Tutorial: How to install, tune and Monitor a ZFS based Lustre file system

2nd annual Lustre Ecosystem Workshop

March 9-10, 2016

Marc Stearman
Lustre Operations Lead
How to build ZFS Lustre file system?

Change mkfs.lustre argument

```
BACKFSTYPE=LDISKFS
```

```
BACKFSTYPE=ZFS
```

![Image](image.jpg)
Expectations

- Why use ZFS and how is it different from Idiskfs?
- What hardware to use? RAID controller or JBOD?
- Operating Systems and Packages
- All about zpools
- The Installation Commands
- Performance and Tuning
- Monitoring
Why did LLNL choose ZFS?

- **Scalability**
  - Dynamic Striping
  - Single OST per OSS
  - Massive Storage Capacity
    - 128 bit from the beginning

- **Cost**
  - Combined RAID+LVM+FS
  - Built for inexpensive disk
  - Can use JBODs, not expensive RAID controllers
  - All Open Source

- **Data Integrity**
  - Copy-on-Write
  - Checksums
    - Metadata and block data
    - Verified on read
    - Stored in parent block
    - Automatically repairs damage
  - Multiple copies of metadata
  - Ditto Blocks
  - Redundancy
    - Stripes
    - N-Way Mirrors
    - Single, Double, Triple Parity
Why did LLNL choose ZFS?

- **Manageability**
  - Online everything
    - Scrubbing
    - Resilvering
    - Pool expansion
    - Configuration Changes
  - Fast file system creation
  - High quality utilities
  - History of changes
  - Event Logging

- **Features**
  - Snapshots
  - Clones
  - Compression
  - Deduplication
  - Dataset Send/Receive
  - Advanced Caching
    - ZFS Intent Log (ZIL)
    - L2ARC
  - Quotas
# Picking the right hardware

<table>
<thead>
<tr>
<th>RAID Controllers</th>
<th>JBODs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adds additional cost</td>
<td>Bare metal is less expensive</td>
</tr>
<tr>
<td>Offload the RAID calculations to special hardware</td>
<td>RAID calculations done by ZFS using node CPU</td>
</tr>
<tr>
<td>Vendor should layout volumes to maximize disk channels</td>
<td>You must optimize pool layout to efficiently use all disk channels</td>
</tr>
<tr>
<td>Improved SES management software</td>
<td>SES management done via “home grown” utils and scripts</td>
</tr>
<tr>
<td>Custom Firmware</td>
<td>Pick vendors with Linux firmware tools</td>
</tr>
<tr>
<td>Integrity Checking done by RAID controller</td>
<td>Integrity checking done by ZFS</td>
</tr>
<tr>
<td>ZFS can detect errors but not fix them</td>
<td>ZFS can detect and fix errors</td>
</tr>
</tbody>
</table>
Operating Systems

- LLNL uses RHEL 6.x, moving soon to RHEL 7.x
- ZFS does not require patching the kernel
- [http://zfsonlinux.org](http://zfsonlinux.org) has a great deal of information
  - Source code for spl and zfs
  - packages for various Linux distributions
  - Large user community using ZFS for not just Lustre

Using the EPEL repo:
$ sudo yum install kernel-devel zfs
ZPOOL 101

- Two main commands
  - ZPOOL: Manages storage pools
  - ZFS: Manages datasets within the storage pools
  - Both have excellent man pages

- The zpool is composed of vdevs, virtual devices comprised of LUNs

- vdevs combine LUNs into redundancy groups, similar to RAID controllers

- zfs datasets are mountable file systems
POOL diagram

Pool made from:
- concrete
- tile
- water
- diving board

Pool full of datasets:
- Big Penguin
- Blue Penguin
- Penguin with bow
- Green Penguin
- Diving Penguin
ZPOOL Diagram

zpool create OSSHOST1 `raidz sda sdb sdc sdd sde raidz sdf sdg sdh sdi sdj`

zpool name: OSSHost1

All datasets share the same space, but can have different properties

dataset_1
dataset_2
dataset_3

/dev/sd[a-e]
/dev/sd[f-j]

