LFSCK 3 MDT - MDT Consistency Solution Architecture

Created and last modified by nasf on May 28, 2014

1 Introduction

With LFSCK 1.5 arriving in Lustre* software version 2.4, we have namespace consistency verification for single MDT including the FID-in-dirent and linkEA check/repair. With Distributed Namespace (DNE) fully enabled, new consistency cases are introduced. The LFSCK work to date is insuffecient to assure namespace consistency on single MDT. LFSCK must verify the global namespace consistency across multiple MDTs. Compared with single MDT, there are two main namespace changes related with DNE:

- **Remote Object:** The file (or directory) name entry and the corresponding MDT-object reside on different MDTs. The MDT-x holds the file name entry in the parent directory, and the name entry references the MDT-object with the FID-in-dirent or FID-in-LMA (XATTR_NAME_LMA) EA (of local agent). The MDT-y holds the file's MDT-object, and it points back to the remote name entry and the parent FID with linkEA (XATTR_NAME_LINK).
- **Striped Directory:** With DNE a directory can be striped to multiple MDTs. A striped directory contains one parent MDT-object and 1-N children MDT-objects; the parent MDT-object stores the stripe information as sub-directories and each sub-directory references one MDT-object; the child MDT-object references its parent MDT-object via the ".." entry.

In LFSCK 3, we will both extend the FID-in-dirent and linkEA verification from single MDT to cross-MDTs cases, and include additional inconsistent check/repair for remote object and striped directory. These additional tests include:

- **Dangling name entry:** The name entry exists, but related MDT-object does not exist.
- **Orphan MDT-object:** The MDT-object exists, but there is no name entry to reference it.
- **Multiple-referenced name entry:** More than one MDT-objects point back to the same name entry, but the name entry only references one of them.
- **Unmatched name entry and MDT-object pairs:** The name entry references the MDT-object which has no linkEA for back-reference or it points back to another name entry that does not exists or does not reference the MDT-object.
- **Unmatched object's type:** The file type in the name entry does not match the type which is claimed by the MDT-object.
- **Invalid nlink count:** The MDT-object's nlink count does not matches the name entries (which reference such MDT-object) count.
- **Invalid name hash for striped directory:** The name entry is corrupted and the name hash for striped directory does not match the LMV EA. Verify that the name entry is hashed to the MDT correctly.

In LFSCK phase 1.5, we implemented the functionality of scanning the single MDT device via low layer object-table based iteration and namespace based directory traversal. In this phase, we will reuse and extend this framework and iterate on all MDT devices in parallel.

2 Use Cases

2.1 Start/stop MDT-MDT consistency check/repair through userspace commands

An administrator can perform MDT-MDT consistency routine check/repair through lctl command. The task can be stopped/paused by lctl command by the administrator.

2.2 Monitor MDT-MDT consistency check/repair

An administrator can view the current LFSCK information for MDT-MDT consistency, includes status, speed, progress, statistics, and so on, through a lproc interface.

2.3 Resume MDT-MDT consistency check/repair from the latest checkpoint

Demonstrate that the MDT-MDT consistency check/repair will begin from the latest checkpoint by default when it is restarted, in spite of the MDT crash or the former LFSCK task failed or was stopped/paused.

2.4 Rate control for MDT-MDT consistency check/repair

An administrator can specify the highest speed for MDT-MDT consistency check/repair when start it by a lctl command. The rate limit can be viewed and adjusted during the LFSCK processing with a lproc interface.

2.5 Repair dangling name entry

The missed MDT-object should be created, or the dangling name entry should be removed, or the dangling name entry will be kept there with some error reported (by default), depends on the start options.

2.6 Repair orphan MDT-object

The unreferenced MDT-object should be found, and if it contains valid linkEA, then inserts related name entry to the namespace, otherwise generates new name entry and inserts it into .lustre/lost+found/MDTxxxx/.

2.7 Repair multiple-referenced name entry

The LFSCK should create new name entry under ./lustre/lost+found/MDTxxxx/, and the unrecognised MDT-object should reference the new created name entry.

2.8 Repair unmatched name entry and MDT-object pairs

The broken name entry and MDT-object paris should be recovered to reference each other.

2.9 Repair invalid file type

The invalid file type continued in the name entry should be fixed to match the type which is claimed by the MDT-object.

2.10 Repair invalid nlink count

The invalid nlink count should be updated to match the real name entries count.

