Milestone Completion for the Parallel Directory Operations Subproject on the Lustre File System Checker Project of the SFS-DEV-001 contract.

Revision History

Date	Revision	Author
12/15/2011	Original	R. Henwood

Table of Contents

Introduction

The following milestone completion document applies to Subproject 1.2 – Parallel Directory Operations subproject within the OpenSFS Lustre Development contract SFS-DEV-001 signed 7/30/2011.

Subproject Description

Per the contract, Implementation milestone is described as follows: “This subproject allows multiple RPC service threads to operate on a single directory without contending on a single lock protecting the underlying directory in the ldiskfs file system. Single directory performance is one of the most critical use cases for HPC workloads as many applications create a separate output file for each task in a job, requiring hundreds of thousands of files to be created in a single directory within a short window of time. Currently, both filename lookup and file system-modifying operations such as create and unlink are protected with a single lock for the whole directory.

This subproject will implement a parallel locking mechanism for single ldiskfs directories, allowing multiple threads to do lookup, create, and unlink operations in parallel. In order to avoid performance bottlenecks for very large directories, as the directory size increases, the number of concurrent locks possible on a single directory will also increase.”

Milestone Completion Criteria

Per the contract, Implementation milestone is described as follows: “Contractor shall complete implementation and unit testing for the approved solution. Contractor shall regularly report feature development progress including progress metrics at project meetings and engineers shall share interim unit testing results as they are available. OpenSFS at its discretion may request a code review. Completion of the implementation phase shall occur when the agreed to solution has been completed up to and including unit testing and this functionality can be demonstrated on a test cluster. Code Reviews shall include:

a. Discussion led by Contractor engineer providing an overview of Lustre source code changes
b. Review of any new unit test cases that were developed to test changes

Location of Subproject Code changes

Complete code is available at:

http://review.whamcloud.com/#change,375

Commit at which code completed Milestone review by Senior and Principal Engineer at:

http://git.whamcloud.com/?p=fs%2Flustre-release.git;a=commit;h=19223651ed250966c0445c91dc91a5b9131dec35

Subproject Feature Confirmation

Multiple RPC service threads to operate on a single directory without contending on a single lock protecting the underlying directory in the ldiskfs file system

Results from code runs was presented to the community at the OpenSFS Lustre Pavilion SC11. This presentation is available from the OpenSFS site and is included in Appendix 1.

The results included in Appendix 2 provide a detailed description of the completion of unit tests and benchmarks.

Conclusion

Implementation has been completed according to the agreed criteria.

Appendix 1

SC11
Nov 15th, 2011

Parallel Directory Operations of Lustre

- Liang Zhen

  Whamcloud, Inc.
  liang@whamcloud.com

whamcloud

How Lustre protects directory on 1.8.x

- A directory is protected by a single LDLM lock
  - It works just like an expensive rw_semaphore for directory operations
  - By default we have max to 512 service threads to handle metadata requests, but some customers require more than 512 threads
  - Assume all threads are waiting on a single lock
- Using VFS interface to access backend filesystem (ldiskfs)
  - VFS APIs always take per-inode lock i_mutex to protect tree topology
  - On Lustre 1.8.* or earlier versions, directory tree topology is _not_ really protected by i_mutex because operations have already been serialized by LDLM lock

whamcloud

How Lustre protects directory on 2.x

- PDO ldlm lock
  - For example
    - create/unlink will take CW lock on directory, PW lock on name entry
  - Parallelized operations for file creation
    - Object creation on backend filesystem
    - Permission check
    - Name entry Lookup
    - OI (Object index) operations
    - Creation of OST objects
  - Performance increased
- No VFS on MDS stack
  - VFS is replaced by MDD/OSD
    - Directly access backend filesystem
  - Name entry operations are still serialized by rw_semaphore in OSD
    - Name entry insert
    - Name entry remove
    - Name entry lookup (READ)
    - They are expensive

© 2011 Whamcloud, Inc.

