Lustre* Troubleshooting

Legal Disclaimer

- THIS DOCUMENT AND RELATED MATERIALS AND INFORMATION ARE PROVIDED "AS IS" WITH NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. INTEL ASSUMES NO RESPONSIBILITY FOR ANY ERRORS CONTAINED IN THIS DOCUMENT AND HAS NO LIABILITIES OR OBLIGATIONS FOR ANY DAMAGES ARISING FROM OR IN CONNECTION WITH THE USE OF THIS DOCUMENT.
- All products, product descriptions, plans, dates, and figures are preliminary based on current expectations and subject to change without notice. Availability in different channels may vary.
- Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.

Module Overview

Topics covered in this module include:

- Preventing data loss
- Types of Lustre* failures
- Data collection
- Troubleshooting Lustre* - Sample flowchart
- Resolving specific Lustre* issues

Preventing Data Loss
An ounce of prevention is worth a pound of cure

Preventing Data Loss

- Lustre* distributes data across multiple storage targets
- Storage targets constructed from arrays of many disks
- Disks occasionally fail - sectors, heads, firmware, etc.
- With many arrays, mean time between failure decreases
- Storage targets need redundancy and hot spares
- The Lustre* architecture protects against data loss very well
  - Keeping backups of critical files is recommended
- Lustre* does not protect against loss of disks or arrays

Lustre* Failures and Kernel Errors

Types of Lustre* Failures

Automatically recoverable failures

- Normal for distributed file systems with many components
- Lustre* clients and servers maintain file system consistency

Manually recoverable failures

- Loss of AC power
- Component failures without implementing any high availability

Unrecoverable failures

- Complete failure of a storage target, or a system administrator “goof”

Kernel Errors (1 of 3)

Lustre* runs (mostly) in the Linux kernel

Types of kernel errors

- Hard Panic (Aiee!)
- Soft Panic (Oops!)
- Linux Bug (BUG)
- Lustre* Bug (LBUG)

Next two slides will cover these kernel errors

Kernel Errors (2 of 3)

Hard Panic (Aiee!)

- Panic routine called: registers / stack trace on console – crash dump saved
- Capture console / note prior events for analysis / analyze dump
- Reboot node, run hardware diagnostics, put back into service...?

Soft Panic (Oops!)

- Kernel assertion failure, exception, etc.
- Thread killed / system not trusted / should reboot
  - Can force panic with /proc/sys/kernel/panic_on_oops
- Collect console and events data / reboot node / run diags

Kernel Errors (3 of 3)

Linux Bug (BUG)

- Pointer error, divide by zero, etc.
  - Should be caught in a subsequent OOPS
- Lockups
  - Soft lockup (no new tasks started)
  - Hard Lockup (no more interrupts happen, either)
  - Can trigger kernel panic - see doc "lockup-watchdogs.txt"

Lustre* Bug (LBUG)

- Panic-style assertion for the executing thread
- Thread is halted / reboot needed to remove halted thread
  - Thread / system untrusted, gather stack trace and reboot
  - Lustre* log file written to /tmp/lustre-log.{timestamp}
- Or, can force panic with /proc/sys/lnet/panic_on_lbug
  - Collect console data / reboot node / run diags

Data Collection

Items of Concern

If you suspect a Lustre* error, examining the recent kernel logs is a great start in trying to identify a Lustre* issue

Clients

- Lustre*, Applications, Client Hardware, ...

Servers

- Lustre*, Attached Storage, Server Hardware, ...

Networks

- Fabric Manager, Connectors, Cabling, Switches, ...

Some Places to Check

Network Management System (NMS)

- Intel® Manager for Lustre* software, etc.

Consoles

- Servers, Switches, Fabric Manager, ...

Logs *(see additional information in the Elite - Lustre* Debugging Module)*

- Servers, Clients, Switches, ...

