# LFSCK 2 MDT - OST Consistency Solution Architecture

**Signed-off**
This document was agreed and signed-off by the PAC on 2013-05-22.

Rev 2 was agreed on 2013-05-29. This revision clarified MDT-OST consistency self-checking, system scanning and included a OST upgrade test.

## 1 Introduction

Normal files on a Lustre* file system (i.e. non-directory) are composed of one MDT-object (the *parent)* and zero or more OST-object(s) (or *childre n*). The parent resides on the MDT, and records the file layout information for the children belonging to the file. With the file layout information, client can locate the specified OST-object. To ensure data integrity, each child on related OST also records its parent identifier information to indicate which file the OST-object belongs to. Under normal operation, the file layout information stored on the parent will be consistent with the parent identifier information stored in its children. With a production system however, a error/failure condition may arise that can cause the parent-child pointers to become inconsistent. The inconsistency includes the following cases:

- **Dangling reference**. The MDT-object1 claims that OST-object1 is its child, but on the OST OST-object1 does not exist, or not initialized (and does not recognize MDT-object1 as its parent).
- **Unreferenced OST-object**. The OST-object1 claims that MDT-object1 is its parent, but on the MDT MDT-object1 does not exist, or is not initialized (and does not recognize OST-object1 as its child).
- **Mismatched reference**. The MDT-object1 claims that OST-object1 is its child, but OST-object1 claims that its parent is MDT-object2. On the MDT, MDT-object2 doesn’t exist, or it doesn’t recognize OST-object1 as its child.
- **Multiple references**. The MDT-object1 claims that OST-object1 is its child, but OST-object1 claims that its parent is MDT-object2 rather than MDT-object1. The MDT-object2 recognizes OST-object1 as its child.

A further type of consistency between MDT and OST is concerned with quota. Each object, both MDT object and OST object, has ownership information (UID and GID) to indicate which user the file/object belongs to. If the owner information for the MDT-object and OST-object(s) belonging to the same file are inconsistent, then quota will be inaccurate.

To fix the MDT-OST inconsistencies listed above a scan of the whole system is needed including both the MDT and OSTs. The existing object-table based iteration, implemented in LFSCK Phase I, will scan the whole system coupled with OI scrub and other LFSCK components for other system consistency check/repair.

## 2 Solution Requirements

### 2.1 Online MDT-OST consistency check/repair

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

### 2.2 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-OST consistency will be implemented within this existing framework.

### 2.3 Support to resume from break point

As an online Lustre tool, LFSCK may run for a extended duration. To ensure a LFSCK full system scan is robust to server node(s) restarting (possibly caused by a system crash), a mechanism for resuming LFSCK from break point (or latest checkpoint) must be provided.

## 2.4 Rate control

LFSCK for MDT-OST 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-OST consistency be added to the rate control framework.

## 2.5 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 the LFSCK does not know how to fix the MDT-OST inconsistency (i.e. unreferenced OST-object,) it should keep the object in the system, and make it visible to the administrator (for example, by link it into the directory "lost+found".) The administrator can process review these failures manually and review for example owner and content information to assist recovery.

# 3 Use Cases

## 3.1 Lustre MDT-OST consistency routine checking

Just as routine fsck is recommended for local file systems, the administrator should perform periodic Lustre consistency check to identify and resolve inconsistency issues as early as possible. The MDT-OST consistency (including both file layout consistency and owner consistency) check/repair is part of such routine check.

## 3.2 Lustre MDT-OST consistency self-checking

The MDT-OST consistency check/repair may run for a long time, and may take the routine checking very long time to identify and fix a rare inconsistency. As a result, the administrator may prefer not to schedule the LFSCK frequently, and instead rely on inconsistency to be identified and resolved during normal Lustre operations as follows:

The object-based RPC from client to OST contains the related MDT-object's identifier (FID) and stripe attribute information. The RPC service thread on the OST will use this information to compare with the target OST-object. If an inconsistency is revealed, the OST will send a RPC to the MDT to identify that either the client given information is incorrect or the information stored in the OST-object is incorrect. If the OST-object is incorrect the OST will try to repair the inconsistency. If multiple errors are detected the LFSCK will be triggered for the entire system MDT-OST consistency check/repair. The threshold of errors detected before starting full LFSCK can be controlled by a /proc tuneable.

