Lustre Feature Test Plan for

Data on MDT

Revision: v1
Date: 11/20/2017

Table of Contents

Revision History

The following is a chronological history of changes made to this document.

Revision	Date	Reason for change	Author
v.1	11/20/2017	Initial Version	Saurabh Tandan

Introduction

Lustre File System performance is currently optimized for large files. This results in additional RPC round-trips to the OSTs, which hurt small file performance. The Data on MDT (DOM) project aims to correct this deficiency by allowing the data for small files to be placed on the MDT so that these additional RPCs can be eliminated and performance correspondingly improved.

Users or system administrators will set a layout policy that locates files to be stored entirely on an MDT. Files that grow beyond this size will be migrated onto OSTs.

This work is tracked with Lustre JIRA LU-3285 .

Use Case Scenario

The following use cases are how we envision users using Data on MDT and/or necessary configurations that should be tested.

1. New files are created with an explicit layout to store the data on the MDT.
2. New files are created with an implicit layout to store the data on the MDT inherited from the default layout stored on the parent directory.
3. A small file is stored without the overhead of creating an OST object, and without OST RPCs.
4. A small file is retrieved without the overhead of accessing an OST object, and without OST RPCs.
5. A client accesses a small file and has the file attributes, lock, and data returned with a single RPC.
6. An administrator sets a global maximum size limit for small files stored on the MDT(s). Files larger than this value do not store their data on an MDT.

Feature Overview

Installation

Data on MDT is already a feature targeted in Community 2.11 and will be installed by default; no special requirement for installation.

Configuration

DoM is configured like any other composite file:

```
lfs setstripe -E 1M -L mdt dom_file
```

The new option here is `'-L'` which stands for 'layout'. The lfs command above creates a composite file with the first component on a MDT with size 1M. Other than `'mdt'`, `'-L'` accepts `'raid0'` and `'released'` options.
The user can decide the max size limit for file/directory for it to be DOM simply by setting size of the first entry via `-E` option. Note, this doesn't mean that all files below 1M will be DOM files;

this is set for a particular file or directory.. Each file that have layout on MDT has it's own setting and it's sub-directories can inherit that layout.
The DoM size limit can be configured using the parameter `'dom_stripesize'` of the Layered Object Device (LOD) and can be set with following command:

```
lctl set_param lod.*.dom_stripesize=<limit>
```

Default value is 1M (megabytes), maximum value is 1G (gigabytes). Note, that setting this parameter to 0 can be used as 'disabler' for DoM on particular server.

The '`-L mdt'` option can also be used in '`lfs find`' and '`lfs getstripe`' to find/check files with mdt component. These features are tested in sanity.sh test_270a, test_270e respectively.

Example:

```
lfs find -L mdt –type f dom_file

lfs getstripe -L mdt dom_file
```

Tests

Testing of the Data on MDT feature is made up of four main categories; functional, performance, recovery, and interoperability. Functional testing: does the feature behave as expected under normal and error conditions. Performance testing: an additional script "dom_performance.sh" was added to compare the performance for normal files and dom files. Recovery: does this feature behave normally under failure conditions. Interoperability testing: is new Lustre with this feature compatible with older Lustre version without this feature.
There are two test scripts added as part of this feature to the test environment: sanity-dom.sh and dom-performance.sh.

All the Lustre test suites will be run and should pass with default Data on MDT file layout on the mount point.

Functional Testing

Functional tests are expected to run on an automated virtual environment.

sanity-dom.sh

Test ID	Test Description	Comment
test_1()	• Write a file on one mount • Truncate on the other • Write again	
test_2()	• Write with a seek • Then append • And finally, read from a single mountpoint	

test_3()	Truncate over DoM size on different nodes • Write on one node to the DoM stripe and then truncate to over DoM size • Read on the second node inside DoM stripe to take a lock data from the first client • Now do local truncate over DoM size and check size is correct	
test_fsx()	Replica of sanityn test_16 but with DoM layout • Runs fsx test from two clients • Does read/write/truncate in different combinations from both clients	
test_sanity()	Run some of the sanity.sh subtests with Data-on-MDT List of sanity subtests run under this subtest: 36 39 40 41 42 43 46 56r 101e 119a 131 150 155a 155b 155c 155d 207 241 251	
test_sanityn()	Run some of the sanityn.sh subtests with Data-on-MDT List of sanityn.sh subtests run under this subtest: 1 2 4 5 6 7 8 9 10 11 12 14 17 19 20 23 27 39 51a 51c 51d	

sanity.sh

Test ID	**Test Description**	**Comments**
test_270a()	DoM: Basic functionality tests • create DoM file • Skip free space checks with ZFS • Write DoM file • Also check direct IO along write • Truncate DoM file • Append to DoM file • Delete DoM file • Combined striping	Use case (1)