Kernel (ring) Buffers *(see additional information in the Elite - Lustre* Debugging Module)*

- Lustre* Servers and Clients

Data Collection

Intel® Manager for Lustre* software - or other NMS

- Intel® Manager for Lustre* software troubleshooting covered in the next module

Simple tools and scripts for system status

- Use pdsh/dshbak to parallelize data collection
  - Start with something simple (clientdf.sh), then expand upon it
  - Then create another script for another check, and another
  - Soon, you will have a set of powerful, easy to use tools

Easy Checks via Scripting

Lustre* provides a *high-level* health status

/proc/fs/lustre/health_check

Should contain the text "healthy" - anything else is bad

"pdsh it" across all the Lustre* nodes

pdsh -g allnodes "lctl get_param health_check" | dshbak -c

Should return "healthy" for all nodes

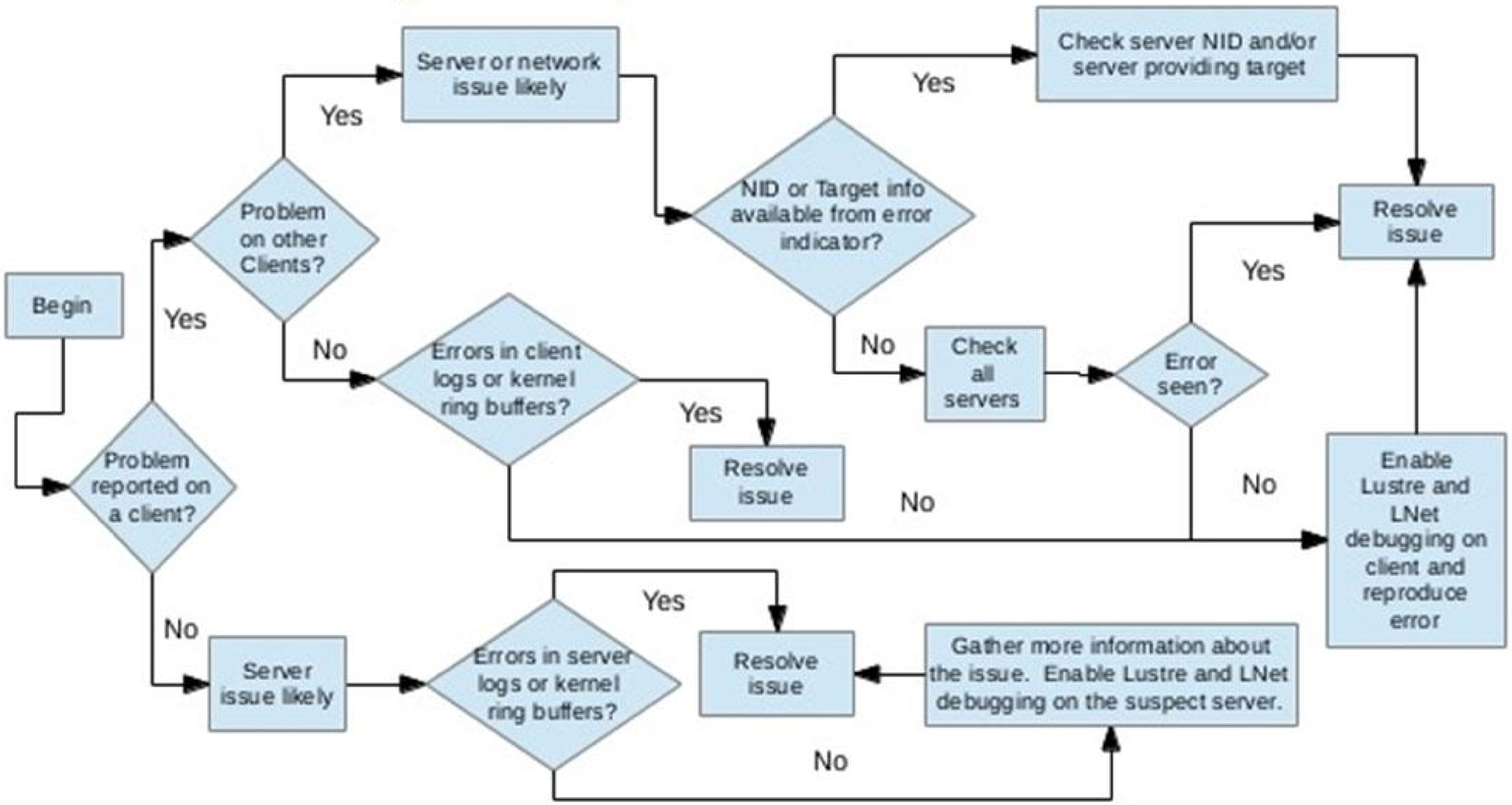
Other easy checks to "pdsh" include

lfs check servers

lctl dl (print device list - all should show UP)

