Test Plan
MDT reply reconstruction improvement

version: 2
date: 2015/07/17
author: Grégoire Pichon - gregoire.pichon@bull.net

Introduction

Currently, the MDT cannot handle more than one single filesystem-modifying RPC at a time, because there is only one slot per client in the MDT last_rcvd file. This slot is used to save the state of the last transaction (transaction number, xid, RPC result, operation data) so the reply can be reconstructed in case of RPC resend if the reply was lost. As a consequence, the filesystem-modifying MDC requests are serialized, leading to poor metadata performance from a single Lustre client.

The goal of this project is to support multiple slots per client for reply reconstruction of filesystem-modifying MDT requests, and to improve metadata operations performance of a single client.

This work is tracked with Lustre JIRA LU-5319.

Tests

Single client metadata performance

The single client metadata performance improvement will be verified with the *mdtest* benchmark. The benchmark will perform creation and unlink operations of a fixed total number of files, using a "directory per process" model.

Auster test suite

The *acceptance-small* test suite will be run to verify no functional regression.
The *Auster* test suite will be enhanced with several tests cases.

sanity

The import structure of a MDC indicates support of multiple filesystem-modifying RPCs in parallel within `connect_flags` field, and `max_mod_rpcs` connect data.

test 245 check mdc connection flag/data: multiple modify RPCs

A MDC is able to send multiple modify RPCs in parallel

test 182 Test parallel modify metadata operations from mdc

The lr_reader tool displays per-client area of LAST_RCVD file and the reply data of the REPLY_DATA file.

test 252 Check lr_reader tool

conf-sanity

The maximum number of modify RPCs in flight can be dynamically updated through the client device procfs file `max_mod_rpcs_in_flight`, but it cannot exceed the maximum number of modify RPCs per client returned by the server during connection establishment. Additionally, it cannot exceed or equal the maximum number of RPCs in flight.
The `max_mod_rpcs_in_flight` value controls the maximum number of modify RPCs in flight of a client device connected to a MDT.
One additional close request can be allowed above the maximum number of modify RPCs in flight, if no other close request is in flight.
The mdt kernel module has a parameter `max_mod_rpcs_per_client` that specifies the maximum number of RPCs in flight allowed per client.

test 86a check max_mod_rpcs_in_flight is enforced
test 86b check max_mod_rpcs_in_flight is enforced after update
test 86c check max_mod_rpcs_in_flight update limits
test 86d check one close RPC is allowed above max_mod_rpcs_in_flight

replay-single

When modify metadata RPCs are sent by a client, the MDT handles the requests, saves the reply data in-memory, and saves the reply data on-disk in the same transaction than the corresponding MDT request disk modification.

If the requests of modify metadata RPCs are lost, the client resends the RPC requests and the MDT handles the requests normally.
test 102a check resend (request lost) with multiple modify RPCs in flight

If the replies of modify metadata RPCs are lost, the client resends the RPC requests and the replies are reconstructed from the in-memory reply data.
test 102b check resend (reply lost) with multiple modify RPCs in flight

If the MDT crashes before the disk transactions have been committed, the MDT handles normally the RPC requests replayed by the client.
test 102c check replay w/o reconstruction with multiple mod RPCs in flight

If the MDT crashes after the disk transactions have been committed, the MDT restores the in-memory reply data from the disk when it recovers, and is able to reconstruct the replies when the client RPC requests are replayed.
test 102d check replay & reconstruction with multiple mod RPCs in flight

Upgrade, downgrade and inter-operability

Upgrade and downgrade of the Lustre client will be verified.
Upgrade and downgrade of the Lustre server will be verified.
Interoperability between a client that supports or does not support and a server that supports or does not support will be verified.