2.11 Repair invalid name hash for striped directory

It should be recovered back to origin valid name, or relocate the name entry to the right MDT.

2.12 The Lustre system is available during the LFSCK for MDT-MDT consistency check/repair

The LFSCK for MDT-MDT consistency can be done during the external services running. Except for the target is corrupt and has not been fixed yet, other normal operations can be processed as without MDT-MDT consistency check/repair cases. The performance may be affected, but the correctness should not.

3 Solution Requirements

3.1 Online MDT-MDT consistency check/repair

LFSCK for MDT-MDT consistency is time-consuming. For large system with many petabytes of capacity and billions of directories and files, consistency check/repair may take days to complete. A production file system cannot be expected to be unavailable curing consistency check/repair. LFSCK must run on-line while the file system is available to clients and normal operations complete without significant disruption.

3.2 LFSCK user space control

LFSCK must have controls in user space so that it can be launched periodically during system usage. LFSCK user space controls were

implemented in LFSCK Phase I. Any new controls required by LFSCK for MDT-MDT consistency will be implemented within this existing control framework.

3.3 Support to resume from break point

As an online Lustre* tool, LFSCK may run for an extended duration. To ensure a LFSCK full system scan is robust to temporary server node failure (caused by power loss for example), a mechanism for resuming LFSCK from break point (or latest checkpoint) must be provided.

3.4 Rate control

LFSCK for MDT-MDT consistency may affect the performance of other Lustre operations, though it is unclear how much the impact will be at this time. A rate control mechanism will be added to enable an administrator to adjust the LFSCK performance. Rate control was implemented in LFSCK Phase I and a new LFSCK component for MDT-MDT consistency be added to the rate control framework.

3.5 No file/object lost

In any inconsistent case, LFSCK should not remove the inconsistent file/object by itself unless instructed to do so by the system administrator. If LFSCK is unable fix the MDT-MDT inconsistency (i.e. orphan MDT-object) it should keep the object in the system under the directory .lustre/lost+found/ as we did in LFSCK phase 2.

4 Solution Proposal

4.1 Repair strategy for MDT-MDT inconsistency

Identifying and resolving the inconsistent name entry and the MDT-object for 7 cases enumerated in the introduction.

4.1.1 Dangling name entry

We will allow the administrator to specify how to handle the dangling name entry when he/she starts the LFSCK: remove the dangling name entry, or create the missed MDT-object, or keep the dangling name entry there with some error reported.

4.1.2 Orphan MDT-object

If the orphan has no linkEA or the name entry corresponding to the linkEA has been occupied by other, then insert new name entry under the .lustre/lost+found/MDTxxxx/ locally; otherwise, add the name entry back to original namespace.

4.1.3 Multiple-referenced name entry

For the unrecognised MDT-object, generate a new name entry and inserts it under ".lustre/lost+found/MDTxxxx/" locally and update the linkEA entry.

4.1.4 Unmatched name entry and MDT-object pairs

Generally, the name entry is trusted because it is directly visible to the namespace, and the linkEA is primarily used for back-parse from FID to pathname. So the name entry will be trusted as long as it does not break the nlink count rules.

4.1.5 Unmatched object's type

The LFSCK will trust the type which is claimed by the MDT-object, and update the name entry.

4.1.6 Multiple referenced MDT-object - invalid/redundant linkEA entries, invalid nlink count

Assume: there are 'L' name entries reference the same MDT-object which has 'M' linkEA entries and 'N' nlink count. Normally, it should be "L == M == N" except that 'M' reaches to the max value because of EA size limitation. The later case is very rare, and once happened, the LFSCK will skip the verification for M. Otherwise: after the first-stage and the second-stage scanning (check/repair), the 'L' will not be greater than 'M', and for a normal file, the 'L' will also not be greater than 'N'; during the third-stage scanning, the invalid and repeated linkEA entries will be removed; and then if the 'M' (or the 'N') is still greater than 'L', then trust either 'M' or 'N' depends on which one is smaller: the redundant linkEA entries will be removed firstly if needed ('M'-- until be equal to 'N'), then the lost linkEA entries will be added back ('L'++ until equal to 'M') to namespace, finally reset 'N' as 'M' if they do not match.