Troubleshooting Lustre* - Sample Flowchart

Troubleshooting Example



Resolving Specific Lustre* Issues

OST Troubleshooting

Deactivating an OST (no new creates)

- A use case is where the OST starts to get too full

Disabling an OST (remove from service)

- A use case is that an entire OST has failed

Marking an OST as degraded (performance)

- A use case is where the RAID set is rebuilding

Deactivating an OST

When to deactivate an OST

- When the OST is in danger of reaching full capacity

Deactivate the OST on the MDS

- Determine the device number of the OST to be deactivated

```
mds# lctl dl | grep " osc "
22 UP osc bleefs-OST0000-osc-ffff8800384efc00 <UUID> 5
23 UP osc bleefs-OST0001-osc-ffff8800384efc00 <UUID> 5
24 UP osc bleefs-OST0002-osc-ffff8800384efc00 <UUID> 5
```

- Deactivate OST0001 via its device number

```
mds# lctl --device 23 deactivate
```

- If OST is still serviceable, do not deactivate on clients
  - This allows reads and writes from a deactivated OST to continue

Verify the correct OST is inactive (IN)

```
mds# lctl dl | grep " osc "
22 UP osc bleefs-OST0000-osc-ffff8800384efc00 <UUID> 5
23 IN osc bleefs-OST0001-osc-ffff8800384efc00 <UUID> 5
24 UP osc bleefs-OST0002-osc-ffff8800384efc00 <UUID> 5
```

Disabling an OST

Used when an OST is completely unavailable

- e.g: fatal RAID controller failure, or server permanently decommissioned

Needs to be disabled on both the MDS and clients:

```
# lctl conf_param osc.<fsname>-<OST name>-*.active=0
```

For example: `# lctl conf_param osc.bleefs-OST0001-*.active=0`

Reads/writes to that OST will fail with I/O error

To enable the OST again:

- Make sure the OST is restored and running

Then run the command: `mds# lctl conf_param osc.bleefs-OST0001-*.active=1`

After enabled, OST will move into recovery

- And after recovery the reads/writes to the OST resume

Marking an OST Degraded

Marking an OST as degraded does not stop IO, but rather it is a hint to the MDS to not allocate new files on that particular OST

On the OSS, write a non-zero value:

```
oss# lctl set_param obdfilter.bleefs-OST0000.degraded=1
oss# lctl get_param –n obdfilter.bleefs-OST0000.degraded=1
```

MDS *is informed* by the OSS that an OST is degraded

- OST is avoided, if possible, in new object allocation
- Helps to prevent global slow-down of file system
- Striping policy may still override
- Should be combined with monitoring of the health of the array

Return to normal by writing zero to the *degraded* file. Flag is reset to zero by a remount of the OST

OST Imbalances - Effect of Full OST

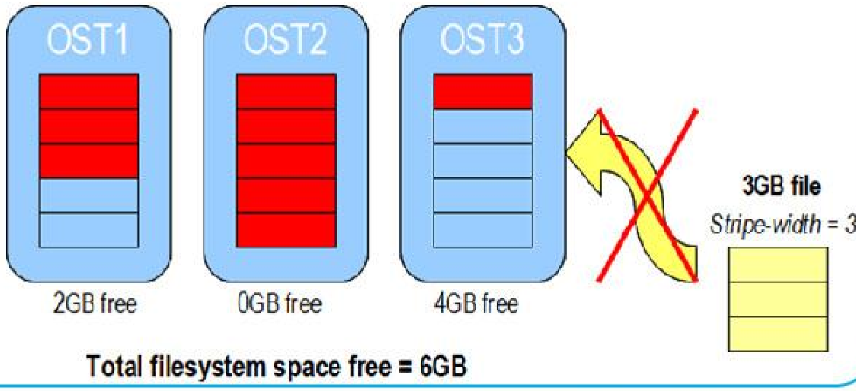
OST Imbalances: OSTs that have a high percentage of utilization – meaning, the amount of free space on the storage target is low

It is fine to have a significant amount of deviation when the capacity utilization for each OST is low, but not so much when the utilization is high