two raidz (4+1) vdevs
ZPOOLs and Lustre – General Advice

- Use consistent naming scheme for devices and pools
- Match zpool name to hostname
- Use multipath aliases or udev rules to give LUNs unique names
  - For example:
    - Alias /dev/sda -> /dev/mapper/hostname_1
    - Alias /dev/sdb -> /dev/mapper/hostname_2
    - Or use udev rules to create enclosure slot names, like A[0-9], B[0-9]
    - /lib/udev/vdev_id, /lib/udev/rules.d/69-vdev.rules, and the examples found in /etc/zfs/vdev_id.conf* come with ZFS rpms
- This helps when doing failover or checking the status of pool devices, as /dev/sd* mappings can change on reboot
ZPOOLs on RAID Controllers – an example

- Assume cluster name is “goodbeer”

- RAID controller setup – 60 bay enclosure, 2 OSS nodes
  - Split the drives in half, 30 for each OSS node
  - Create three 8+2 RAID6 volumes/LUNs
  - Stripe the zpool across all three LUNs for each node, creating one vdev
  - Match the hostname with the pool name
  - OSS1
    - zpool create goodbeer1 /dev/alias/goodbeer1_1 \ /dev/alias/goodbeer1_2 /dev/alias/goodbeer1_3
  - OSS2
    - zpool create goodbeer2 /dev/alias/goodbeer2_1 \ /dev/alias/goodbeer2_2 /dev/alias/goodbeer2_3

Note: No raidz because the RAID controller is providing redundancy
ZPOOLs on JBODs – an example

- Assume cluster name is “goodbeer”
- JBOD setup – 60 bay enclosure, 2 OSS nodes
  - Split the drives in half, 30 for each OSS node
  - Create three 8+2 raidz2 vdevs
  - Match the hostname with the pool name
  - OSS1
    - `zpool create goodbeer1 raidz2 /dev/alias/A0 ... /dev/aliasA9
      raidz2 /dev/alias/A10 ... /dev/alias/A19 raidz2 /dev/alias/A20
      ... /dev/alias/A29`
  - OSS2
    - `zpool create goodbeer2 raidz2 /dev/alias/A30 ... /dev/aliasA39
      raidz2 /dev/alias/A40 ... /dev/alias/A49 raidz2 /dev/alias/A50
      ... /dev/alias/A59`
- Depending on your JBOD manufacturer, you may need to choose devices in a different pattern to maximize bandwidth
ZPOOL Import / Export

- Pools are brought online with the `zpool import` command

  ```bash
  [root@porter-mds1:~]# zpool import porter-mds1
  ```

- When done using the pool, use `zpool export`

  ```bash
  [root@porter-mds1:~]# zpool export porter-mds1
  ```
Sharing zpools between nodes

NEVER import an active pool on another node!!!!

OSS Node 1

OSS Node 2

Module A

Module B

Pool 1

Pool 2

Multi-Mount Protection (MMP) is still under development

Need reliable Fencing/STONITH
zpool status

[root@porter1:~]# zpool status
pool: porter1
state: ONLINE
scan: none requested
config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>porter1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>porter1_1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>porter1_2</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>porter1_3</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

errors: No known data errors
[root@porter1:~]#
zpool status

[root@porter-mds1:~]# zpool status
pool: porter-mds1
state: ONLINE
scan: none requested
cfg:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>porter-mds1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A10</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B10</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mirror-11</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A21</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B21</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Other ZPOOL Redundancy Options

- **RAID1**
  - `zpool create <poolname> mirror /dev/sd1 ... /dev/sdN`

- **Traditional RAID10**
  - `zpool create <poolname> mirror sda sdb mirror sdc sdd mirror sde sdf ...`

- **RAID5**
  - `zpool create <poolname> raidz /dev/sd1 ... /dev/sdN`

- **Triple Parity**
  - `zpool create <poolname> raidz3 /dev/sd1 ... /dev/sdN`

- **You must decide your own level of risk vs performance vs capacity. Test various layouts and configurations**
**ldev Makes Creating File Systems Easier**

- From the man page: “ldev can be used to query information about lustre devices configured in /etc/ldev.conf. It is used by the lustre init script.”

