OI Scrub and inode Iterator Solution Architecture

Introduction

Unknown macro: 'html'

Lustre is entering it’s second decade as a production high performance parallel filesystem. Lustre has grown and adapted to continual demands to increase performance while maintaining production quality stability. With the advent of the 2.x release series, a number of features were introduced to lay the foundations for another decade of Lustre scalability. This document describes the File Identifier (FID) feature and presents a tool to automate consistency checking and repair of the Lustre filesystem metadata.

In Lustre 2.x, File Identifier (FID) is the unique identifier for file or object. It differs from traditional inode number/generation and is independent from the backend filesystem. Internal to the Lustre Object Storage Device (OSD), an Object Index (OI) table is used to map the FID to the local inode number.

One scenario where the inode number will be changed is if the MDT is restored from file-level backup. In this case, the OI table that is restored from backup will be stale, since the inode number/generation will be newly allocated by the backend filesystem for each file. The OI table can be rebuilt by iterating through all inodes and correcting the FID-to-inode mapping in the OI. In addition, the ability to iterate through inodes and cross-checking the OI against the filesystem is useful to maintain filesystem consistency.

Today, a large MDT may have hundreds of millions inodes. The number of inodes may reach billions in the future. While the Inode Iterator will be optimized for efficient IO on the MDT, the task may take considerable time. To make Inode Iterator and OI Scrub useful on a large MDT the tool must run online (i.e. while the filesystem is mounted and in use). Clients may access the MDT during the OI Scrub rather than having to wait for it to complete. Performance of some Lustre operations may be affected by an active, online OI Scrub, but correctness will not.

For Phase I of the new Lustre Filesystem Check (LFSCK) suite, the Inode Iterator will be provided as component in kernel space against single OSD, initially the MDT. More functionality will be implemented when the DNE (Distributed Namespace) feature is introduced in the future. This will be a common foundation for subsequent LFSCK functionality. As the first valuable function of LFSCK, OI Scrub will use the Inode Iterator to scan all inodes in the MDT filesystem to check and update the OI table. OI Scrub will include both kernel thread(s) and user space tool(s) for control.

Lustre Filesystem Check suite is a project with a multi-phase development and deployment schedule. Each phase of this project will have a detailed design available to the community for feedback. Code will be prepared for up-stream landing by demonstrating rigorous testing and publishing performance benchmarks and review. By developing a new high-performance inode iterator, duplication of existing work will be avoided. Where expedient, existing solutions and designs will be employed to accelerate delivery.

Solution Requirements

MDT is available after file-level backup/restore

Providing the ability to perform file-level backup and restore is the core to the usefulness of this phase. The OI Scrub will update the OI table and verify all FID-to-inode mappings. As an online tool, the OI Scrub should also guarantee that MDT is available (clients can access the Lustre filesystem without interruption) and provide correct behaviour to clients irrespective of whether the OI Scrub is complete.

However, there is one exception: a client-side RPC that involves FID-to-path (lustre_rsync) or accessing a file directly by FID (mainly for re-export of Lustre by NFS) in the case where the corresponding FID-to-inode mapping has not corrected by LFSCK yet. In this case, the OSD should return a special error code to notify the MDD that the FID is invalid and that it needs to resend the RPC at a later time. At this point, if the lookup is for a DLM lock it is possible to notify the client to wait until the lock can be granted (when the mapping is fixed for that FID). Otherwise, clients will block waiting for a reply from the MDT until the FID-to-inode mapping is updated for that FID.

OI Scrub user space control

The OI Scrub should support to be controlled from user space. OI Scrub may be launched periodically (to check for inconsistencies,) or manually after the MDT is restored from file-level backup. The MDT device should also be configurable to verify OI lookups: if an inconsistency is discovered a single entry may be rebuilt or a full scrub may be automatically launched.

Checkpoint support

As an online Lustre tool, the OI Scrub may operate for a prolonged duration. To ensure that the MDT scan checks all of the in-use inodes on a regular basis and can recover from interruptions, the Inode Iterator will periodically write a checkpoint record with its current state. This will allow the Inode Iterator to resume from where it left off, in case of interruption.

Rate control

As an online Lustre tool, the OI Scrub may affect the performance of other Lustre operations. Since the Inode Iterator is performing entirely linear reads from disk, it is expected to have a low impact on other MDS operations. However, until the tool is written and benchmarked it is unclear what the impact will be at this time. To enhance the value of Inode Iterator, and mitigate the risk of unknown behavior in production systems, a rate control mechanism will be included. The rate control will allow periodic background Lustre OI table scrubs to be run while avoiding significantly impacting the performance of other Lustre operations. The rate limit will be adjustable at runtime.