Lustre* attempts to maintain OST balance

If striping policy causes a write to a full OST:

- Application will receive out-of-space error (ENOSPC)
- Even if other OSTs have free space available



OST Imbalances - Query OST Capacity Utilization

Linux *df* reports aggregated utilization

Lustre* *lfs df* reports aggregated and individual target utilization

```
# lfs df
UUID                 1K-blocks       Used   Available Use% Mounted on
bleefs-MDT0000_UUID     786256      35796      698032   5% /lustre[MDT:0]
bleefs-OST0000_UUID   10446648     549016     9373280   6% /lustre[OST:0]
filesystem summary:   10446648     549016     9373280   6% lustre
```

```
# lfs df -i
UUID                    Inodes      IUsed       IFree IUse% Mounted on
bleefs-MDT0000_UUID     524288         24      524264   0% /lustre[MDT:0]
bleefs-OST0000_UUID     153600         88      153512   0% /lustre[OST:0]
filesystem summary:     524288         24      524264   0% /lustre
```

```
# lfs df -h (human readable format)
```

OST Imbalances - Automated Rebalancing

Disks are fastest when they are empty!

MDS has two (2) algorithms for object allocation

- Round Robin (RR)
  - Allocates objects ***equally*** across OSTs
- Quality of Service (QOS)
  - Uses ***weighted free space*** for allocation decisions

Only one of the algorithms is used for each new file

QOS tunables are configurable on the MDS

- Use “lctl get_param” and “lctl set_param” to fetch and set parameters
  - lov.*.qos_threshold_rr is free space skew between OSTs for QOS
  - lov.*.qos_prio_free is weighting given to balance space vs. performance

File Allocation Algorithms – Round Robin

Round Robin (RR)

- Is the **faster** algorithm of the two
- Allocates objects sequentially across all the available OSTs
- Object allocation example using different stripe counts
  - File 1: OST0, OST1, OST2
  - File 2: OST3, OST4, OST5, OST6
  - File 3: OST7, OST0, OST1, OST2, OST3, OST4, OST5
  - File 4: OST6, OST7, OST0, OST1, OST2

Note: The MDS does NOT order OSTs by their index number as shown above. Also, the ordered list is not a static list, as it changes over time

RR always used when OST's are "equally full"

- "Equally full" is defined by the value in:
  /proc/fs/lustre/lov/*/qos_threshold_rr (default value is 17%)
- Meaning: If OST % available space differs by less than 17%, RR is used

File Allocation Algorithms – Quality of Service

- Always used when OST's are not "equally full"
  - OST % available space differs by qos_threshold_rr or more
- OST's are sorted by capacity utilization
- Allocation of objects is based on the sorted list
  - QOS uses a **weighted** free space algorithm
  - % utilization, as well as other factors
- May, but more likely may not, allocate objects equally across OSTs
  - Meaning, some OSTs may get more than one object, while others may get no objects
- Allocation of objects to OST's is impacted by this variable

  /proc/fs/lustre/lov/*/qos_prio_free (default value is 91%)
    - 0 means each OST is allocated once (priority for balance)
    - 100 means OSTs are selected proportional to % utilization
  - Less full OSTs are more likely to be selected more than once

Rebalancing OSTs Manually

Use *lfs_migrate* script to re-balance OSTs

- Simple process

```
Client# lfs_migrate  /lustre              <— Entire file system
Client# lfs_migrate /lustre/bigfiles <— Subset of the file system
```

- What happens in the lfs_migrate process
  - Objects "move" AWAY from more full OSTs, and TO less full OSTs
    - While the Lustre* file reference stays in the same directory
    - Keeps the same stripe count, stripe size, etc.
  - Objects are redistributed by
    - Creating new objects on different OSTs
    - Deleting the old objects
    - Before the deletion of the old objects, a "file" verification takes place
  - In short: Copy, checksum, delete old, rename new

Rebalancing OSTs Manually - Examples

These examples demonstrate how to use *lfs_migrate* to move objects away from full OSTs, as well as to move objects to new or lesser filled OSTs

Example 1: Migrate objects away from OSTs

OST000[2,4] are too full from files from last 2 days

```
$ lfs find /myth -type f -mtime -2 -size +2G \
    --ost myth-OST0002 --ost myth-OST0004 | lfs_migrate -y
```