/etc/ldev.conf:

```plaintext
#local  foreign/- label device-path [journal-path]
#
porter-mds1  -  lse-MGS0000  zfs:porter-mds1/mgs
porter-mds1  -  lse-MDT0000  zfs:porter-mds1/mdt0
#
porter1    porter2    lse-OST0000  zfs:porter1/lse-ost0
porter2    porter1    lse-OST0001  zfs:porter2/lse-ost0
. . .
porter80   porter79    lse-OST004f  zfs:porter80/lse-ost0
```

[root@porter-mds1:~]# ldev echo %i %d %f %l %n
lse-MGS0000: 0 porter-mds1/mgs lse-MGS0000 172.19.1.165@o2ib100
lse-MDT0000: 0 porter-mds1/mdt0 lse lse-MDT0000 172.19.1.165@o2ib100
[root@porter-mds1:~]#
Example ZFS Lustre Creation

```
mkfs.lustre --mgs --backfstype=zfs --fsname=lsb \
goodbeer-mds1/lsb-mgs mirror A0 B0 mirror A1 B1 \
mirror A2 B2 mirror A3 B3

Note that mkfs.lustre creates the zpool for you

mkfs.lustre --mdt --backfstype=zfs --fsname=lsb \
--index=0 --mgsnode=192.168.64.1@tcp goodbeer-mds1/lsb-mdt0

ldev mkfs.lustre --ost --backfstype=zfs --fsname=%f \
--index=%i --mgsnode=192.168.64.1@o2ib0 --failnode=%N \
goodbeer1/lsb-ost0 /dev/mapper/goodbeer1_1 \
/dev/mapper/goodbeer1_2 /dev/mapper/goodbeer1_3

ldev mkfs.lustre --ost --backfstype=zfs --fsname=%f \
--index=%i --mgsnode=192.168.64.1@o2ib0 --failnode=%N \
goodbeer2/lsb-ost0 /dev/mapper/goodbeer2_1 \
/dev/mapper/goodbeer2_2 /dev/mapper/goodbeer2_3
```
Performance and Tuning

Transform this

Into That!

Image from: academyofmusicanddancenj.com

Image from: San Francisco Symphony
Performance Issues

- ZFS is slower than Idiskfs when used as an MDS
- Small file I/O is more Metadata intensive

LLNL file systems have 500M – 3.5B files on them

> 90% of files are < 32KB

Users store source repositories and compile in Lustre

Much of this workload is suitable for NFS
Performance Issues

- Some sites run mixed ldiskfs MDS and ZFS OSS
- Tradeoff Performance vs Integrity and Online Tools
- Advantages of ZFS as an MDS node
  - ZFS gives you online file system checking
  - ZFS can detect and repair file system errors for Metadata and Data
  - You can easily expand your MDS volume
  - Don’t have to worry about number of inodes during creation
Ways To Improve ZFS MDS Performance

- Use SSDs instead of SAS drives
- Add more RAM and increase the ARC size
- Lock free space maps (metaslabs) in memory
- Follow the ZIL work – LU-4009
  - The patch in the first comment is being used at LLNL to emulate a ZIL
  - This patch replaces the txg_wait_synced() call with a tunable delay. The delay is intended to take the place of the time it would take to synchronously write out the dirty data.
  - Be aware that no data is guaranteed to be written until the transaction group completes. If the server crashes a small amount of data may not make it to disk.
- DNE in Lustre 2.8 should help improve performance
Basic ZFS/SPL Module Tuning parameters

- `zfs_prefetch_disable`
  - 1 for OSS nodes
  - 0 (default) for MDS nodes

- `metaslab_debug_unload`
  - 1 for OSS nodes
  - 0 or 1 for MDS depending on memory usage

- `zfs_txg_history=120`

- Descriptions of these tunings can be found in the support slides as well as the following man pages:
  - spl-module-parameters(5)
  - zfs-module-parameters(5)
Many parameters can be set with the zpool and zfs commands