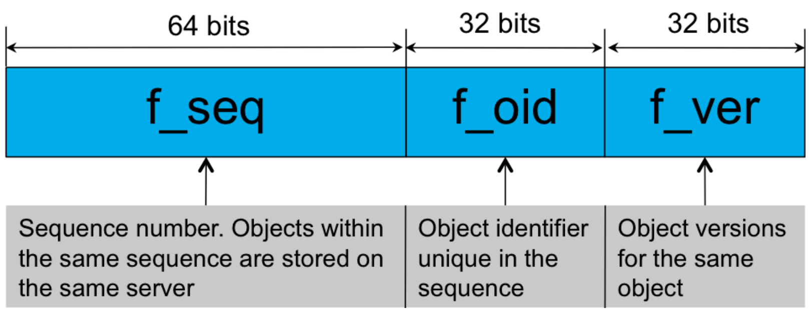
Operating status

An administrator should be able to view the current status of the inode iterator. This should include the current position and relative progress through the task and the rate limit. Visibility of the different components of the Lustre filesystem is critical to tuning behavior and tracking down performance regressions.

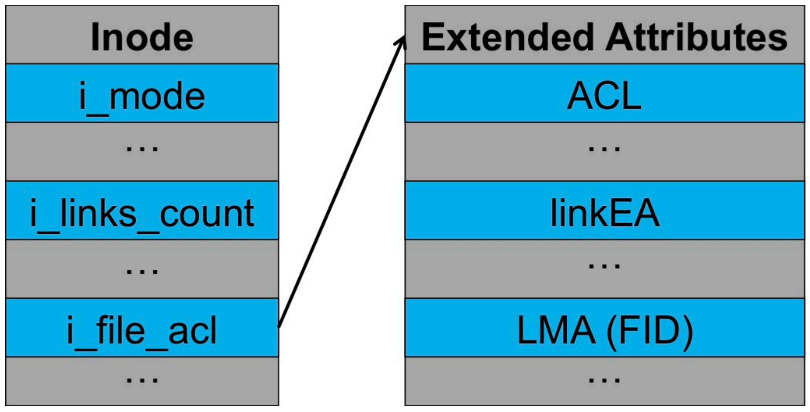
Solution Proposal

Overview of the OI Table

In Lustre 2.x, a File Identifier (FID) is the unique identifier for files and objects across an entire Lustre filesystem. A FID is a 128-bit structure and independent of backend filesystem. The FID is illustrated in the figure below. This design provides considerable advantages and is transparent to the filesystem clients. However, the MDT must map from a FID to the internal filesystem inode number where useful information is recorded. This task is called 'FID-to-inode' mapping.


64 bits
32 bits
32 bits
f_seq
f_oid
f_ver
Sequence number. Objects within the same sequence are stored on the same server
Object identifier unique in the sequence
Object versions for the same object

A FID is recorded in the extended attribute (called the Lustre Metadata Attribute, LMA) of an inode to allow mapping from the inode to the FID number, and for redundancy. The figure below illustrates the FID as a Lustre Metadata Attribute. In order to allow inode lookups by FID number, a table of FID-to-inode entries is maintained. This table is called the OI Table.



During a file-level backup, all the extended attributes are archived. When a file level backup archive is restored, new inodes are created, and the extended attributes are restored. At this point, records in the OI table (which is stored in the filesystem as a regular file) are incorrect as new inode numbers are allocated to the restored files.

The primary task of this project is to deliver a tool to verify the OI table. This process is known as OI Scrub. The core OI Scrub mechanism is comparison and correcting entries in the OI table with the FID value recorded in inode extended attributes.

Inode table based iteration

As described in the introduction, a valuable application of OI Scrub is updating FID-to-inode mappings after a file-level MDT restore. In the case of a restore, updating the FID-to-inode requires that all inodes are visited, and this task should be completed as quickly as possible. Inode table based iteration is a good choice to achieve these aims.

OI Scrub will scan the MDT inode table (according to inode bitmap) sequentially. For each inode, the FID is read from the LMA extended attribute. This FID is looked up in the OI table to locate the corresponding FID-to-inode mapping entry. If no entry is found, then a new entry is inserted. If the entry is found it is verified for correctness, or updated as necessary with the new inode number.