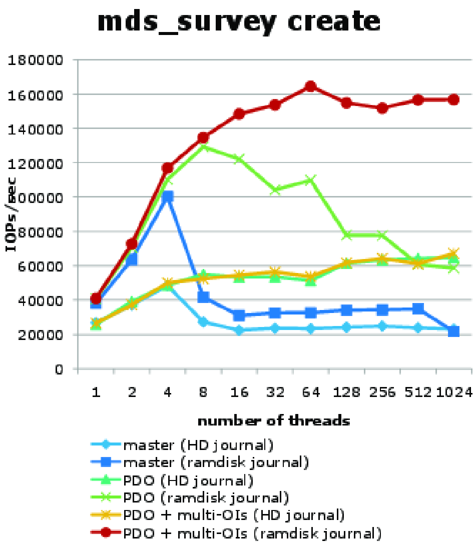
Operations on htree based directory

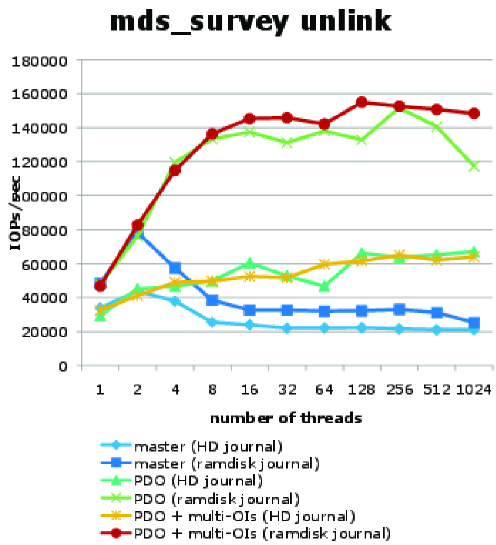
- probe htree-path
- Insert name-entry to DE-block
- Remove name-entry from DE-block
- Iterate over all DE-blocks
- Split DE-block
- Split DX-block
- Grow tree depth
  - Support N-level htree
- How to parallelize these operations?
  - No loss in performance of FFP
  - w/o rewriting htree directory of ldiskfs

Protecting htree dir by htree-lock (1/2)

- preliminary idea
  - Child-lock only protects DE-block
    - Search/insert/remove entry from DE-block
  - Tree-lock protect all other operations
    - Probe htree-path
    - split DE-block
    - split DX-block
    - grow tree depth
  - However
    - split DE-block for each ~100 creation
      - Block size is 4K, each entry has name string + extra, so bytes of each entry ~= 40byets, and each DE-block can fit in ~100 entries
    - We have hundreds or thousands service threads
      - Always some threads want to exclusively lock the tree because they need to split DE-block
    - Performance results are not cool enough

Graphs


mds_survey create
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
IOPs/sec
1 2 4 8 16 32 64 128 256 512 1024
number of threads
master (HD journal)
master (ramdisk journal)
PDO (HD journal)
PDO (ramdisk journal)
PDO + multi-OIs (HD journal)
PDO + multi-OIs (ramdisk journal)


mds_survey unlink
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
IOPs/sec
1 2 4 8 16 32 64 128 256 512 1024
number of threads
master (HD journal)
master (ramdisk journal)
PDO (HD journal)
PDO (ramdisk journal)
PDO + multi-OIs (HD journal)
PDO + multi-OIs (ramdisk journal)

Appendix 2

Technical White Paper
November 2011

Lustre Parallel Directory Operations

By Liang Zhen, Bryon Neitzel

Concurrent operations within a single ext4 file system directory are slow due to serialization caused by coarse-grained locking. Whamcloud has achieved significant performance improvements by modifying the ext4 locking granularity to allow parallel operations within a single directory.

is modified.

- Reduce thread context switch (sleep/wakeup) delays caused by contention on the single lock.

PDO Design

The Lustre service `ldiskfs` uses a hashed tree (HTree) indexing method to organize and locate directory entries. Each directory is protected by a single mutex lock. Although this single-lock protection strategy is simple to understand and implement, it creates a performance bottleneck because directory operations must obtain and hold the lock for the duration of the operation.

PDO implements a new locking mechanism that allows multiple threads to concurrently search or modify directory entries safely in the HTree. With PDO, MDS service threads can process, in parallel, multiple `create`, `lookup`, and `unlink` requests in the shared directory. Users will see performance improvement for these commonly performed operations.

Note that some applications may not see the benefit of PDO if the files being accessed are striped across many OSTs. In this case, the overhead on shared file operations with widely striped files will mask the gain of the parallelized operations.

PDO is associated with the `ldiskfs` component of the Lustre file system, which is responsible for storing data to disk and is part of the application stack that a user assumes is completely reliable. It is important that the performance gain for multi-threaded operations not come at the cost of degrading the performance of single-thread operations.