Datasets will inherit the settings on the pool

*zpool get all; zfs get all* will list all the parameters

```
[root@porter1:~]# zfs get all | grep compress
porter1  compressratio  1.66x  -
porter1  compression    on     local
porter1  refcompressratio  1.00x  -
porter1/lse-ost0  compressratio  1.66x  -
porter1/lse-ost0  compression    on     inherited from porter1
porter1/lse-ost0  refcompressratio  1.66x  -
[root@porter1:~]#
```

On OSS nodes: “*zfs set compression=on porter1*” will enable compression on the pool. Datasets will inherit that feature.
ZFS parameters and Lustre

- Many Lustre parameters are stored as ZFS dataset parameters

```bash
[root@porter1:~]# zfs get all | grep lustre
porter1/lse-ost0  lustre:svname  lse-OST0001  local
porter1/lse-ost0  lustre:failover.node  172.19.1.168@o2ib100  local
porter1/lse-ost0  lustre:version  1  local
porter1/lse-ost0  lustre:flags  2  local
porter1/lse-ost0  lustre:fsname  lse  local
porter1/lse-ost0  lustre:index  1  local
porter1/lse-ost0  lustre:mgsnode  172.19.1.165@o2ib100  local
[root@porter1:~]#
```
Snapshots

- Snapshots are quickly created and can be mounted as POSIX file systems while Lustre is running.

- If something is horribly broken, you can roll back to the latest snapshot.

- Before doing an OS/Lustre/ZFS upgrade, LLNL will take a snapshot on all the nodes in case something goes wrong.

```
# cideri /root > pdsh -Av service lustre stop

# Stopping Lustre exports the pool; Need to re-import it to take snapshot

# cideri /root > pdsh -av
   pdsh> zpool import -d /dev/mapper `hostname`
   pdsh> zfs snapshot -r `hostname`@PreUpdate_v1
```
Snapshots