# 4 Solution Proposal

## 4.1 Repair strategy for file layout inconsistency

Identifying and resolving the inconsistent file layout between MDT-object and OST-object(s) for above four cases (listed in the Introduction).

### 4.1.1 Dangling reference

A parent has a dangling reference. There are two cases for dangling reference:

1. Former allocated OST-object is lost. A new OST-object must be allocated with the specified object external identifier (FID or IDIF) and initialized with the parent identifier (FID). If such OST-object has never been written before, then above repair is complete. Otherwise, the data from the missing object is lost.  Recovering such lost data is out of this project scope.
2. The OST-object is present, but it is not initialized, and without SUID + SGID mode set. The OST-object will be initialized with the given parent.

### 4.1.2 Unreferenced OST-object

There are two cases for unreferenced OST-objects:

1. Former allocated MDT-object is lost. The LFSCK does not know if the unreferenced OST-object is useful or not. The LFSCK will recreate the missed MDT-object with the specified parent identifier and initialize it with the given child information. In addition, a special directory on the MDT, like lost+found, is required to hold those files with recreated MDT-objects. Without this directory the file will be invisible.

Next, the administrator can manually process the files under lost+found/ with more human knowledge, like file content, owner, and so on, to determine whether the files should be move into the production namespace, partly copy data and compose new file, or unlink them. A /proc tunable will be available to change the LFSCK behaviour to delete unreferenced objects instead of linking them into lost+found. OST-objects without any data will be deleted, since there is no benefit in recovering them.

2. The MDT-object is present, and the file can be found in the namespace, but the MDT-object has lost its object layout. We can initialize it with found OST-object(s).

Above inconsistency resolutions are limited to the local view. If one considers the global perspective, the cases will be more complex. For example, the MDT-object1 claims that the OST-object1 is its child, but on the OST, the OST-object1 does not exist (either unallocated or uninitialized). On the other hand, the OST-object2 on the same OST claims that the MDT-object1 is its parent, but the MDT-object1 does not recognize the OST-object2 as its child. The MDT-object1 and OST-object2 probably belong to the same file. So we will try to combine the MDT-object1 which has dangling reference and the non-referenced OST-object2 which claims to be referenced by the dangling MDT-object1 together. The strategy is that: for non-referenced OST-object, if related MDT-object has new created OST-object for fixing dangling reference, but nobody has written the new created OST-object, then replace the new created OST-object with the non-reference OST-object. Otherwise, the non-referenced OST-object will be regarded as orphan and processed as described in 4.1.2.

### 4.1.3 Mismatched reference

An unmatched referenced MDT-object and OST-object pair:

The LFSCK cannot guarantee that the unmatched MDT-object and OST-object belong to the same file or which one is correct. In such cases the MDT-object information will be used, which means that the OST-object will be updated to recognize the new parent. The “new” file is still in the same position in the namespace. The MDT-object layout EA is trusted over the OST-object back-pointer(s) because it relates to user visible file data, while the OST-object back-pointers are only used for internally recovery purposes and are not visible to the user, so do not affect proper file usage information, nor were they kept consistent for Lustre 1.8.x MDT file-level backup/restore, because 1.8.x MDT-object identifier will be re-generated after MDT file-level backup/restore.

### 4.1.4 Repair multiple referenced OST-object

To guarantee unique OST-object references, the LFSCK will create new empty OST-object with a new external identifier, and the unrecognised MDT-object will reference the new created OST-object. The external identifier for the new OST-object can neither conflict with any existing object, nor to be reused by other. Full details will be understood once detailed design is completed.

