Test Plan
For the
Imperative Recovery Project
On the
ORNL Scalable File System Development Contract

Revision History

Date	Revision	Author
07/19/2011	Original	J. Xiong
08/16/2011	Rev A	J. Xiong I. Colle
08/31/2011	Rev B	J. Xiong I. Colle

Table of Contents

Imperative Recovery Test Plan

whamcloud

I. Introduction

The following test plan applies to the Imperative Recovery (IR) project within the ORNL Scalable File System Development contract signed 11/23/2010 and modified on 03/18/2011. ORNL is experiencing exceedingly long service outages during server failover and recovery. Currently this process takes unacceptably long because it depends on timeouts scaled to accommodate congested service latency.

Large-scale lustre implementations have historically experienced problems recovering in a timely manner after a server failure. This is due to the way that clients detect the server failure and how the servers perform their recovery.

From the client side, the only way it can detect the failure of a server is by the timeout of Remote Procedure Calls (RPCs). If a server has crashed, the RPCs to the targets associated with that server will time out and the client will attempt to reconnect to the failed server, or to a failover server if configured. After reconnection, the client will participate in recovery and then continue in normal service.

A restarting target will try to recover its state by replaying RPCs executed but not committed by its previous incarnation. These RPCs must be executed in the original order to ensure the target reconstructs its state consistently. The target must therefore wait for all clients to reconnect before recovery completes and new requests can be serviced because a missed replay transactions could result in recovery failure and a late replay transaction could be invalidated by new requests. This wait, called the “recovery window” is bounded to ensure a failing client cannot delay resumption of service indefinitely.

The above processes are time consuming, as most are driven by the RPC timeout which must be scaled with system size to prevent false diagnosis of server death. This is especially difficult in an environment of ORNL’s scale.

The purpose of imperative recovery is to reduce the recovery window by actively informing clients of server failure. The resulting reduction in the recovery window will minimize target downtime and therefore increase overall system availability.

II. Test cases

This development provides a new recovery type to the system: imperative recovery (IR). The purpose of the following tests is to compare the performance between the previously existing standard recovery (SR) and IR under different situations.

The test cases are divided into three components:
1 – Regression Testing
2 – Unit Testing

3 – Scalability Testing

Regression Testing

Regression testing will be accomplished by running acc-sm tests.

Unit Testing

Unit Testing consists of a code coverage test and feature coverage tests as further specified below:

- test_100 – Verify SR works w/o IR
- test_101 – Verify IR works w/o SR
- test_102 – Verify new client receives updated nidtbl version after MGS restart
- test_103 – Verify MDS can start w/o MGS and receive updated nidtbl version after MGS startup
- test_104 – Verify NOIR option works
- test_105 – Non-IR client suppor

System Testing

System testing consists of failover testing and scalability testing.

Failover Testing:

- Set up active/passive failover pairs for OSTs, turn off master server and start slave server, ensure slave server's NID is transferred and clients reconnect to it
- Repeat above test with MDTs
- Perform same testing with workload on targets and verify applications do not see errors

Scalability Testing:

- Verify IR is able to support a minimum of 75k clients using Hyperion
- Verify IR continues to perform when heavy workload on OSTs using Whamcloud Toro cluster
- Verify IR continues to perform when there is a heavy workload on the MGS using the Whamcloud Toro cluster

For the scalability portion of the testing, there are several control entities under procfs about imperative recovery. This is a list of proc entries we will use during the tests.

Procfs entries to be used

NODE	PATH	SETTINGS
MGS	mgs.MGS.live.lustre	Read: print out stats and state of IR [root@wolf6 tests]# lctl get_param mgs.MGS.live.lustre … Imperative Recovery Status: state: Full, nonir clients: 0 nidtbl version: 5 notify total/max/count: 0/0/3 Write: 0 Clear stats;

		Write: "status=Disabled" Disable IR; Write: "status=Full" Set IR to full state.
CLIENT	Osc.lustre-OST0000-osc-ffff*.pinger_recov	This can control if standard recovery will be used. If this is disabled, the corresponding OST can be recovered by only imperative recovery. Read: Write: 0 – disable pinger recover; 1 – enable pinger recover.
CLIENT	Osc.lustre-OST0000-osc-ffff*.import	Read: target: lustre-OST0000_UUID state: FULL instance: 1859057999 Check state: FULL to make sure the osc has been reconnected to the ost successfully.