- Snapshots will accumulate space the longer they exist.
- When destroyed, the space is reclaimed in the background

```bash
# cider-mds1/ root > zfs list -t snapshot
NAME                               USED  AVAIL  REFER  MOUNTPOINT
cider-mds1@PreUpdate_v1               0      -    30K  -
cider-mds1/lsf-mdt0@PreUpdate_v1  16.0G      -   150G  -
cider-mds1/mgs@PreUpdate_v1        148K      -  5.34M  -

# cider-mds1/ root > zfs destroy -nvr cider-mds1@PreUpdate_v1
would destroy cider-mds1@PreUpdate_v1
would destroy cider-mds1/lsf-mdt0@PreUpdate_v1
would destroy cider-mds1/mgs@PreUpdate_v1
would reclaim 18.7G

# cider-mds1 /root > zfs destroy -r cider-mds1@PreUpdate_v1
```
How many monitors are needed for monitoring?
Useful Commands for Monitoring ZFS

```
zpool iostat -v <interval>
```

```
[root@porter1:~]# zpool iostat -v 1

<table>
<thead>
<tr>
<th>pool</th>
<th>capacity</th>
<th>operations</th>
<th>bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>alloc</td>
<td>free</td>
<td>read</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>porter1</td>
<td>22.0T</td>
<td>43.2T</td>
<td>18</td>
</tr>
<tr>
<td>porter1_1</td>
<td>7.35T</td>
<td>14.4T</td>
<td>6</td>
</tr>
<tr>
<td>porter1_2</td>
<td>7.35T</td>
<td>14.4T</td>
<td>6</td>
</tr>
<tr>
<td>porter1_3</td>
<td>7.35T</td>
<td>14.4T</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pool</th>
<th>capacity</th>
<th>operations</th>
<th>bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>alloc</td>
<td>free</td>
<td>read</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>porter1</td>
<td>22.0T</td>
<td>43.2T</td>
<td>4</td>
</tr>
<tr>
<td>porter1_1</td>
<td>7.35T</td>
<td>14.4T</td>
<td>0</td>
</tr>
<tr>
<td>porter1_2</td>
<td>7.35T</td>
<td>14.4T</td>
<td>1</td>
</tr>
<tr>
<td>porter1_3</td>
<td>7.35T</td>
<td>14.4T</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Useful Commands for Monitoring ZFS

arcstat.py

[root@porter-mds1:~]# arcstat.py 1

<table>
<thead>
<tr>
<th>time</th>
<th>read</th>
<th>miss</th>
<th>miss%</th>
<th>dmis</th>
<th>dm%</th>
<th>pmis</th>
<th>pm%</th>
<th>mmis</th>
<th>mm%</th>
<th>arcsz</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:10:57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:10:58</td>
<td>5.8K</td>
<td>765</td>
<td>13</td>
<td>750</td>
<td>13</td>
<td>15</td>
<td>100</td>
<td>714</td>
<td>14</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:10:59</td>
<td>5.4K</td>
<td>714</td>
<td>13</td>
<td>662</td>
<td>12</td>
<td>52</td>
<td>88</td>
<td>700</td>
<td>13</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:11:00</td>
<td>7.4K</td>
<td>945</td>
<td>12</td>
<td>930</td>
<td>12</td>
<td>15</td>
<td>50</td>
<td>928</td>
<td>13</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:11:01</td>
<td>6.5K</td>
<td>695</td>
<td>10</td>
<td>679</td>
<td>10</td>
<td>16</td>
<td>100</td>
<td>677</td>
<td>10</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:11:02</td>
<td>7.4K</td>
<td>1.2K</td>
<td>15</td>
<td>1.2K</td>
<td>15</td>
<td>26</td>
<td>100</td>
<td>1.2K</td>
<td>16</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:11:03</td>
<td>5.1K</td>
<td>597</td>
<td>11</td>
<td>583</td>
<td>11</td>
<td>14</td>
<td>100</td>
<td>553</td>
<td>11</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:11:04</td>
<td>4.4K</td>
<td>637</td>
<td>14</td>
<td>615</td>
<td>14</td>
<td>22</td>
<td>100</td>
<td>626</td>
<td>15</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:11:05</td>
<td>4.7K</td>
<td>486</td>
<td>10</td>
<td>460</td>
<td>9</td>
<td>26</td>
<td>100</td>
<td>474</td>
<td>10</td>
<td>47G</td>
<td>62G</td>
</tr>
<tr>
<td>15:11:06</td>
<td>5.2K</td>
<td>516</td>
<td>9</td>
<td>500</td>
<td>9</td>
<td>16</td>
<td>100</td>
<td>504</td>
<td>10</td>
<td>47G</td>
<td>62G</td>
</tr>
</tbody>
</table>
**Useful Commands for Monitoring ZFS**

```
[root@porter-mds1:~]# cat /proc/spl/kstat/zfs/porter-mds1/txgs

<table>
<thead>
<tr>
<th>txg</th>
<th>birth</th>
<th>state</th>
<th>ndirty</th>
<th>nread</th>
<th>nwritten</th>
<th>reads</th>
<th>writes</th>
<th>otime</th>
<th>qtime</th>
<th>wtime</th>
<th>stime</th>
</tr>
</thead>
<tbody>
<tr>
<td>52811163</td>
<td>7798884088201256 C</td>
<td>52731392</td>
<td>3226624</td>
<td>64561664</td>
<td>831</td>
<td>15611</td>
<td>4999901529</td>
<td>35715</td>
<td>7763</td>
<td>591428399</td>
<td></td>
</tr>
<tr>
<td>52811165</td>
<td>7798894088127796 C</td>
<td>51042816</td>
<td>2472448</td>
<td>65292288</td>
<td>842</td>
<td>15583</td>
<td>5000871472</td>
<td>39858</td>
<td>38187</td>
<td>594869016</td>
<td></td>
</tr>
</tbody>
</table>

... 

| 52811504 | 7800585936267125 C | 30833664 | 3013632 | 45619200 | 925 | 8056 | 5000901784 | 46732 | 96238 | 388466439 |
| 52811506 | 7800595938146164 C | 29492736 | 2585088 | 40825344 | 737 | 7060 | 5001077095 | 50629 | 8839 | 337624618 |
| 52811507 | 7800600939223259 O | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Transaction Group: 52811165