## 4.2 Repair strategy for owner inconsistency

If the ownership information for the parent MDT-object is inconsistent with the child OST-object’s ownership information, then the parent is trusted. Given the chown/chgrp processing order is: client => MDT => OST, it is more likely that the OST-object owner information is stale rather than the MDT-object’s. In addition, the MDT-object’s owner information is visible to users, while the OST-object’s owner information is only used internally by quota. The resolution is to synchronize OST-object owner information according to the MDT-object’s owner.

## 4.3 Race control between MDT-OST consistency 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. For this reason, LFSCK will rollback some reparation activities when race with a normal RPC process is detected. This will avoid holding locks on the MDT-object during related OST-object processed on the OST. Another aspect of LFSCK scanning is cache pollution: new objects fetched by the scanner and to be used once will purge other useful objects from the cache. A new cache strategy may be used to avoid such a case.

### 4.3.1 File layout consistency check/repair and create

To repair the file which parent has dangling reference, we need to initialize the uninitialized OST-object. During such process, we need to identify the uninitialized OST-object from normal newly created OST-objects that have never been modified. The “SUID+SGID” mode is the basis for this check.

To repair the repeated referenced OST-object, the LFSCK needs to create new OST-object with new external identifier. The new external identifier cannot be reused by other create operations. It can be controlled on the create path by the MDS: the MDS transfers the non-used pre-created OST-object identifier to the OST for such creation.

### 4.3.2 File layout consistency check/repair and unlink/chown

When the LFSCK is verifying (and potentially repairing) file layout consistency, it needs to handle the race of some others unlink/chown the target file during the LFSCK. It can use the existing OSD-level read-write lock (or ldlm lock) to prevent the target file to be unlinked/chowned before related LFSCK RPC sent out. We do not want the LFSCK to hold such lock when the LFSCK RPC in-processing on the OST, because it may cause some potential deadlock. So the LFSCK needs to do some rollback/cleanup work after the LFSCK RPC reply and the target file unlinked/chowned by others during the RPC processing on the OST.

## 4.4 System scanning

The LFSCK for MDT-OST consistency must scan all of the objects in the file system, not only on the MDT, but also on the OSTs. Ensuring that the scan of the file system is comprehensive and efficient is one of the key design goals of this project. The requirements include:

- **Completeness**. All MDT-objects and OST-objects (except for those newly created during the LFSCK run) should be checked to ensure all dangling reference cases and unreferenced cases can be found.
- **Efficiency**. System scanning may take a considerable time on a very large filesystem. The performance for different scanning methods may vary greatly. Generally, sequential scanning is most efficient. But from different layers, “sequential” definitions may be different. For example, namespace based “sequential” tree traversal is probably not actually “sequential” for the backend filesystem format. For the MDT-OST consistency check/repair, backend filesystem based “sequential” is preferred. Object table based iteration was implemented during LFSCK Phase I, that is backend filesystem based “sequential” scanning. Object table based iterations should be used for the MDT-OST consistency check/repair.

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

1. The first stage scanning will be driven by the LFSCK engine on the MDT with low layer object-table based iteration on the MDT device, which will send RPCs to the OSTs for check/repair the inconsistency, including the MDT-object with dangling reference, unmatched referenced MDT-object/OST-object pairs, multiple referenced OST-object, inconsistent owner information, and so on. In contrast to single device scanning, the scanning on the OST will be affected by the MDT. The LFSCK thread on the MDT can scan the MDT device sequentially. For each MDT-object the LFSCK thread must verify its children on the OSTs. Such check/repair order is expected to be random for the OSTs because of mixed create/unlink operations and due concurrent scanning from multiple MDTs, when DNE is employed in the future. Since MDT is the LFSCK controller, and the most constrained resource in the file system, the MDT scanning will be sequential. Random scanning will be tolerated on the OSTs. The first-stage scanning on different MDTs can run in parallel. Each MDT just scans its local namespace, the global namespace consistency (for DNE cases) will be resolved in LFSCK phase III.
2. The second stage canning will be driven by the LFSCK engine on the OST with low layer object-table based iteration on the OST device. For the OST-objects that have not been scanned or accessed during the first stage scanning, a RPCs will be sent to the MDTs to check/repair unreferenced OST-objects. As an optimization, the LFSCK may skip the files that have been accessed successfully within last N hours/days. On MDS_CLOSE the client can report if all stripes were accessed to verify on-the-fly. On OST side we may mark objects accessed with regular RPC and there by let LFSCK skip them. The second-stage scanning can run on all OSTs in parallel.