	• Also check direct IO along write	
test_270b()	DoM: Maximum size overflow checks for DoM-only file • Truncate over the limit • Write over the limit • Append over the limit	Use case (7)
test_270c()	DoM: DoM EA inheritance tests • Check files inherit DoM EA • Check directory inherits DoM EA and use it as default	Use case(2)
test_270d()	DoM: Change striping from DoM to RAID0 • Inherit default DoM striping • Change default directory striping	Use case(2)
test_270e()	DoM: Testing "lfs find" with DoM files test • Find DoM files by layout • There should be 1 dir with default DOM striping • Find DoM files by stripe size • Find files by stripe offset except DoM files	
test_270f()	DoM: maximum DoM stripe size checks • Exceed maximum stripe size • Too low values to be aligned with smallest stripe size 64K	Use case (6)
test_271a()	DoM: Cache data for read after write	
test_271b()	DoM: no glimpse RPC for stat - only for DoM files	Use case(3), Use case(4)
test_271ba()	DoM: no glimpse RPC for stat – combined files	Use case(3), Use case(4)
test_271c()	DoM: IO lock at open saves enqueue RPCs	Use case(3), Use case(4)

sanityn.sh

Test ID	Test Description	Comments
test_100a()	DoM: Glimpse RPCs for stat without IO lock (DoM only file) • First stat from server should return size data and save glimpse • Second stat to check size is NOT cached on client without IO lock	Use case(3), Use case(4)
test_100b()	DoM: No glimpse RPC for stat with IO lock (DoM only file) • First stat? • Second stat to check size is cached on client	Use case(3), Use case(4)
test_100c()	DoM: Write vs Stat without IO lock (combined file) • Check's that size is merged from MDT and OST correctly	
test_100d()	DoM: Write+Truncate vs Stat without IO lock (combined file) • Check's that reported size is valid after file grows to OST and is truncated back to MDT stripe size	
test_101a()	Discard DoM data on unlink	
test_101b()	Discard DoM data on rename	
test_101c()	Discard DoM data on close-unlink	

Inter-operation

Lustre clients from the latest 2.10 and earlier will not be able to read small files on MDTs created by Lustre 2.11 clients. The older clients will return an '–EINVAL' i.e. number '-22' error message due to not understanding the new DOM file layout. There is also no way for these clients to access the file data on the MDT, so no compatibility mode is possible.

Accessing Lustre files systems from clients of different versions is a supported configuration. A large site may have a file system shared between different systems where the clients are upgraded independently. Since files are only created with DOM by request, it is possible for DOM-capable clients connected to a DOM-capable server to create DOM files while non-DOM-capable clients are still accessing the filesystem. To avoid an old client seeing errors when trying to access files created with DOM by a new client, either all of the clients should be upgraded before the servers, or an administrator may disable DOM explicitly on the MDT(s) until all clients are upgraded. The mechanism for how to disable Data on MDT is discussed in the high level design.

Failure and recovery Testing

All existing failure and recovery tests will be run - https://wiki.hpdd.intel.com/display/ENG/Regression+Test+Suites+and+Failover+Test+Suites

Performance Testing

Additional script "dom-performance.sh" has been added to the test suite to compare the performance benefits between the normal file and DoM file. We expect better stat, read, append and partially write performance for DoM files. If the same file is stored on MDT and OST, the file stored on MDT will be retrieved faster as there will be less RPCs, given that the both servers are identical. But at the same time it is hard to say exactly about numbers. It depends on server hardware and cluster configuration, e.g. normal file creation/write is balanced among all OSTs, while DOM goes to the singe MDT in most cases, so benefits of DOM may be hidden by this. Also it depends on cluster itself, since DoM saves RPCs. Hence, better effect will be seen on clusters where network is a bottleneck, not disk, otherwise, with lightning-speed network results will depends only on server hardware mostly. If the file grows beyond the DoM size it grows right to the OST objects hence the performance for it is same as for the normal files in that case.

The following tests will be run for both normal file and DoM file and later compared for performance between them:

- Mdtest
- IOR
- Dbench
- Smallfile
- Smalliomany

Benchmarks IOR, MDtest and FIO were run with configuration of 10 clients, 2 MDTs on 2 MDS each and 4 OSTs on 2 OSSs each which confirms the above mentioned improvements.

Documentation

For any more information please refer to LUDOC-385 .