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

2nd annual Lustre Ecosystem Workshop

Marc Stearman
Lustre Operations Lead

March 9-10, 2016

LLNL-PRES-683717
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

How to build ZFS Lustre file system?

Change mkfs.lustre argument

Expectations

- Why use ZFS and how is it different from ldiskfs?
- 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

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

Operating Systems

- LLNL uses RHEL 6.x, moving soon to RHEL 7.x
- ZFS does not require patching the kernel
- 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 localinstall --nogpgcheck http://archive.zfsonlinux.org/epel/zfs-release.el6.noarch.rpm
$ 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:

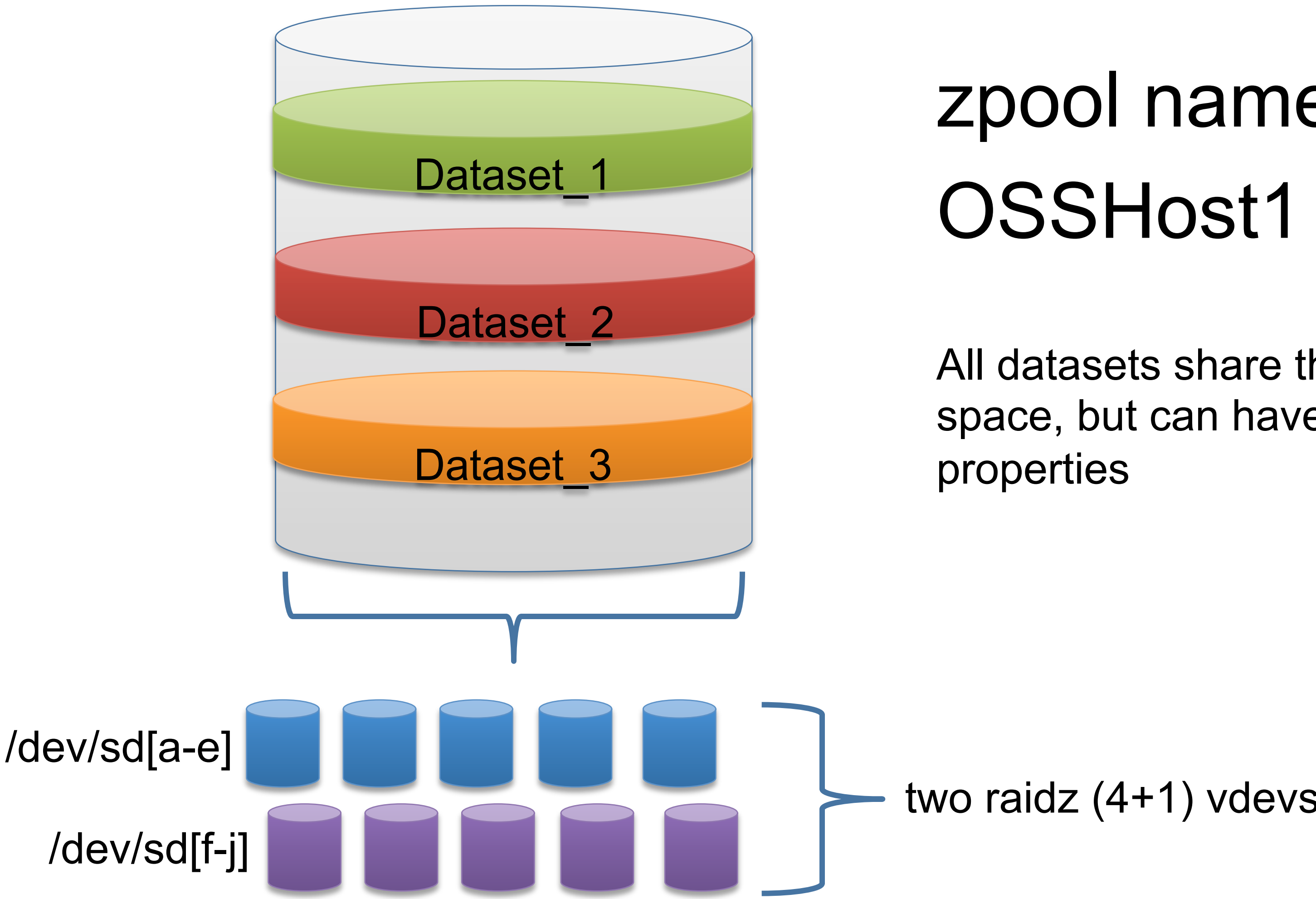
- concrete
- tile
- water
- diving board