### 4.4.1 Search for unreferenced OST-object

MDT-driven scanning of OST objects is not sufficient for complete correctness because OST-objects that were not referenced by the MDT must also be discovered. To achieve this OST-objects that were not referenced during the MDT scan are tracked and verified that they do not have corresponding MDT-objects.

To discover the unreferenced OST-objects it is necessary to know if the OST-object is referenced by a MDT-object or not. This is a challenge! The simplest way is to query the MDT with the parent identifier. This will cause additional random scanning on the MDT, which is undesirable. In fact, during the first cycle random scanning (driven by the MDT side sequential iteration) on the OST it is already known related OST-objects have been referenced by related MDT-objects. These OST-objects can be recorded to avoid query the MDT again. In addition, for OST-objects that are accessed by normal RPCs (read/write/truncate) from clients during the LFSCK, they are also referenced by related MDT-objects (otherwise, clients cannot know how to access them). These OST-objects should also be recorded. With this approach for discovering unreferenced OST-objects, it is only necessary to query the MDT for the OST-objects that are never accessed during the LFSCK.

There exists a challenge to record the accessed OST-objects during the LFSCK. The current solution is that the LFSCK can maintain an OST-objects bitmap on the OST, and update the bitmap to record OST-objects accesses. After the first cycle random scanning driven by MDT-side sequential iteration all the accessed OST-objects’ external identifiers are recorded in the bitmap. Another sequential scanning through object table based iteration on the OST is performed. For each OST-object, attempt to discover it in the bitmap. If it cannot be found, query the MDT to discover if it is unreferenced or not, and repair it when needed.

To simplify the design and accelerate normal LFSCK process, such OST-objects bitmap is maintained by the OST in RAM only, without storing on the device. The potential risk is recovery process for OST crash during the LFSCK. Because without storing the bitmap on the device, if the OST crashes, it will lose the record for those former accessed OST-objects. In this case it has to query the MDT for those former accessed

OST-objects although they are unprofitable.

For a large cluster with a large number of OSTs, it is not unexpected that some OST(s) may fail during the LFSCK. If only single OST failed during the LFSCK, it is not considered problematic to make the failed OST re-join the LFSCK processing after recovery. If multiple OSTs fail during the LFSCK then these failed OSTs re-joining the LFSCK processing will generate significant random IO on the MDT. To avoid this potential performance degradation, a failed OST(s) is prevented from re-join the LFSCK processing. To guarantee the LFSCK completeness, a subsequent LFSCK will be triggered sometimes later.

## 4.5 Resume from latest checkpoint

Introduce a new local file, named as “lfsck_layout” on the MDT, to trace the LFSCK processing. The LFSCK status, progress, statistics, and so on, will be recorded in such file. The file “lfsck_layout” will be updated (in memory) for every object check/repair. But for performance, the updated “lfsck_layout” will be synced to disk periodically. The default sync interval (or sync cycle) is 60 seconds. For each time syncing to disk, a new checkpoint will be written to disk that includes the scan position. If the MDT crashed during the LFSCK, then after the MDT restarts, the LFSCK can be resumed automatically or manually from the latest checkpoint in the file “lfsck_layout”. At most one sync cycle of work may be lost due to the crash.

## 4.6 Trigger strategy for MDT-OST consistency check/repair

### 4.6.1 Periodically or manually triggered from userspace

As part of the work for the LFSCK user space tools, lctl commands to control the kernel space LFSCK (start/stop) from user space will be implemented. A administrator can make scripts with these commands, and trigger them periodically with crond or atd (or similar timer).