Results

Test Bed

Hardware

- Lustre server node
  8 CPUs, 2 sockets Intel Xeon X5560 @ 2.80GHz, 36GiB memory, 1 InfiniBand FDR adapter
- Lustre client node
  16 CPUs, 2 sockets Intel Xeon E5-2690 @ 2.90GHz, 64GiB memory, 1 InfiniBand FDR adapter

Software

- Kernel 2.6.32-431.29.2
- OFED 3.12
- Lustre 2.7.56
  + patch #14861 “tests: testcases for multiple modify RPCs feature”
  + patch #14862 “utils: update lr_reader to display additional data”
- Lustre 2.7.0 for interoperability and upgrade/downgrade tests

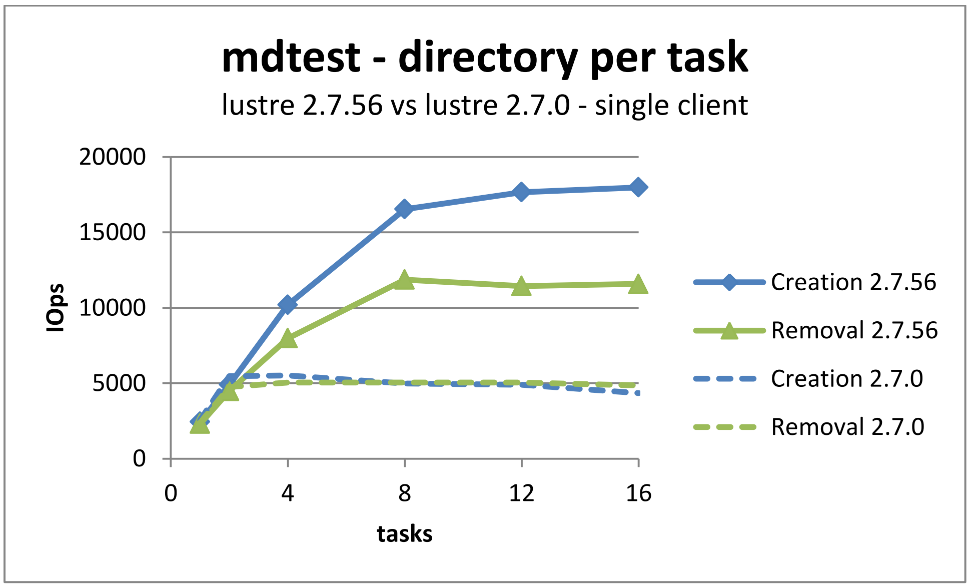
File System

- MGT /var/loop/mgt ldiskfs 200000 Bytes
- MDT0 /dev/ram0 ldiskfs 4GiB
- MDT1 /dev/ram1 ldiskfs 4GiB
- OST0 /dev/ram2 ldiskfs 4GiB
- OST1 /dev/ram3 ldiskfs 4GiB

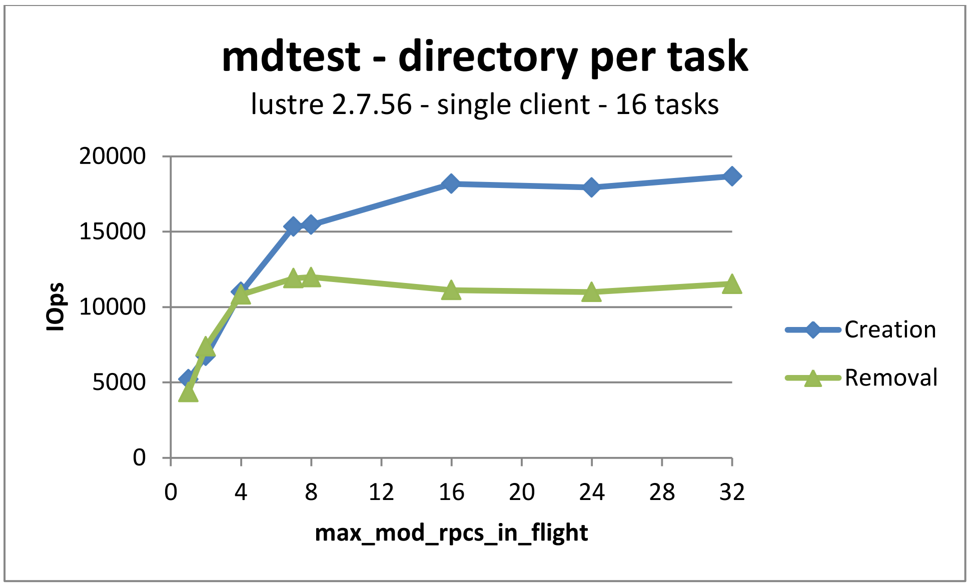
Test suite	Test case	Status
sanity	test 182: Test parallel modify metadata operations	PASS
	test 245: check mdc connection flag/data: multiple modify RPCs	PASS
	test 252: check lr_reader tool	PASS
replay-single	test 53a: |X| close request while two MDC requests in flight	PASS
	test 53b: |X| open request while two MDC requests in flight	PASS
	test 53c: |X| open request and close request while two MDC requests in flight	PASS
	test 53d: close reply while two MDC requests in flight	PASS
	test 53e: |X| open reply while two MDC requests in flight	PASS
	test 53f: |X| open reply and close reply while two MDC requests in flight	PASS
	test 53g: |X| drop open reply and close request while close and open are both in flight	PASS
	test 53h: open request and close reply while two MDC requests in flight	PASS
	test 102a: check resend (request lost) with multiple modify RPCs in flight	PASS
	test 102b: check resend (reply lost) with multiple modify RPCs in flight	PASS
	test 102c: check replay w/o reconstruction with multiple mod RPCs in flight	PASS
	test 102d: check replay & reconstruction with multiple mod RPCs in flight	PASS
conf-sanity	test 86a: check max_mod_rpcs_in_flight is enforced	PASS
	test 86b: check max_mod_rpcs_in_flight is enforced after update	PASS
	test 86c: check max_mod_rpcs_in_flight update limits	PASS
	test 86d: check one close RPC is allowed above max_mod_rpcs_in_flight	PASS