Reads: 842, 2472448 Bytes (2 MB)  --  Average of 2936 Bytes per read
Writes: 15583, 65292288 Bytes (65 MB)  --  Average of 4190 Bytes per write
Open: 5 seconds
Sync time: .594 seconds
Useful Commands for Monitoring ZFS

**ltop**

Filesystem: lse

Inodes: 1134.796m total, 685.484m used (60%), 449.311m free
Space: 5017.085t total, 1820.437t used (36%), 3196.648t free
Bytes/s: 0.198g read, 0.471g write, 0 IOPS
MDops/s: 193 open, 175 close, 1794 getattr, 6 setattr
0 link, 25 unlink, 0 mkdir, 0 rmdir
0 statfs, 18 rename, 49 getxattr

<table>
<thead>
<tr>
<th>OST</th>
<th>S</th>
<th>OSS</th>
<th>Exp</th>
<th>CR</th>
<th>rMB/s</th>
<th>wMB/s</th>
<th>IOPS</th>
<th>LOCKS</th>
<th>LGR</th>
<th>LCR</th>
<th>%cpu</th>
<th>%mem</th>
<th>%spc</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>porter1</td>
<td>4099</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>17323</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>60</td>
<td>35</td>
</tr>
<tr>
<td>(1)</td>
<td>porter2</td>
<td>4099</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15427</td>
<td>20</td>
<td>11</td>
<td>0</td>
<td>59</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td>porter3</td>
<td>4099</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>15397</td>
<td>12</td>
<td>7</td>
<td>0</td>
<td>59</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td>porter4</td>
<td>4099</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18128</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>60</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td>porter5</td>
<td>4099</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18495</td>
<td>11</td>
<td>6</td>
<td>0</td>
<td>60</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

...
Using Splunk to aggregate and visualize data

- Logging MDS data via simple shell scripts
  - `/proc/meminfo`
  - `/proc/slabininfo`
  - `/proc/spl/kstat/zfs/arcstats`
  - `/proc/spl/kstat/zfs/`hostname`/txgs`
  - `lctl get_param mdt.*MDT*.md_stats`

- Splunk will automatically create fields based on key=value

- Use some `grep`, `sed`, `awk`, `tr`, `cut`, and some `perl` to transform data into key=value pairs

- “Memtotal: 131931280 kB” → “MemTotalkB=131931280”
Lustre MDS Operations view from Splunk
Lustre MDS Operations view from Splunk
Lustre MDS Memory Monitoring in Splunk

\texttt{lctl set_param ldlm.namespaces.*.lru_size=2400}

Limiting lock resources avoids out of memory conditions.

Lustre Locks exhausted free memory.
ZFS Module Settings Description

`zfs_prefetch_disable`

Disable the ZFS prefetch. ZFS's prefetch algorithm was designed to handle common server and desktop workloads. Unfortunately, the workload presented by a Lustre server with N threads accessing random blocks in M objects does not fit ZFS's expectations. The result of which appears to be that the prefetch algorithm will read in blocks which are not promptly used wasting IOPs. Testing on zwicky under the SWL has shown that this can cause timeouts when the services are already heavily loaded and don't have IOPs to spare. Therefore, we disable the ZFS prefetch and only read blocks on demand.
metaslab_debug_unload

This option prevents ZFS from unloading the spacemaps from a metaslab once it is read in. This is required because, for reasons not yet fully understood, Lustre's workload results in ZFS repeatedly unloading and loading metaslabs looking for a good one. This results in substantial IO and has a negative impact on performance. By setting this option and preventing the metaslabs from being unloaded we can prevent this IO. However, the cost of doing this is a significantly increased memory footprint. Much of this is solved in the current release, however there are some circumstances where it still helps and if you have enough RAM, it does not hurt to keep the metaslabs in memory.
This option instructs ZFS to keep a history of recent transaction groups. This is useful if you want to track how long transaction groups were open and how long it took to sync that group to disk. This can be useful information to record over time with applications like Splunk or logstash to find trends and correlations with user I/O.