### 4.6.2 Auto trigger LFSCK if detects any MDT-OST inconsistency during normal Lustre operations

Introduce new switch to enable/disable MDT-OST consistency verification during normal RPC processing. If found related inconsistency under enable mode, it will try to fix the inconsistency or trigger the LFSCK when needed.

## 4.7 Rate control

For the LFSCK process on the MDT, the scanning speed can be controlled by the existing rate control mechanism that is implemented in former LFSCK phase. As for the LFSCK process on OST, rate control is divided into two:

1. The random device-scanning phase, driven by the LFSCK process on the MDT. Scanning speed can be controlled by the rate control on the MDT indirectly.
2. The sequential device-scanning phase, driven by itself on the OST. Since it will reuse the existing object table based iteration, scanning speed can be controlled by the existing rate control mechanism directly on the OST.

## 4.8 Wire protocol changes

There will be wire RPC protocol changes for the MDT-OST consistency check/repair, including:

1. MDT uses OST_SET_INFO RPC to notify OST the LFSCK events: start, stop, internal synchronization, and so on. We need new keys for the new purposes.
2. OST uses MDS_SET_INFO RPC to notify MDT the LFSCK events: status changes, success/fail, and so on. We need new keys for the new purposes.
3. MDT uses enhanced OST_GETATTR RPC (or new OST_CONSISTENCY_VERIFY RPC, with parent MDT-object FIDEA, owner information in the request) for related MDT-OST consistency check/repair. If a related inconsistency was found, then the OST will try to repair the inconsistency without additional RPC(s) between MDT and OST. For example, inconsistent owner information, MDT-object with dangling reference, unmatched referenced MDT-object and OST-object pair, and so on.
4. OST uses enhanced MDS_GETATTR RPC (or enhanced MDS_REINT RPC, with child OST-object identifier, owner information in the request) for related MDT-OST consistency check/repair. And if related inconsistency was found, then the MDT will try to repair the inconsistency without additional RPC(s) between MDT and OST. For example, create the missed MDT-object for the unreferenced OST-object, or extend the existing MDT-object layout EA for the unreferenced OST-object, and so on.
5. MDT uses enhanced OST_CREATE RPC to create the specified OST-object on the OST.

# 5 Unit/Integration Test Plan

## 5.1 Start/stop MDT-OST consistency check/repair through userspace commands

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

## 5.2 Monitor MDT-OST consistency check/repair

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

## 5.3 Resume MDT-OST consistency check/repair from the latest checkpoint

Demonstrate that the MDT-OST 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.

## 5.4 Rate control for MDT-OST consistency check/repair

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

## 5.5 Repair file which parent has dangling reference

The missed OST-object should be recreated, and the uninitialized OST-object should be initialized.

## 5.6 Repair unreferenced OST-object

The unreferenced OST-object should be found, and related MDT-object should be recreated, and linked into lost+found directory for further process if it does not exist in the namespace before.

## 5.7 Repair unmatched referenced MDT-object and OST-object pair

The unmatched referenced MDT-object and OST-object should compose the new matched reference pair within the same file.

## 5.8 Repair repeated referenced OST-object

The LFSCK should create new OST-object, and the unrecognised parent should reference the new created OST-object.

## 5.9 Repair inconsistent file owner information

The owner information recorded by the OST-object should be synchronized with its parent on the MDT.

## 5.10 Handle the OST upgrading from Lustre-1.8

For Lustre-1.8 OST, the MDT-object's IGIF is stored by the OST-object. When the MDT performs file-level backup/restore, the IGIF cannot be preserved. As a result, a restored OST-objects will have stale MDT-object information. The LFSCK will find out those OST-objects have stale information and repair them.

## 5.11 The Lustre system is available during the LFSCK for MDT-OST consistency

## check/repair

The LFSCK for MDT-OST 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-OST consistency check/repair cases. The performance may be affected, but the correctness should not.

## 6 Acceptance Criteria

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

1. All test scenarios are demonstrated.

---