Two kernel threads are necessary to maximize the performance of this operation. One OSD thread performs the inode table iteration, which scans MDT inode table and submits inode read requests asynchronously to drive disk I/O efficiently. The second thread is the OI Scrub thread which searches the OI table and updates related mapping entries. The two threads run concurrently and iterate inodes in a pipeline.

The Object Storage Device (OSD) is the abstract layer above a concrete back-end filesystem (ext4, ZFS, Btrfs, and so on). Each OSD implementation differs internally to support concrete filesystems. In order to support OI Scrub the inode iterator will be presented via the OSD API as a virtual index that contains all the inodes in the filesystem. As much as possible, common interface calls will be chosen to implement inode table based iteration to avoid possible redundant work in the future. However, to simplify development and avoid external dependencies, the first phase of new LFSCK will be developed against the ext4/ldiskfs OSD.

Inode iterator and checkpoints.

Checkpoints will be provided by introducing a new file named `scrub_status`. This file will be created on the root of the MDT partition. The `scrub_status` file on the root is invisible to clients. The checkpoint will record the latest checkpoint position and other scrub parameters, and this record will enable restarting the scrub with only a minimum of repeated effort. Additional records will record the following:

- The time the scrub started, when `scrub_status` was last updated, and when it finished.
- The iterator type: either inode table based or namespace tree based (optional).
- The current position in the scrub: inode number for inode based iteration; or directory FID, filename, and readdir cookie for namespace tree based scanning.
- MDT mount options and system parameters related to the OI Scrub.
- OI Scrub statistics information, including number of items scanned, updated or ignored.

For performance reasons, it is prohibitive to update the `scrub_status` for each item scanned. Updating every iteration is inefficient and unnecessary, especially if the filesystem is in a consistent state already. The `scrub_status` will be updated opportunistically. For example, `scrub_status` will be updated at most once per minute or every 10000 items scanned by default, or at a tunable interval.

Each time `scrub_status` is updated, a new checkpoint is created and changes are committed to disk. As a result, even if MDT restarts during the OI Scrub scanning, only the inodes scanned since the latest checkpoint (at most one minute or 9999 items) will need to be repeated. The MDT can reload information from `scrub_status` after restart and continue to process from the latest checkpoint.

Statistics and tunables stored in `scrub_status` will be exported via procfs to userspace. This will allow access to the scrub statistics consistent with existing reporting for Lustre filesystems.

OI Scrub trigger strategy

Auto start if detects MDT file-level restore

It is desirable that the IO Scrub is triggered automatically after a file-level restore takes place. Detecting that a file-level restore has taken place can be achieved as follows: `scrub_status` can have its unique FID recorded in both the extended attribute of the inode and in the OI table. When the MDT mounts, the `scrub_status` file's FID is retrieved from the extended attribute and compared to the one in the OI table. If the `scrub_status` FID does not map correctly back to the same inode (match both the inode number and the inode generation), the error return will alert the MDT that the filesystem has been restored from a backup or the OI is corrupted and OI Scrub can begin.

In addition, the absence of the `scrub_status` file may also trigger OI Scrub. For example, when upgrading an MDT from an earlier version of Lustre 2.x to one that supports LFSCK (without backup/restore), the `scrub_status` file will be absent. In such cases, triggering the OI Scrub may be desirable because old version MDT does not support routine OI consistency checking and there may already be inconsistencies in the OI table.

Periodically or manually triggered from userspace

As part of the work for the OI Scrub user space tool, `lctl` commands will provide manual control of the kernel space OI Scrub. An administrator can schedule periodic IO Scrub scheduling from userspace with cron or at.

Administrative monitoring of the OI Scrub progress can be achieved by examining the `scrub_status` file.

Auto start if inconsistency is detected during OI lookup

A new switch will be introduced to enable/disable OI lookup verification. With this switch enabled the FID-to-inode mapping is checked for validity when OI lookup takes place. If an inconsistency is found, it will be repaired and optionally trigger the complete OI Scrub automatically. The switch will be controlled through the `lctl` command.

Recovery processing

The OI Scrub allows client-side RPCs to be processed during checking and updating the MDT. If an RPC triggers an OI lookup, but the corresponding FID-to-inode mapping has not been updated yet, then the mapping should be updated properly to ensure other subsequent FID based RPCs can be processed correctly and efficiently.

RPCs that perform a lookup-by-name which trigger a non-updating OI lookup that is found to be incorrect, but does not have a transaction open, cannot directly repair the OI FID-to-inode entry without the overhead of starting an additional transaction (which will cause interoperability issues for old clients during recovery). The action of repairing the OI table is delegated to the scrub thread, which can process such updating request as a priority. In such cases, the thread doing theOI lookup will be blocked until related updates have been committed.