Single Client Metadata Performance

The charts present the results of the mdtest benchmark performing creation and unlink operations of one million files using a “directory per process” model.
Note: the following format parameter has been added for OSTs to ensure enough objects can be created: OST_FS_MKFS_OPTS="-N 4000000".



The Lustre 2.7.56 measurement has been performed with max_mod_rpcs_in_flight=32.
We observe a performance improvement between Lustre 2.7.56 and Lustre 2.7.0 for the single client metadata modify operations. The creation rate increased by more than 200% and the removal rate increased by more than 100%.
The removal rate increase is lower than expected. This needs to be analyzed to understand if this comes from the feature itself, or from another feature landed in the meantime.



The single client performance of file creations and removals increases with the number of modify RPCs allowed in parallel (max_mod_rpcs_in_flight mdc parameter).

Interoperability

The following interoperability configurations have been tested:

- Client node with Lustre 2.7.0 and server node with Lustre 2.7.56
- Client node with Lustre 2.7.56 and server node with Lustre 2.7.0

In each interoperability configurations, several filesystem operations were performed, including metadata operations (creation, stat, unlink).

Additionally, the Maloo tests succeeded while the client side patch #14374 and the server side patch #14860 was independently submitted into Gerrit.

Upgrade / Downgrade

The upgrade and downgrade of Lustre version has been tested with the following sequence.

Client and server nodes are installed with Lustre 2.7.0

1. Format the filesystem
2. Mount the targets and the client
3. Perform filesystem operations (sanity test 182, IO from/to files, lfs df, …)
4. Unmount the targets
5. **Upgrade the server node to Lustre 2.7.56**
6. Mount the targets
7. Perform filesystem operations
8. Unmount the client
9. **Upgrade the client node to Lustre 2.7.56**
10. Mount the client
11. Perform filesystem operations
12. Unmount the client
13. **Downgrade the client node to Lustre 2.7.0**
14. Mount the client
15. Perform filesystem operations
16. Unmount the targets
    a. for each MDT target sequentially: mount with abort_recovery option and unmount this clears the OBD_INCOMPAT_MULTI_RPCS incompatibility flag
17. **Downgrade the server node to Lustre 2.7.0**
18. Mount the targets
19. Perform filesystem operations

20. Unmount and cleanup the filesystem

A similar sequence has been tested with first upgrade of the client node, then upgrade of the server node, then downgrade of the server node and finally downgrade of the client node.

Downgrading the server node without doing step 16.a has been tested. In that case, an expected error occurred while mounting an MDT target (mount.lustre: mount /dev/ram0 at /mds1 failed: Invalid argument).