Lustre System Administration Tutorial

Dustin Leverman
HPC Storage Engineer
Oak Ridge National Laboratory

Rick Mohr
HPC Storage Engineer
University of Tennessee

Outline

1:00PM - Configuration/Tuning: (Dustin - 30 min)

- General benchmarking
- Router/client/server tuning

1:30PM - Monitoring/Metrics: (Dustin - 1 hour)

- Performance monitoring
- Health monitoring

2:30PM - 15 minute break

2:45PM - When Things Go Wrong: (Rick - 1 hour)

- Lustre recovery
- Gathering debug data: kdump, lctl dk
- Network debugging
- Repairing filesystem issues

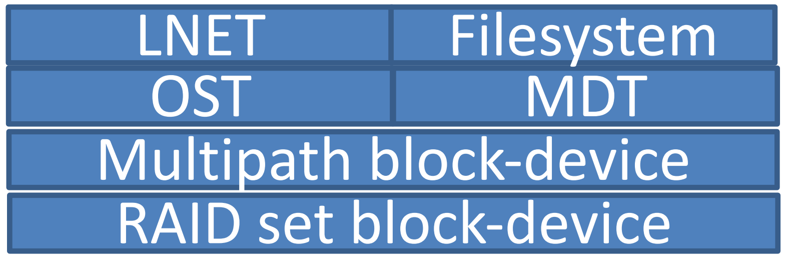
3:45PM - Other Useful Admin Info: (Rick - 30 min)

- Striping
- How RR and QOS allocator work
- Layouts
- PFL
- DoM

4:15PM – 45 minute panel session

Benchmarking

- Keep vendors accountable for requirements and system acceptance
- Establishing a performance baseline helps to detect performance issues later on.
- Benchmark at the different system layers:
  - Block-device/multipath-device (XDD/sgp_dd)
    - ZFS dRAID exception
  - LDISKFS/ZFS (fio)
  - LNET (lnet-selftest)
  - Filesystem data (obdfilter-survey)
  - Filesystem metadata (mdtest)
  - Filesystem data and metadata (IOR)



- There are many tools that can be used, these are just examples

Benchmarking block/multipath device

- Tuning can be different for block and multipath devices. Need to check both.
- Testing /dev/sd vs. /dev/sg can tell you how caching is impacting performance.
  - Need to understand connectivity to RAID system if IB/SAS/FC attached so you don't stress out the same LUN more than once
- XDD or sgp_dd are good tools to use
- XDD example:
  - /opt/xdd/bin/xdd.linux -rwratio 100 -targets 10 /dev/sg{0,2,4,6,8,30,32,34,36,38} -sgio -reqsize 1 -numreqs 10000000 -blocksize 1048576 -verbose -passes 3 -queuedepth 16 -timelimit 120 -seek truesequential -datapattern random

Benchmarking LDISKFS/ZFS

- You can use a tool like fio to write data to ldiskfs or ZFS to test your performance tunables
- Run streaming and random reads/writes
- Test metadata performance
  - Can impact the OST metadata lookup

Benchmarking LNET

- Lustre has a built in tool called lnet-selftest that will exercise the network only (no disk activity)
  - Pick which servers and clients to use
  - Reads/writes
  - Data size
  - Concurrency (simultaneous number of threads participating)
- This will allow you to ensure that the network is performing and tuned
- Very well documented here:
  - http://wiki.lustre.org/LNET_Selftest
  - We don't run this frequently, but we reference this documentation when we do (you don't have to memorize all this stuff)
  - Rick will be giving an example later

Benchmarking filesystem data layer

- Lustre has a built in tool called obdfilter-survey that will exercise the disk layer only (no LNET activity)
- This will allow you to ensure the disks are performing and tuned independent of LNET
- Note: This test can be destructive; run these tests before production
- More info on Lustre Wiki: http://wiki.lustre.org/OBDFilter_Survey