As for filesystem-modifying RPCs, if they trigger an OI lookup which involves invalid FID-to-inode mappings, they can update related mappings in their own transactions, but would require additional transaction credits reservation. On the other hand, this may introduce some potential (but unnecessary) dependencies between modifying RPCs and non-modifying RPCs that involve the same invalid FID-to-inode mapping(s) during recovery. To make the process unique and simple, we prefer to do all RPC-driven OI table updates through the dedicated thread, and block the non-idempotent RPC reply until related OI updates have been committed.

Unit/Integration Test Plan

Test 1: System is usable after file-level restore.

The system can trigger the OI Scrub automatically during MDT mounts if it detects file-level backup/restore against new Lustre-2.x partition.

Test 2: Administrator can mount an MDT with or without OI Scrub enabled.

The OI Scrub can be controlled by mount options when MDT mounts: start the OI Scrub or not.

Test 3: Administrator can trigger OI Scrub with a mounted MDT filesystem.

The OI Scrub can be initiated controlled by the OI Scrub user space tool on a mounted MDT. Start/Stop the OI Scrub, verify OI lookup and rate control can be controlled by the administrator.

Test 4: Monitoring of a running OI Scrub.

The administrator can view the current state (started, stopped, rate, progress) of OI Scrub via /proc.

Test 5: OI Scrub can be performed with an online filesystem.

Client can access the MDT while OI Scrub is running. Except the operations of FID-to-path or accessing parent from non-directory child, other operations behave as normal.

Test 6: Clients can use a filesystem during OI Scrub.

When the OI Scrub checking and updating the MDT is complete after file-level backup/restore, all the FID-to-inode mappings should be valid. Client can access the MDT as does before the backup, including the operations of FID-to-path and accessing parent from non-dir child, no correctness issues.

Test 7: Restart interrupted OI Scrub from checkpoint.

Interrupt the OI Scrub during operation. Demonstrate that OI Scrub begins from the checkpoint when it is restarted.

Test 8: Rate of OI Scrub can be controlled at run-time.

Demonstrate the OI Scrub rate can be modified during operation.

Acceptance Criteria

The acceptance test will be performed against ext4-ldiskfs based Lustre-2.1+ MDT (single MDT only). Inode Iterator and OI Scrub will be accepted if meets the following requirements:

1. All test scenarios are demonstrated.

Glossary

ldiskfs: Lustre backend filesystem, this is a ext4 filesystem with additional patches applied.

MDT: Metadata Target, server component that manages the Lustre file system namespace.

OSD: Object Storage Device, the abstract layer on the top of backend filesystems, like ldiskfs.

Appendix

Use Cases

Online tool for MDT file-level backup/restore

An administrator wants to perform a MDT backup/restore for disaster recovery purposes. File-level backup/restore provides additional value for a system administrator who may employ this technique to migrate an MDT to a new device or to take advantage of new filesystem features.

The administrator performs the following steps:

1. Format the new MDT device with new parameters.
2. Mount the new empty MDT partition as local filesystem (e.g. ldiskfs) directly.
3. Restore the MDT data (including extended attributes) from former file-level backup (untar).
4. Remount the new MDT partition as Lustre for exporting services to clients.
5. OI Scrub rebuilds the OI table.
6. The administrator remounts the filesystem on the clients.

Lustre MDT consistency routine checking

The administrator routinely employs fsck on local filesystems as good practice against indeterminate errors. As a Lustre administrator the MDT OI table is also checked for consistency as a best practice. This increases the administrators confidence that the system is running smoothly and that the problems reported by users are due to their code.

Lustre MDT consistency self-healing

During normal operation on a Lustre filesystem a invalid entry in the OI table maybe discovered. This inconsistency may trigger an automatic OI Scrub of the entire table.

Inode iterator rate-limiting

The administrator has a large filesystem to maintain and the MDT has millions of inodes. The administrator want to run an OI Scrub but not disrupt normal operations. The administrator begins a OI Scrub with the rate-limit reduced so the users do not notice the background task. Controlling of the rate during runtime will be possible.

Inode iterator check-pointing

Inode iterator maybe interrupted. Inode iterator should be able to continue an interrupted activity with minimal duplication of effort.

Inode iterator status

An administrator should be able to view the current status of the inode iterator. This should include the current position and relative progress through the task and the rate limit.