p3;
  8. Unlock cluster wide global lock;

5 Logical Specification

5.1 Follow ancestor first order locking

All operations' locking schema should be changed to follow these rules:

- For locking a pair of parent/child (unlink, stat ...): Always parent first.
- For locking two parents (rename, link): If they are ancestor-offspring, ancestor one first. Otherwise, little resource first.
- For locking two child directories (rename): Little resource first. (they can't be ancestor-offspring)
- For locking two child files (rename): Any order is ok, because it has been serialized by parents' locking.

The rename procedure should looks like:

1. Client send rename PRC to the source mds (source parent reside in);
2. Take global lock and parent locks on source mds;
3. Lookup source and target. If the dest parent is on a remote mds, we need a RPC to do lookup on target;
4. Sanity check on the source and target;
5. Lock the source and target in resource order;
6. Do rename work;
7. Unlock source and target;
8. Unlock parents, then unlock global lock;

5.2 Serialize cross-dir rename

Each cross-dir rename should take the cluster-wide global lock at first. This lock is a dlm lock for an artificial res on mds0.

5.3 Ancestor checking

To determine wether two directories is ancestor-offspring relationship on mds, we need a mechanism to travel from the leaf directory to root:

- For normal (or master splitted) directory, get parent by '..';
- For slave splitted directory, get parent id from EA_MEA;
- For cross-ref directory, get parent id from EA_PID;

6 State Management

- The locking order change doesn't affect recovery.
- If rename and all other operations follow "ancestor first" locking order, and all cross-dir rename are serialzed, it'll be deadlock free. See proof in *Documentation/filesystem/directory-locking*.

7 Alternatives

Serveral ways to optimize the cross-dir rename:

- Take "per-mds" locks instead of global lock. (only lock the mds related to this rename)
- If source and dest have common ancestor, take the common ancestor lock instead of the global lock.

All these optimize way need verification after locking, if directory topology changes during lock, we need to drop locks and retry locking.

8 Focus for inspection

- Is there any locking case missed?
- Is it proper to get all 4 locks on source mds for cross-mds rename?
- Is there any better way to do remote child lookup?