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

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

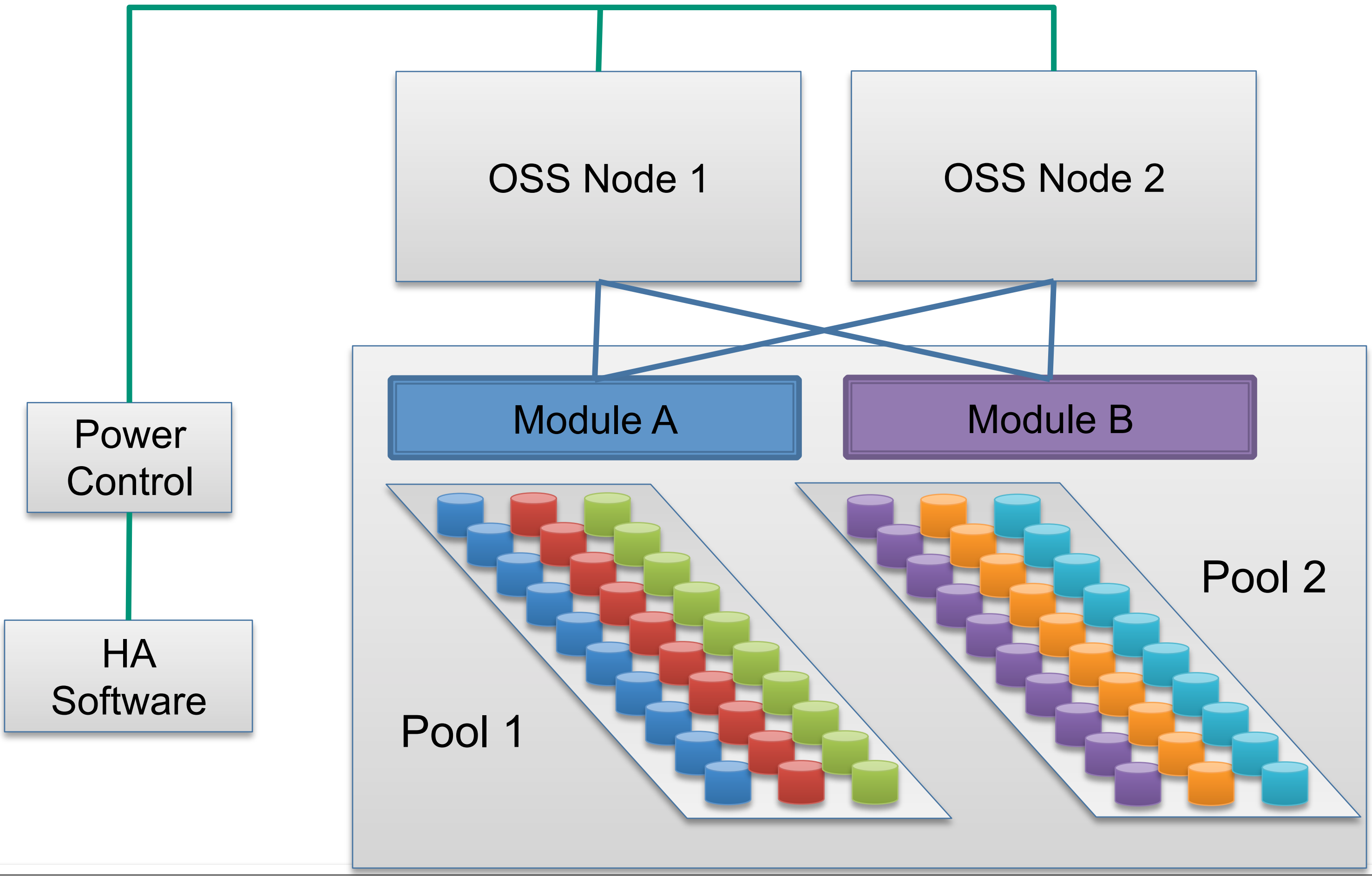
```
[root@porter-mds1:~]# zpool import porter-mds1
```

