Lustre Performance Regression Test Plan

Author	Date	Description of Document Change	Client Approval By	Client Approval Date
Minh Diep	12/18/08	Initial draft		
Minh Diep	1/29/09	Update after Cray's review		

I. Test Plan Overview

This test plan is a high level guide for using at-scale* cluster to test performance regression on 1.6 and 1.8 branch and enable/disable feature comparison in 1.8.

* At-scale: large scale cluster in various sites (Sun internal, Cray, LLNL...)

Executive Summary

- Lustre 1.8 branch has features that have been through the internal test cycle and have not been tested for performance regression. We can use at-scale system to regularly test each feature for performance regression and fix the issue before it go out to customers.
- Results will be reviewed by developers.
- At-scale cluster will be used for this test plan
- The feature must pass internal testing prior to becoming a candidate for regression testing on the at-scale Cluster.

Problem Statement

Lustre 1.8 features have not been tested for performance on regular base. It's impossible to keep monitor and improve the performance if we don't do regular performance regression testing from build to build. It's also difficult to understand the performance impact from each feature after enabling it.

Goal

The goal is to compare performance between 1.6 and 1.8; and run existing features to find out if there are any performance impact and regression from enabling it.

Success Factors

The success factor will be that there are no performance regression and we do performance testing on regular base and report performance issue back to the development team.

Benchmarking

Following is the list of benchmarks to measure performance in this plan

- Benchmarking individual disks with dd (lustre independent)
- Benchmarking raw RAID6 performance with Sgpdd-survey
- Benchmarking RAID6 performance with OBDFilter-survey
- Benchmarking network bandwidth with LNET Selftest
- Benchmarking metadata operations with Metabench on multiple clients
- Benchmarking Lustre clients with PIOS on single client, IOR on multiple clients

Testing Plan

- ### Performance comparison between 1.6 and 1.8

  Testing will be to compare performance between 1.6 and 1.8 and all default features. This is to make sure by default there is no performance impact for new releases.

  - Quota: performance comparison between enable and disable.
    Set quota to user with quota limit to about 110% the amount of data written.

- ### Performance comparison on each feature for 1.8

  Testing will be to compare performance in a feature when it's enable and disable. This is to make sure that each feature does not impact performance when being enable.

  - Quota: performance comparison between enable and disable.
    Set quota to user with quota limit to about 110% the amount of data written.
  - OST Pool: performance comparison between pool and non-pool filesystem.
    Create a single pool with all OSTs.
  - Adaptive Timeout: performance comparison between enable and disable.
  - VBR: This feature can not be disable.

Test Cases

Test Case Description:

1. **Benchmarking individual disks with dd**

   Individual disks need to be measured in order to remove low performing disks from Lustre configuration. We suggest running following command in 3 iterations to verify the individual disk performance on each OSS.
   # dd if=/dev/zero of=<disk name> bs=1M count=16384 oflag=direct

   An average of the 3 iterations should be considered to remove/replace lowest performing disks. A variation of +/- 3% in the disk performance is acceptable.

   Note: This measurement only needs to run once and it's Lustre independent.

2. **Benchmarking raw RAID6 performance with Sgpdd-survey on Single OSS**

   Sgpdd-survey is a wrapper script available in Lustre IOKit. This script exercises disk performance using sgp_dd tool available in SG3 utilities (http://sg.torque.net/sg/p/sg3_utils-1.27.tgz).

   Disks Tuning:

```
# Set max_sectors_kb to 4096 to increase I/O size to disks
for i in /sys/block/sd*/queue/max_sectors_kb; do
        echo 4096 > $i;
done
# Change cache setting to “Write Through”
for i in /sys/block/sd*/device/scsi_disk*/cache_type; do
        echo 'write through' > $i;
done
# Disable NCQ/TCQ since it re-orders writes
for i in /sys/block/sd*/device/queue_depth; do
        echo 1 > $i;
cat $i;
done
```