Benchmarking filesystem data layer

- Example:
  - modprobe obdecho
  - mkdir -p /tmp/obdfilter-survey_output
  - nobjlo=1 nobjhi=16 thrlo=1 thrhi=1024 size=32768 rslt_loc=/tmp/obdfilter-survey_output targets="testfs-OST0000 testfs-OST0001 testfs-OST0002" case=disk obdfilter-survey
- Tunables:
  - nobj[lo|hi]: concurrent object count per OST to iterate over
  - thr[lo|hi]: Thread range to iterate over
  - size: total amount of data to be written
    - Should target something that is >2x larger than WBC size of RAID controller
  - rslt_loc: Where to write the result data to
- You should write a script to coordinate the obdfilter-survey processes per node if you have a shared RAID sub-system (DDN, NetApp E-series, etc…)

Filesystem-level benchmarking

Node count:

- "Hero" performance (max performance possible)
  - Usually ~10-30% of total node count at large-scale
  - Node placement may be important (network layout, router layout, etc…)
- Single-client
  - Helps to understand what a small job will see
  - Helps to understand scaling behavior
- All clients (max performance from all clients)
  - Generally ~10-30% slower than "hero" at large-scale
  - Important to understand for full-scale jobs

Thread count:

- Single-threaded
  - Most common use case for users
- Multi-threaded
  - Max possible performance per node

Lustre filesystem metadata benchmarking

- mdtest is a very common tool for exercising filesystem metadata from multiple clients using MPI
- Considerations:
  - shared directory
  - unique directory
  - just metadata (zero-length files) or file IO too?
- Lots of tunables:

#-b: branching factor of hierarchical directory structure
#-B: no barriers between phases (create/stat/remove)
#-c: collective creates: task 0 does all creates and deletes
#-C: only create files/dirs
#-d: the directory in which the tests will run
#-D: perform test on directories only (no files)
#-e: number of bytes to read from each file
#-E: only read files
#-f: first number of tasks on which the test will run
#-F: perform test on files only (no directories)
#-h: prints help message
#-i: number of iterations the test will run
#-I: number of items per tree node
#-l: last number of tasks on which the test will run
#-L: files/dirs created only at leaf level
#-n: every task will create/stat/remove # files/dirs per tree
#-N: stride # between neighbor tasks for file/dir stat (local=0)
#-p: pre-iteration delay (in seconds)
#-r: only remove files/dirs
#-R: randomly stat files/dirs (optional seed can be provided)
#-s: stride between the number of tasks for each test
#-S: shared file access (file only, no directories)
#-t: time unique working directory overhead
#-T: only stat files/dirs
#-u: unique working directory for each task
#-v: verbosity (each instance of option increments by one)
#-V: verbosity value
#-w: number of bytes to write to each file
#-y: sync file after write completion
#-z: depth of hierarchical directory structure

Lustre filesystem metadata benchmarking

File creates, metadata only, unique-directory, 3 iterations, 5-minute delay

```
#BSUB -q storage                                  # Job queue
#BSUB -o mdtest_unique_dir_multi-node.o%J   # output is sent to file job.output
#BSUB -e mdtest_unique_dir_multi-node.e%J   # error is sent to file job.error
#BSUB -J mdtest_unique_dir_multi-node         # name of the job
#BSUB -nnodes 630                                 # Number of nodes to use in the job
#BSUB -W 360                                      # wallclock -W [hour:]minute[/host_name | /host_model]
#BSUB -U PT
#BSUB -P ACCEPTANCE

MOUNT="alpine"

BINDIR="/gpfs/alpine/stf002/scratch/leverman/alpine_acceptance"
OUTDIR="$BINDIR/${LSB_JOBID}_md_test"
[ -e $OUTDIR ] || {
 mkdir -p $OUTDIR
}
cd $BINDIR
module load gcc
jsrun -n 630 -c ALL_CPUS -a 20 -X 1 $BINDIR/build/mdtest_build/mdtest -n 32768 -p 300 -F -u -C -r -i 3 -v -v -u
$OUTDIR
```

Lustre filesystem data benchmarking

- IOR is a very common tool for exercising filesystem from multiple clients using MPI
- Considerations:
  - FPP or SSF
  - Random vs. Sequential workload (random is more realistic on an aging system)
  - picking IO size (alignment with RAID engine or user workload)
  - picking the amount of data (want to write for long enough to exceed client, server, and RAID engine caches
  - Don't let vendors stonewall, pre-create, etc… as part of acceptance
- Lots of tunables:

-a S api -- API for I/O [POSIX|MPIIO|HDF5|HDFS|S3|S3_EMC|NCMPI]
-A N refNum -- user reference number to include in long summary
-b N blockSize -- contiguous bytes to write per task (e.g.: 8, 4k, 2m, 1g)
-B useO_DIRECT -- uses O_DIRECT for POSIX, bypassing I/O buffers
-c collective -- collective I/O
-C reorderTasksConstant -- changes task ordering to n+1 ordering for readback
-d N interTestDelay -- delay between reps in seconds
-D N deadlineForStonewalling -- seconds before stopping write or read phase
-e fsync -- perform fsync upon POSIX write close
-E useExistingTestFile -- do not remove test file before write access
-f S scriptFile -- test script name
-F filePerProc -- file-per-process
-g intraTestBarriers -- use barriers between open, write/read, and close
-G N setTimeStampSignature -- set value for time stamp signature
-h showHelp -- displays options and help
-H showHints -- show hints
-i N repetitions -- number of repetitions of test
-I individualDataSets -- datasets not shared by all procs [not working]
-j N outlierThreshold -- warn on outlier N seconds from mean
-J N setAlignment -- HDF5 alignment in bytes (e.g.: 8, 4k, 2m, 1g)
-k keepFile -- don't remove the test file(s) on program exit
-K keepFileWithError -- keep error-filled file(s) after data-checking
-l data packet type-- type of packet that will be created [offset|incompressible|timestamp|o|i|t]
-m multiFile -- use number of reps (-i) for multiple file count
-M N memoryPerNode -- hog memory on the node (e.g.: 2g, 75%)
-n noFill -- no fill in HDF5 file creation
-N N numTasks -- number of tasks that should participate in the test
-o S testFile -- full name for test
-O S string of IOR directives (e.g. -O checkRead=1,lustreStripeCount=32)
-p preallocate -- preallocate file size
-P useSharedFilePointer -- use shared file pointer [not working]
-q quitOnError -- during file error-checking, abort on error
-Q N taskPerNodeOffset for read tests use with -C & -Z options (-C constant N, -Z at least N) [!HDF5]
-r readFile -- read existing file
-R checkRead -- check read after read
-s N segmentCount -- number of segments
-S useStridedDatatype -- put strided access into datatype [not working]
-t N transferSize -- size of transfer in bytes (e.g.: 8, 4k, 2m, 1g)
-T N maxTimeDuration -- max time in minutes to run tests
-u uniqueDir -- use unique directory name for each file-per-process
-U S hintsFileName -- full name for hints file
-v verbose -- output information (repeating flag increases level)
-V useFileView -- use MPI_File_set_view
-w writeFile -- write file
-W checkWrite -- check read after write
-x singleXferAttempt -- do not retry transfer if incomplete
-X N reorderTasksRandomSeed -- random seed for -Z option
-Y fsyncPerWrite -- perform fsync after each POSIX write
-z randomOffset -- access is to random, not sequential, offsets within a file
-Z reorderTasksRandom -- changes task ordering to random ordering for readback