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

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

Sharing zpools between nodes

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



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:

     NAME          STATE       READ WRITE CKSUM
     porter1       ONLINE         0     0     0
       porter1_1   ONLINE         0     0     0
       porter1_2   ONLINE         0     0     0
       porter1_3   ONLINE         0     0     0

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

zpool status

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

     NAME            STATE        READ  WRITE  CKSUM
     porter-mds1     ONLINE          0      0      0
       mirror-0      ONLINE          0      0      0
         A10         ONLINE          0      0      0
         B10         ONLINE          0      0      0
      ...
       mirror-11     ONLINE          0      0      0
         A21         ONLINE          0      0      0
         B21         ONLINE          0      0      0
```

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:
#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

Image from: academyofmusicanddancenj.com

Image from: San Francisco Symphony

Into That!

Performance Issues

- ZFS is slower than ldiskfs 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)

zpool/zfs get/set

- 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 paramenters and Lustre

- Many Lustre parameters are stored as ZFS dataset parameters

```
[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

```
# 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?

Image from www.space.com

Useful Commands for Monitoring ZFS

`zpool iostat -v <interval>`

```
[root@porter1:~]# zpool iostat -v 1
                  capacity     operations     bandwidth
pool             alloc   free   read  write   read  write
-----------      -----  -----  -----  -----  -----  -----
porter1          22.0T  43.2T     18    135  1.46M  5.70M
  porter1_1      7.35T  14.4T      6     44   498K  1.90M
  porter1_2      7.35T  14.4T      6     44   497K  1.90M
  porter1_3      7.35T  14.4T      6     46   497K  1.90M
-----------      -----  -----  -----  -----  -----  -----

                  capacity     operations     bandwidth
pool             alloc   free   read  write   read  write
-----------      -----  -----  -----  -----  -----  -----
porter1          22.0T  43.2T      4      0  8.50K      0
  porter1_1      7.35T  14.4T      0      0  2.00K      0
  porter1_2      7.35T  14.4T      1      0  3.50K      0
  porter1_3      7.35T  14.4T      1      0  3.00K      0
-----------      -----  -----  -----  -----  -----  -----
```

Useful Commands for Montoring ZFS

arcstat.py