   **** Below is suggestion for creating Raid6 on Sun Hardware ****
   Create Raid6 (Hardware dependent)

   ### RAID6 Configuration with udev on Thumper/Thor:

```
yes | mdadm -C /dev/md0 --auto=yes -c 256 -l 6 -n 6 -x 1 /dev/dsk/c{0,1,2,3,4,5}d1
/dev/dsk/c0d8
yes | mdadm -C /dev/md1 --auto=yes -c 256 -l 6 -n 6 /dev/dsk/c{0,1,2,3,4,5}d2
yes | mdadm -C /dev/md2 --auto=yes -c 256 -l 6 -n 6 /dev/dsk/c{0,1,2,3,4,5}d3
yes | mdadm -C /dev/md3 --auto=yes -c 256 -l 6 -n 6 /dev/dsk/c{0,1,2,3,4,5}d4
yes | mdadm -C /dev/md4 --auto=yes -c 256 -l 6 -n 6 /dev/dsk/c{0,1,2,3,4,5}d5
```

```
yes | mdadm -C /dev/md5 --auto=yes -c 256 -l 6 -n 6 /dev/dsk/c{0,1,2,3,4,5}d6
yes | mdadm -C /dev/md6 --auto=yes -c 256 -l 6 -n 6 /dev/dsk/c{0,1,2,3,4,5}d7

for i in 0 1 2 3 4 5 6; do
        echo 16384 > /sys/block/md$i/md/stripe_cache_size;
        blockdev --setra 8192 /dev/md$i;
done
```

Refer to
https://cepedia.sfbay.sun.com/index.php?title=TokyoTech_Thumper_and_Lustre or
https://bugzilla.lustre.org/show_bug.cgi?id=17462

for udev configuration with Sun Thumper (X4500) and Thor (X4540) servers. Above configuration creates 7 arrays of RAID6 (4+2) with disks on different controllers and leaves 6 disks to be used for external journal and as spare.

Sgpdd-survey Tuning:

```
# Create raw devices out of RAID6 arrays
for i in 0 1 2 3 4 5 6; do
        raw /dev/raw/ra1$i /dev/md$i;
done
raw -qa
```

Use the Sgpdd-survey script shown in Bug 17218 (https://bugzilla.lustre.org/attachment.cgi?id=20068) which has few improvements over sgpdd-survey script from Lustre IOKit. Improvements include, use of block device layer to support directIO instead of sg device layer, 1 MB blocksize, directIO.

Sgpdd-survey Invocation:

```
rawdevs="/dev/raw/raw10 /dev/raw/raw11 /dev/raw/raw12 /dev/raw/raw13
/dev/raw/raw14 /dev/raw/raw15 /dev/raw/raw16" rszlo=1024 rszhi=1024 crglo=1
crghi=16 thrlo=1 thrhi=16 size=32768 ./sgpdd-survey
```

While the command is running, verify that all writes in /proc/mdstat are zero copy writes and not copied writes. Rsz paramter of sgpdd-survey and chunksize parameter to mdadm should be chose in such a way that, all writes will be zero-copy writes. For example,

```
# grep zcopy /proc/mdstat
reads: 0 for rmw, 14420 for rcw. zcopy writes: 33554432, copied writes: 0
reads: 0 for rmw, 14678 for rcw. zcopy writes: 33554432, copied writes: 0
reads: 0 for rmw, 13979 for rcw. zcopy writes: 33554432, copied writes: 0
reads: 0 for rmw, 17646 for rcw. zcopy writes: 33554432, copied writes: 0
reads: 0 for rmw, 18130 for rcw. zcopy writes: 33554432, copied writes: 0
reads: 0 for rmw, 16084 for rcw. zcopy writes: 33554432, copied writes: 0
```

3. Benchmarking RAID6 OST performance with OBDFilter-survey

OBDFilter-survey is a tool from Lustre IOKit that exercises the OBDFilter stack of OSS. It

can be run directly on disks, OBDFilter devices and from clients. Make sure your Lustre version includes patch for obdfilter-survey mentioned in https://bugzilla.lustre.org/show_bug.cgi?id=17382

Verify that “lctl device list” command shows all the OBDFilter devices. For example,

```
# lctl dl | grep obdfilter
2 UP obdfilter lustre-OST0000 lustre-OST0000_UUID 7
3 UP obdfilter lustre-OST0001 lustre-OST0001_UUID 7
4 UP obdfilter lustre-OST0002 lustre-OST0002_UUID 7
5 UP obdfilter lustre-OST0003 lustre-OST0003_UUID 7
6 UP obdfilter lustre-OST0004 lustre-OST0004_UUID 7
7 UP obdfilter lustre-OST0005 lustre-OST0005_UUID 7
8 UP obdfilter lustre-OST0006 lustre-OST0006_UUID 7
```