Example 2: Migrate objects to OSTs

OST000[5,6] are newly added (empty) OSTs

- Move files TO the empty OST's
  - Argument (!) means find files not on the named OSTs

```
$ lfs find /myth -mtime +90 -size +20G -name "*.iso" \
    ! --ost myth-OST0005 ! --ost myth-OST0006 | lfs_migrate -y
```

Storage Target(s) Not in Service

Lustre* uses ldiskfs and ZFS* as storage target (backing) file systems

- Services associated with targets cannot start without the target mounted
- Services unable to start if backing file system corrupted

Causes

- Hard shutdown, hardware failure or errors, operational errors, etc.

Options

- Debugging the storage target(s)
- Run a file system check (e2fsck) to repair the ldiskfs targets
- Perform a “writeconf” to clear and regenerate the targets’ config logs
- Restore from backup and reintegrate restored target(s) into the file system

Storage Target(s) Not in Service - Debugging

Start by debugging the problem

- Attempt to mount the target in service mode
  - Monitor client output as well as syslog/console output on the server
- Attempt to mount the target in non-service mode
  - Pass the "-i nosvc" option to the mount command
    - Mount occurs but Lustre* services do not start
  - If the nosvc mount fails, run e2fsck in "non-fixing mode" (-n arg)
- If e2fsck finds errors, a full e2fsck should be executed
  - Covered later

Storage Target(s) Not in Service - Writeconf (1 of 2)

If the configuration logs get corrupted; a Lustre* writeconf can help get those logs back into a functional state

Performing a writeconf

- Erases the system configuration logs on all targets
- Forces the regeneration of the configuration logs on mount
- MGS gets a new copy of the file system information

Uses

- Recover from catastrophic damage to existing config logs
- Changing a server NID
- Mount an OST on an OSS that is not a designated failnode

Concerns

- File system must be down (all clients and servers un-mounted)
- Erases all pool definitions and changes made with conf_param
  - Keep pool definitions and conf_param settings in a script!

Storage Target(s) Not in Service - Writeconf (2 of 2)

All Lustre* services must be stopped

- Ensure that all clients and all management, metadata, and object storage targets are unmounted
- Ensure that failover software is stopped (if in use)
- Ensure Lustre* backing file systems are healthy
  - mgs# tunefs.lustre --writeconf <MGT disk device>
  - mds# tunefs.lustre --writeconf <MDT disk device>
  - oss# tunefs.lustre --writeconf <OST disk device>

Restart Lustre* in the following order:

- MGS, MDT, OSTs, and then mount all clients

Storage Target(s) Not in Service - LFSCK (1 of 2)

- Most serious of the options - not a Linux fsck
- Use when file system corruption exists
  - Dangling inode – inode exists but missing object on OST
  - Orphaned objects – OST has object but no MDT inode
  - Corrupted MDT – multiple inodes reference objects
- Use after MDT is restored / out of sync with OSTs
- Should be run on a quiesced file system
  - Fastest if run on an idle system
- Time to run depends on size of file system
  - Can take a very long time on a large file system
- Rarely necessary to run Lustre* *fsck*
  - Lustre* can work fine without it
- Run in a *script* session to save the output

Storage Target(s) Not in Service - LFSCK (2 of 2)

The Lustre* fsck (lfsck) process has several steps:

- For all targets, run the Lustre* "e2fsck -f" to fix any problems with the underlying file system
- On the MDS:
  - Create a database of the MDS inodes - MDS DB
  - Run e2fsck in non-fixing mode (-n) – create an MDS DB using the mdsdb option
  - Make the MDS DB (a file) available on all the OSS's
- For every OST:
  - Run e2fsck in non-fixing mode (-n) – create a OST DB using the ostdb option
- Mount all targets as type Lustre*
- Mount the Lustre* file system on any client or MDS
- Run the Lustre* *lfsck* from the node where the Lustre* file system is mounted
  - Lustre*'s *lfsck* uses the mdsdb and ostdb's to resolve corruption

Summary

Summary

Preventing data loss

Types of Lustre* failures

Data collection

Troubleshooting Lustre* - Sample flowchart

Resolving specific Lustre* issues

Congratulations! You have completed:

Lustre* Troubleshooting