```
[root@porter-mds1:~]# arcstat.py 1
    time  read  miss  miss%  dmis  dm%  pmis  pm%  mmis  mm%  arcsz     c
15:10:57     0     0      0     0    0     0    0     0    0    47G   62G
15:10:58  5.8K   765     13   750   13    15  100   714   14    47G   62G
15:10:59  5.4K   714     13   662   12    52   88   700   13    47G   62G
15:11:00  7.4K   945     12   930   12    15   50   928   13    47G   62G
15:11:01  6.5K   695     10   679   10    16  100   677   10    47G   62G
15:11:02  7.4K  1.2K     15  1.2K   15    26  100  1.2K   16    47G   62G
15:11:03  5.1K   597     11   583   11    14  100   553   11    47G   62G
15:11:04  4.4K   637     14   615   14    22  100   626   15    47G   62G
15:11:05  4.7K   486     10   460    9    26  100   474   10    47G   62G
15:11:06  5.2K   516      9   500    9    16  100   504   10    47G   62G
```

Useful Commands for Montoring ZFS

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

txg	birth	state	ndirty	nread	nwritten	reads	writes	otime	qtime	wtime	stime
52811163	7798884088201256	C	52731392	3226624	64561664	831	15611	4999901529	35715	7763	591428399
52811165	7798894088127796	C	51042816	2472448	65292288	842	15583	5000871472	39858	38187	594869016
. . .											
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 Montoring ZFS

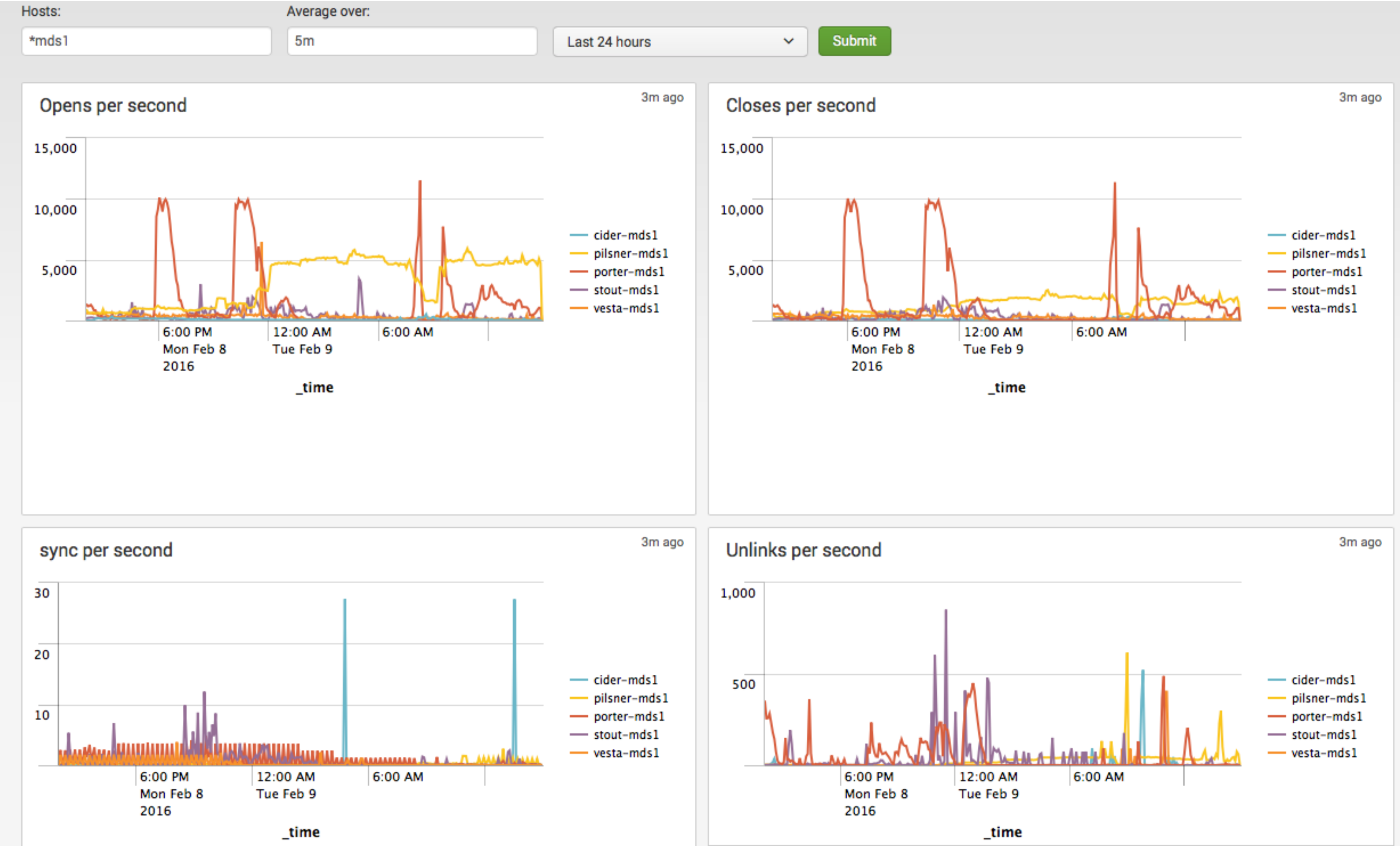
ltop

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

Using Splunk to aggregate and visualize data

- Logging MDS data via simple shell scripts
  - /proc/meminfo
  - /proc/slabinfo
  - /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"

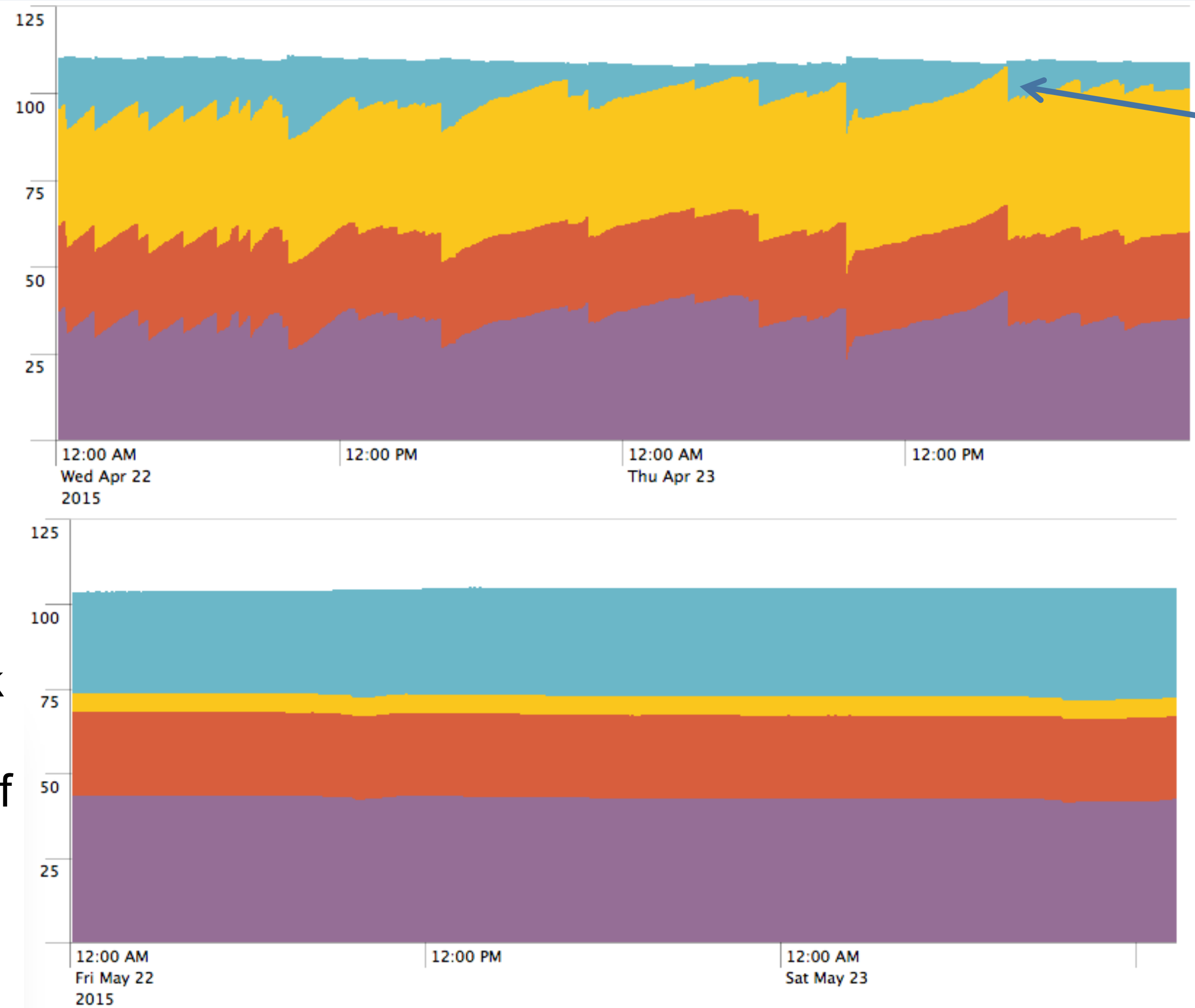
Lustre MDS Operations view from Splunk



Lustre MDS Operations view from Splunk

Lustre MDS Memory Monitoring in Splunk

lctl set_param ldlm.namespaces..lru_size=2400*



Lustre Locks exhausted free memory

Limiting lock resources avoids out of memory conditions

Free Memory
Lustre Locks
ZFS Space Maps
ZFS ARC

Lawrence Livermore National Laboratory

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.

ZFS Module Settings Description

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.

ZFS Module Settings Description

zfs_txg_history

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.