For running OBDFilter-survey on OSS, invoke it as:
targets=" lustre-OST0000 lustre-OST0001 lustre-OST0002 lustre-OST0003 lustre-OST0004 lustre-OST0005 lustre-OST0006” ./obdfilter-survey

For running it from Lustre client, invoke it as:
targets="oss01:lustre-OST0000 oss01:lustre-OST0001 oss01:lustre-OST0002 oss01:lustre-OST0003 oss01:lustre-OST0004 oss01:lustre-OST0005 oss01:lustre-OST0006” ./obdfilter-survey

4. Benchmarking Lustre Network with LNET Selftest

Following three test cases must be benchmarked before running any tests on Lustre clients. LNET selftest will measure performance of the network with Lustre networking protocol assuming unlimited disk bandwidth.

- LNET selftest between 1 Lustre client and 1 Lustre OSS server
- LNET selftest between 1-8 lustre clients and 1 Lustre OSS
- LNET selftest between 20 lustre clients and 4 Lustre OSS

Use following scripts to run LNET selftest.

```
#!/bin/bash
export LST_SESSION=$$
lst new_session read/write
lst add_group servers 5.6.128.233@o2ib
lst add_group readers 5.6.132.30@o2ib
lst add_group writers 5.6.132.30@o2ib
lst add_batch bulk_rw
lst add_test --batch bulk_rw --concurrency 8 --from readers --to servers \
brw read size=1M
lst add_test --batch bulk_rw --concurrency 8 --from writers --to servers \
brw write size=1M

# start running
lst run bulk_rw
# display server stats for 180 seconds
lst stat servers & sleep 180
```

```
lst stop bulk_rw
# tear down
lst end_session
pkill lst


#!/bin/bash -x
DATADIR="/ws/data"
TAG="1s1c"
DT=`date '+%d_%m_%y_%Hh_%Mm_%Ss'`
dstat -C 0,1,2,3,4,5,6,7 --output $DATADIR/dstat.$TAG.$DT.csv 1 >
$DATADIR/dstat.$TAG.$DT.txt &
mkdir -p $DATADIR
./lnet_ib.sh > $DATADIR/lnet.$TAG.$DT.txt
pkill python
```

5. Benchmarking metadata operations with Metabench on Lustre

Metabench is tool to stress metadata operations like file creation, stat, unlink, delete etc. This can be run on single client or multiple clients with MPI. Following tests should be conducted with Lustre:

- file creates, stat, unlink and delete with single client and MDS with 4,8,16,32 & 64 processes
- file creates, stat, unlink and delete with multiple clients and MDS with 4,8,16,32 & 64 processes

Invoke metabench command as follows:
./mpirun -np 4 ./metabench -w /mnt/lustre -c 30400 -C -S -k -D
Make sure lustre filesystem is unmounted and mounted again between runs.

6. Benchmarking Lustre Client with PIOS

PIOS is parallel I/O simultor designed to mimic common I/O patterns of high performance computing applications. PIOS can generate I/O patterns based on writing number of regions where each region is composed of number of same sized chunks. PIOS has ability to vary number of threads, regions, chunks and introduce randomness in each of the parameters.
Following test cases must be run with PIOS on Lustre.

- PIOS on single Lustre client writing single shared file
- PIOS on single Lustre client reading and verifying single shared file
- PIOS on single Lustre client writing files per processes
- PIOS on single Lustre client reading and verifying files per processes

Invoke PIOS as follows for large I/O tests:

```
./pios -t 4,8,16,32,64 -n 8192 -c 1M -s 4M -o 4M -p /mnt/lustre
./pios -t 4,8,16,32,64 -n 8192 -c 1M -s 4M -o 4M -p /mnt/lustre –verify
./pios -t 4,8,16,32,64 -n 8192 -c 1M -s 4M -o 4M –load=fpp -p /mnt/lustre
./pios -t 4,8,16,32,64 -n 8192 -c 1M -s 4M -o 4M –load=fpp -p /mnt/lustre --verify
```