4.1.7 Invalid name hash for striped directory

The LFSCK instance on the master (MDT-object) will verify that the master MDT-object only contains the shard sub-directories. The LFSCK instance on the slave (MDT-object) will verify the name hash for all its children name retries: If some name hash does not match the LMV EA, then either the name entry corrupted or the name entry was inserted on the wrong MDT. Via checking the MDT-object which is referenced by the FID of the name entry, we can know whether the name entry is still valid or not: if the name entry is still back-referenced by some MDT-object, then move the name entry to the right MDT; otherwise, depends on the start options, either keep the bad name entry there (by default) or remove it.

4.2 Race control between MDT-MDT inconsistency check/repair and other operations

The basic policy for the race control is to avoid the impact on other normal operations, especially the correctness, should not be affected. LFSCK is designed with the intention to minimize the number of locks taken, and try to avoid holding locks on one MDT-object and check on another MDT. If finds some inconsistency to repair then the lock mechanism used by DNE for modification will be used.

4.3 System scanning

Two-stage scanning will be employed for the MDT-MDT consistency check/repair:

1. The first stage scan will be driven by the LFSCK engine on the MDT with low layer object-table based iteration and namespace based directory traversal.
2. The second stage scan will iterate the lfsck trace file (which is an index file) for multiple-linked files (and remote orphans) verification.

NOTE: The local ldiskfs e2fsck or ZFS checker should guarantee that there is neither local dangling nor local orphan, then the LFSCK can focus on the global (remote) dangling/orphan.

4.4 Trigger strategy for MDT-OST consistency check/repair

In previous LFSCK phases we implemented a user space tool to control LFSCK start/stop. In this phase, we will enhance this tool to use "-t namespace -A" for the MDT-MDT consistency verification. The administrator can control the LFSCK manually, or develop scripts with these commands and trigger them periodically with crond or atd (or similar timer). The default behaviour of the "lctl lfsck_start" will trigger all known LFSCK components.

4.5 On-disk files changes

LFSCK phase 3 will reuse and extend the existing “lfsck_namespace” file on the MDT to trace the MDT-MDT LFSCK processing. The LFSCK status, progress, statistics, etc will be recorded in this file. The file “lfsck_namespace” will be updated (in memory) for every object check/repair. For performance, the updated “lfsck_namespace” file will only be synced to disk periodically. For each sync to disk, a new checkpoint will be written that includes the scan position. If the MDT crashs during the LFSCK, when the MDT restarts, the LFSCK can be resumed automatically or manually from the latest checkpoint in the file “lfsck_namespace”. At most one sync cycle of work may be lost due to the crash.

NOTE: In the case of upgrade from LFSCK 1.5 the "lfsck_namespace" will use new magic. This will ensure that when the MDT is upgrade from (or downgrade to) old lustre-2.x (x <= 6), the "lfsck_namespace" magic will not be recognized, so the the "lfsck_namespace" tracing file will be reset. This will cause the LFSCK to re-start from the beginning, but the burden is acceptable since upgrade/downgrade is rare.

4.6 APIs changes

LFSCK phase 3 has no strong requirements to change the existing APIs, except to enhance some APIs input parameter, such as dt_index_operations::dio_insert(), its parameters should contain the target MDT-object's type (especially for remote MDT-object) instead of assuming it is directory object. The detail will be described in the design document.

4.7 Wire protocols changes

LFSCK phase 3 will mainly use LFSCK_NOTIFY/LFSCK_QUERY RPC for LFSCK control and use related RPC stubs in OSP as DNE does to perform check/repair. We may add some lfsck_events/lfsck_event_flags within the LFSCK_NOTIFY/LFSCK_QUERY RPC formwork to enhance the LFSCK control. The detail will be described in the design document.

4.8 Dependency or requirements for DNE

Currently, DNE is not full implemented yet, there are some dependency and requirements:

1. Implement striped directory as sub-directories mode, and adjust LMV format. It is better to land this one to lustre-2.6, otherwise we have to handle the compatibility issues.
2. Remote object for normal file.
3. Cross-MDTs link/rename.

5 Acceptance Criteria

The acceptance test will be performed with code running on Lustre master branch. The LFSCK for MDT-MDT consistency will be accepted if meets the following requirements:

1. All Use Cases are demonstrated.

Like Be the first to like this

2 Comments

Nathan Rutman

Invalid name hash for striped directory – does this cover the case where the name hash does not match the actual MDS where the child MDT object actually lives? I.e. mismatch between recorded and actual child object MDT #?

nasf

For a file under a striped directory, the LFSCK 3 will verify whether its stripe hash (name to index) matches the MDT (where the name entry resides on) or not. And for the file itself, its name entry and its MDT-object can also reside on different MDTs.