The new PDO-related code will be freely licensed under the GNU GPL. To be of value to the community, it is essential that it be easy to maintain.

PDO Requirements

The requirements for PDO are described below, which will provide a number of benefits to Lustre users with applications that perform highly concurrent metadata operations.

No performance loss for small directory operations

Implementing HTree directories with parallel directory operations provides optimal performance for large directories. However, within a directory, the minimum unit of parallelism is a single directory block (on the order of 50-100 files, depending on filename length). With PDO, performance will not scale for modifications within a single directory block, but PDO must not degrade performance for small directory operations.

No performance regressions

To be useful in practice, any new locking mechanism should maintain or reduce resource consumption compared to the previous mechanism. PDO performance for single application threads must be similar to single threaded performance without PDO.

Easy to maintain

The existing HTree implementation is well tested and in common usage. To maintain this state, the PDO implementation must minimize in-line changes and maximize ease of maintenance. Thus, PDO must not significantly restructure the current `ldiskfs` HTree implementation.

Graceful fallback when calling `ldiskfs` from the Linux virtual file system (VFS)

If `ldiskfs` is called directly from VFS rather than from Lustre, `htree-lock` will be set to `NULL` and `ldiskfs` will assume the directory is well protected by the mutex mechanism in VFS. This behavior makes the PDO version of `ldiskfs` gracefully degrade to single directory operations when accessed via the VFS interface (such as when `ldiskfs` is used to mount the MDT locally).

N-level HTree

Very large directories can contain many millions of files. Currently, the `ldiskfs` HTree structure has only two levels and can accommodate at most about 15 million files. To enable PDO changes to the HTree, the `ldiskfs` implementation will be enhanced to support an N-level HTree and larger directories. This feature is not in the requirements of PDO, but has been included in scope as it is judged by OpenSFS and Whamcloud to be a worthwhile addition to PDO work.

Big buffer LRU

The *least recently used* (LRU) buffer is a per-CPU cache used in Linux for fast searching of buffers. The default LRU size is 8. This default value is too small for Lustre to support an N-level HTree for very large directories. The purging of active buffers of this size would significantly degrade performance due to the slow and expensive buffer searching path that would need to be traversed. To avoid this scenario, an additional patch to configure the LRU buffer size with a default value of 16 will be provided.

Performance

The PDO capability has been extensively tested, including unit testing on a single-machine and large cluster performance testing. Figure 1 shows the results for open/create operations in a single directory on modest hardware. This test was run using the `mds_survey` tool, which simulates metadata traffic at the metadata target (MDT) layer of the MDS software stack. These results are useful for comparing the performance of Lustre prior to implementing the PDO feature to Lustre with the PDO feature.The test equipment for this setup included:

- Kernel: 2.6.18 rhel5, Lustre 2.1+
- CPU: I7 processor, 24G memory
- HDD: WD1002FAEX 1TB 7200 RPM 64MB cache
- SSD: Crucial RealSSD C300 Series 128GB

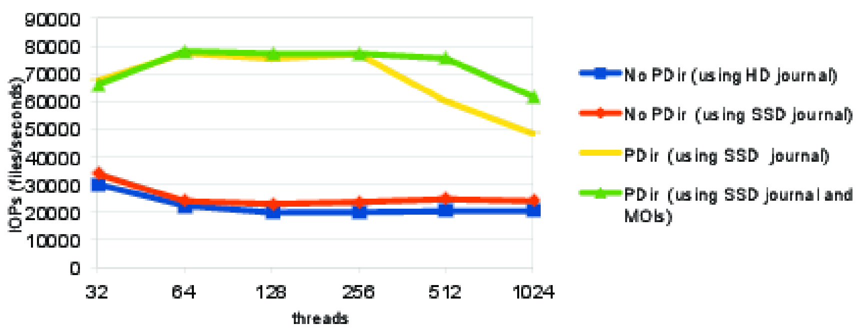


Figure 1. PDO performance test results for open/create operations

Conclusion

Whamcloud is continuing to make investments in improving Lustre performance in various areas in the code base. PDO is an excellent example of these improvements. This feature is targeted for the Lustre 2.2 release which will be available in the first half of 2012.

Whamcloud, Inc., 696 San Ramon Valley Blvd., Suite 261, Danville, CA 94526
Phone +1 925-272-9557 Email: info@whamcloud.com