Invoke PIOS as follows for small I/O tests:

```
./pios -t 4,8,16,32 -n 8192 -c 4k,8k,16k,32k,64k,128k -s 128k -o 128k -p /mnt/lustre
./pios -t 4,8,16,32 -n 8192 -c 4k,8k,16k,32k,64k,128k -s 128k -o 128k -p /mnt/lustre –
```

verify
./pios -t 4,8,16,32 -n 8192 -c 4k,8k,16k,32k,64k,128k -s 128k -o 128k -p /mnt/lustre –load=fpp -p /mnt/lustre
./pios -t 4,8,16,32 -n 8192 -c 4k,8k,16k,32k,64k,128k -s 128k -o 128k -p /mnt/lustre –load=fpp -p /mnt/lustre –verify

7. Benchmarking Lustre clients with IOR

Interleaved or Random (IOR) benchmarks is developed by the Scalable I/O Project (SIOP) at LLNL. It is used for benchmarking parallel file systems using POSIX, MPIIO, or HDF5 interfaces. IOR is opensource and freely available at http://sourceforge.net/projects/ior-sio/ IOR has Lustre specific settings which can be used to tune IOR performance on Lustre. This document assumes that IOR is compiled with Lustre support and proper MPI libraries (LAMMPI) are installed on all Lustre clients.

A typical IOR profile for Lustre looks like:

```
IOR START
testFile = /mnt/lustre/regression
filePerProc=0
api=POSIX
repetitions=3
verbose=1
blockSize=32g
transferSize=1m
verbose=1
writeFile=1
readFile=1
maxTimeDuration=900
lustreStripeCount=-1
keepFile=0
useO_DIRECT=0
RUN
IOR STOP
```

Please refer to table below for IOR test cases needed to be run on Lustre clients. IOR should be run with 1 process per cpu. Disks failures in a RAID array can be simulated by using the following commands:

```
# Simulate a disk failure in RAID6 array md0
mdadm –manage –set-faulty /dev/mds0 /dev/dsk/c0t1d0
# Verify resync with spare disk has started
cat /proc/mdstat
```

No	State of Filesystem	Failure Modes	IOR Configuration
1.	Filesystem is empty	No failures	Single Shared File with DirectIO
			Files per process with DirectIO
			Single Shared File without DirectIO

			Files per process without DirectIO
			Single Shared File with DirectIO on stripecount=1
			Files per process with DirectIO on stripecount=1
			Single Shared File without DirectIO on stripecount=1
			Files per process without DirectIO on stripecount=1
			Single Shared File with DirectIO with transfersize=128m
			Files per process with DirectIO with transfersize=128m
			Single Shared File without DirectIO with transfersize=128m
			Files per process without DirectIO with transfersize=128m
		One RAID6 group on each OSS re-syncing with spare disk (assumed a disk failure	Single Shared File with DirectIO
			Files per process with DirectIO
			Single Shared File without DirectIO
			Files per process without DirectIO
		One RAID6 group on each OSS is offline (assumed to be undergoing lfsck)	Single Shared File with DirectIO
			Files per process with DirectIO
			Single Shared File without DirectIO
			Files per process without DirectIO
2	Filesystem is 50% full	No failures	Single Shared File with

			DirectIO
			Files per process with DirectIO
			Single Shared File without DirectIO
			Files per process without DirectIO
		One RAID6 group on each OSS re-syncing with spare disk (assumed a disk failure	Single Shared File with DirectIO
			Files per process with DirectIO
			Single Shared File without DirectIO
			Files per process without DirectIO
3	Filesystem is 90% full	No failures	Single Shared File with DirectIO
			Files per process with DirectIO
			Single Shared File without DirectIO
			Files per process without DirectIO
		One RAID6 group on each OSS re-syncing with spare disk (assumed a disk failure	Single Shared File with DirectIO
			Files per process with DirectIO
			Single Shared File without DirectIO
			Files per process without DirectIO

II. Test Plan Approval

- Internal review (?)
- External review (?)
- Date(s) agreed to by the client to conduct testing

III. Test Plan – Final Report

Test Results

Test result will be available in the tracking ticket

Test Cases

Conclusions

Conclusions will be added to the tracking tickets

Next Steps

Continual addition of new or preexisting features will need to be added to this living document.