Final Report for the Performance Subproject of the Lustre File System Checker of the SFS-DEV-001 Contract

Revision History

Date	Revision	Author
03/16/16	Draft	R. Henwood

Executive Summary

This document finalizes the activities undertaken during the Lustre* File System Checker, Sub Project 3.4: Performance project within the OpenSFS Lustre Development contract SFS-DEV-001 signed July 30th 2011 and the subsequent modification signed October 10th 2012.

Notable highlights of this project include:

- Delivered LFSCK performance improvements between 40% and 195% depending on the configuration.
- Performance met or exceeded exceptions running in multiple MDT environments.
- All assets generated for OpenSFS during the project are attached to the OpenSFS wiki: http://wiki.opensfs.org/Contract_SFS-DEV-001

Statement of Work

This subproject ensures that LFSCK is ready to be used in production environments. It will characterize and optimize the performance of the features implemented in Subprojects 3.1-3.3, ensure that the performance impact of background scrubbing is sufficiently controlled, and determine whether Lustre protocol modifications (e.g. support for aggregate RPCs) are required. Administrative controls and monitoring will be finalized and documentation and procedures will be provided for system admin

The complete scope statement was agreed on 2014-10-12 and is available at:

http://wiki.opensfs.org/images/3/32/LFSCK_Performance_ScopeStatement.pdf

Summary of Solution Architecture

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.

Acceptance Criteria

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.

The complete Solution Architecture including Acceptance Criteria was agreed on 2014-12-04 and is available at:

http://wiki.opensfs.org/images/3/3c/LFSCK_Performance_SolutionArchitecture.pdf

Summary of Implementation

LFSCK 4: Performance is implemented in the following patches:

Change #	Subject
12737	LU-1452 scrub: OI scrub skips uninitialized groups
12738	LU-1453 scrub: auto trigger OI scrub more flexible
12958	LU-1453 scrub: NOT miss to auto detect inconsistent OI mapping
12766	LU-5682 lfsck: optimize ldlm lock used by LFSCK
	LU-5820 evaluation: linkEA verification history in RAM performance*
12966	LUDOC-259 lfsck: review and update LFSCK documentation
14014	LU-6177 lfsck: calculate the phase2 time correctly
14008	LU-6350 lfsck: lock object based on prediction for bad linkEA
14009	LU-6351 lfsck: check object existence before using it
13993	LU-6343 lfsck: locate object only when necessary
13948	LU-6322 lfsck: show start/complete time directly
13933	LU-6317 lfsck: NOT count the objects repeatedly
13923	LU-6316 lfsck: skip dot name entry

The complete Implementation milestone was agreed on 2014-12-23 and is available at:

http://wiki.opensfs.org/images/c/c7/LFSCK_Performance_Implementation.pdf

Summary of Demonstration

Between LFSCK 3 and LFSCK 4, performance increases of 40% and 195% were realized, depending on the configuration.

Complete results were agreed on 2015-05-18 and are available at:

http://wiki.opensfs.org/images/3/3b/LFSCK_Performance_Demonstration.pdf