Scalability Test cases

Test Case Name	**ir.scalability.ost**
Purpose	To measure and compare how quick the IR can recover a failing OST
Actors	OST, MGS and client
Description	Shutdown and restart an OST in a running cluster, investigate the time for the OST to get recovered. This test case will run with the cases of IR and SR specifically
Environment Settings	1. Must have workload on the failing OST. I recommend running IOR on each client node to write objects on this OST; 2. When the system is up, make sure the MGS is on Full state by checking: at the MGS: grep state: /proc/fs/lustre/mgs/MGS/live/{fsname}
Trigger	OST shutdown and restarted
Preconditions	1. MGS is running; 2. all clients have connected;

	3. IOR are running on clients
Postconditions	
Special Requirements	Disable SR by: On all clients: lctl set_param osc.{ost import name}.pinger_recov=0 Disable IR by: On the MGS: lctl set_param mgs.MGS.live.{fsname}=”status=Disabled”
Assumptions	
Expected Results	Clients should be able to reconnect to the failing target; Investigate the time for all clients to get reconnected to the failing target. We shall calculate the maximum and average recovery time from each client
Notes and Issues	Compare maximum and average recovery time under SR and IR case

Test Case Name	**ir.scalability.mdt**
Purpose	To measure how quickly the IR can recover a failing MDT
Actors	MDT, MGS and client
Description	Shutdown and restart an MDT in a running cluster, investigate the time for the MDT to get recovered. This test case will run with the cases of IR and SR specifically
Environment Settings	1. Configure the MGS and MDT separately; 2. There must be workload on the failing MDT. I recommend running mdsrate on each client node to create objects on this MDT; 3. When the system is up, make sure the MGS is on Full state by checking: at the MGS: grep state: /proc/fs/lustre/mgs/MGS/live/{fsname}
Trigger	MDT shutdown and restarted
Preconditions	1. MGS is running; 2. all clients have connected; 3. mdsrate is running for a while to make sure there is enough dirty data unflushed on the MDT
Postconditions	
Special Requirements	Disable SR by: On all clients: lctl set_param osc.{ost import name}.pinger_recov=0 Disable IR by: On the MGS: lctl set_param mgs.MGS.live.{fsname}=”status=Disabled”

Assumptions	
Expected Results	Clients should be able to reconnect to the failing target; Investigate the time for all clients to get reconnected to the failing target. We will calculate the maximum and average recovery time from each client.
Notes and Issues	Compare maximum and average recovery time under SR and IR case

Test Case Name	**ir.scalability.multiple_targets**
Purpose	To make sure imperative recovery works well if multiple targets fail meanwhile
Actors	OST, MGS and client
Description	Suppose there are tens of OSTs on each OSS, and restart as many OSSes as possible on the same time, to investigate if IR can handle this case smoothly. This test case will run with the cases of IR and SR specifically
Environment Settings	N/A
Trigger	OSSes shutdown and restarted
Preconditions	MGS is running and all clients have connected
Postconditions	
Special Requirements	Disable SR by: On all clients: lctl set_param osc.{ost import name}.pinger_recov=0 Disable IR by: On the MGS: lctl set_param mgs.MGS.live.{fsname}="status=Disabled"
Assumptions	
Expected Results	Clients should be able to reconnect to all restarting OSTs in a short time
Notes and Issues	Compare maximum and average recovery time under SR and IR case

Test Case Name	**ir.scalability.scalability**
Purpose	To verify that imperative recovery can notify many clients in a reasonable time
Actors	OST, MGS and client
Description	Mount thousands of mount points on each client node, and then restart an OST;

	This test case will run with the cases of IR and SR specifically
Environment Settings	N/A
Trigger	OSSes shutdown and restarted
Preconditions	1. MGS is running and all clients have connected 2. Make sure you have this line in your modprobe.conf file: options obdclass lu_cache_percent=1, otherwise, you can't mount over 200 mountpoints on one node 3. Clear the IR stats by: at the MGS node: lctl set_param mgs.MGS.live.{fsname}=0
Postconditions	
Special Requirements	Disable SR by: On all clients: lctl set_param osc.{ost import name}.pinger_recov=0 Disable IR by: On the MGS: lctl set_param mgs.MGS.live.{fsname}="status=Disabled"
Assumptions	
Expected Results	Clients should be able to reconnect to the restarting OST in a short time. Collect the output of lctl get_param mgs.MGS.live.{fsname}
Notes and Issues	One of the IR's problems is that the MGS has to notify each clients for the update of target register, so it may have the scalability problem if there are two many clients in the cluster. This test is to verify IR works well with at least 100K clients. Compare maximum and average recovery time under SR and IR case Measure CPU load on MGS during IR.