LFSCK 4 Solution Architecture

Introduction

After successfully completing all previous components of the LFSCK project, Lustre file systems now have a complete solution for checking file consistency. LFSCK can now scale by running in parallel and supports DNE file systems. In addition, the success of the LFSCK development work now means file level backups are available as an additional option for system administrators. All of the new features have been documented with man pages and entries in the Operations Manual.

This final component of the LFSCK project is primarily concerned with LFSCK performance. During the development of LFSCK, a small number of performance enhancements were identified but were out of scope for the given contract phase. These optimizations were collected and this phase of the work is concerned with implementing them.

Requirements Description

LFSCK is a scalable consistency checker that can run on many file system components in parallel. LFSCK is designed to run on production file systems in an on-line and un-degraded state. LFSCK needs to:

- perform consistency checking without blocking any other normal file system functions.
- perform consistency checking without noticeably reducing file system performance.

Use Cases

1. A filesystem has a large number of unused inodes in the ldiskfs file system.

LFSCK does not scan inodes in unused block allocation groups or beyond the unused_inodes high water mark to minimize the amount of disk IO and contention with user workloads.

2. An administrator wants to avoid unnecessary scanning.

LFSCK will only trigger a full system scan after a threshold of inconsistent files are discovered.

If a small number of inconsistencies are discovered LFSCK should only repair individual objects that are found to be inconsistent and not run a full scan.

3. An user wants to access to files during LFSCK scanning.

LFSCK will, where ever reasonable, lock at the lowest granularity possible to safely perform consistency related operations and updates.

4. An administrator wants LFSCK to find inconsistencies as quickly as possible.

LFSCK will be run with memory vs disk storage for `lfsck_namespace` to characterize scanning performance for given workloads.

5. An administrator needs to know what options are available, and what they do.

LFSCK documentation will be reviewed and verified as complete.

Solution Proposal

Four independent development tasks will be created to achieve each of the use cases. These tasks will be executed in accordance with the published development guidelines and with peer review and automated testing. In addition, a performance measurement will be made on the completed implementation.

1. A filesystem has a large number of unused inodes in the ldiskfs file system.

For ldiskfs-based backend, the OI scrub currently scans the local device linearly. It iterates all the inodes on the ldiskfs partition in the inode tables

in each block group without distinguishing whether the block group that contains the inode table has been initialised or not. In practice, to speed up the mke2fs and local e2fsck, the ldiskfs supports "uninit_bg" feature that allows to create the backend-filesystem without initializing all of the block groups. This dramatically reduces e2fsck time.

So for iteration, LFSCK (including backend OI scrub) should also make use of such feature to skip uninitialised block groups to optimise the scanning.

2. An administrator wants to avoid unnecessary scanning.

Generally, scanning the whole device for OI scrub routine check may take a long time. If the whole system only contains a few bad OI mappings, then it is not prudent to trigger OI scrub automatically with full speed when bad OI mapping is auto-detected. Instead, We should make the OI scrub to fix the found bad OI mappings only, and if more and more bad OI mappings are found that exceeds some given threshold, the OI scrub will run with full speed to scan whole device. The threshold of bad OI mappings that will trigger a complete scan can be adjusted via a proc interface.

3. An user wants to access to files during LFSCK scanning.

Currently, when LFSCK repairs an inconsistency, it needs to take related ldlm lock(s). In the first instance, this lock prevents concurrent modifications or purge client side cache. To simplify the implementation, the LFSCK just simply acquires LCK_EX mode ibits lock(s) on related objects. For example, when insert a name entry into the namespace, it will take LCK_EX ibits lock on the parent directory, then it will prevent all others to access such directory until related repairing has been done.

Generally, if there is very little inconsistency in the system, then such lock policy is satisfactory. However, if the inconsistency cases are more common then this lock policy is inefficient. We need to consider more suitable ldlm lock mechanism, like MDT PDO lock to allow more concurrent modifications under the shard directory.

4. An administrator wants LFSCK to find inconsistencies as quickly as possible.

For the task to provide performance optimization using available memory, a first step is to measure how much performance impact writing entries to `lfsck_namespace` actually has. There is no benefit to implementing this change if it is not going to improve performance. The performance impact of writing entries to `lfsck_namespace` will be evaluated by running LFSCK on a file system with some percentage of hard links (say 1%, 5%, 10%, 25%, 50%) either with the current code having `lfsck_namespace` written to a file on disk, or a hack mode where it is recorded on in memory (e.g. linked list or similar). If there is no significant difference in the performance there is no reason to implement this change. If the performance improvement is significant then an implementation that has inodes only written to `lfsck_namespace` when they are pushed form RAM will be evaluated.

5. An administrator needs to know what options are available, and what they do.

Review the whole of the LFSCK documentation in the manual to ensure it is fit for purpose.

Acceptance Criteria

All use cases will be demonstrated.