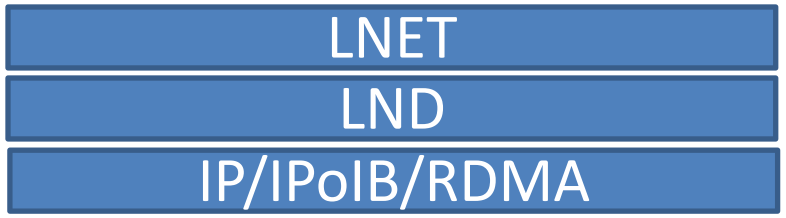
Lustre filesystem data benchmarking

FPP, read/write, 16MB transfer size

```
#!/bin/bash
#BSUB -q storage                    # Job queue
#BSUB -o IOR_fpp_32MB_seq_alpine.o%J     # output is sent to file job.output
#BSUB -e IOR_fpp_32MB_seq_alpine.e%J     # error is sent to file job.error
#BSUB -J IOR_fpp_32MB_seq_alpine        # name of the job
#BSUB -nnodes 504                   # Number of nodes to use in the job
#BSUB -W 240                        # wallclock -W [hour:]minute[/host_name | /host_model]
#BSUB -P ACCEPT
#BSUB -alloc_flags "smt4 isolategpfs"    # Isolate GPFS processes and configure for SMT4

MOUNT=$(pwd | awk -F/ '{print $3}')
BDIR="/gpfs/alpine/stf002/scratch/leverman/alpine_acceptance"
TDIR="$BDIR/ior_testdir"
ITERS=3
BSIZE="7168g"
INTERFACE="POSIX”
TSIZE="16m"

mkdir -p ${TDIR}
cd ${BDIR}
module load gcc

date
echo "POSIX read/write run for seq file per process 16MB transfer size, 20min"
jsrun -n 504 -c ALL_CPUS -a 1 ${BDIR}/build/ior-3.1.0/src/ior -g -d 360 -o ${TDIR}/POSIX_fpp_ior -F -i ${ITERS} -b ${BSIZE} -t ${TSIZE} -w -r -a ${INTERFACE} -e -v -v
date
exit 0
```

Lustre Tuning (general – ALL)

- Tuning BIOS
  - Disable c-states
  - Put in “performance” mode
  - Performance power governor in OS
- Ko2iblnd (our lustre file systems are all IB attached – that will be the assumption for these slides)
  - options ko2iblnd ib_mtu=2048 timeout=100 credits=2560 ntx=5120 peer_credits=63 concurrent_sends=63 fmr_pool_size=1280 fmr_flush_trigger=1024
- LNET
  - /etc/modprobe.d/lnet.conf
    - options lnet check_routers_before_use=1 router_ping_timeout=120 dead_router_check_interval=50 avoid_asym_router_failure=0 live_router_check_interval=50
  - /etc/lnet.conf
    - net:
    - - net type: o2ib2
    - local NI(s):
    - - nid:
    - interfaces:
    - 0: ib1
    - tunables:
    - peer_timeout: 180
    - peer_credits: 63
    - peer_buffer_credits: 0
    - credits: 2560
    - global:
    - discovery: 0
  - Peer Credits
    - Modern systems you generally set peer credits to 63 (may need to be lower with FDR IB – 8 because of concurrent sends issue)
    - Compute vendors may set something specific (need to keep credit the same across all clients and servers)
- Striping (talk about this later)



Lustre Server Tuning

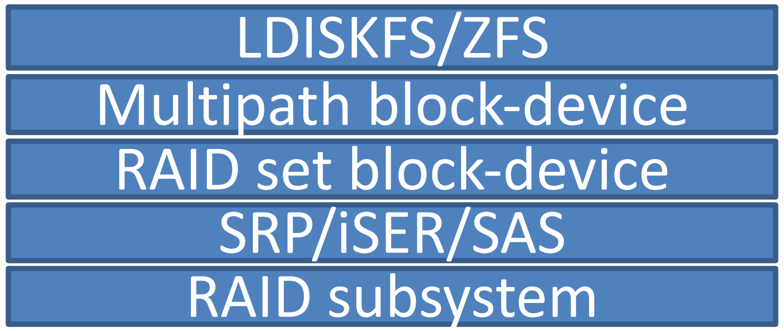
- SRP (for an SFA14KX):
  - /etc/modprobe.d/ib_srp.conf
    - options ib_srp cmd_sg_entries=255 indirect_sg_entries=2048 allow_ext_sg=1 use_blk_mq=N
- Block devices (udev rules for an SFA14KX):

```
KERNEL=="sd*", ENV{ID_VENDOR}=="DDN*", ENV{ID_MODEL}=="SFA14KX*", \
    ATTR{device/timeout}="68", \
    ATTR{queue/scheduler}="deadline", \
    ATTR{queue/nr_requests}="192", \
    ATTR{queue/read_ahead_kb}="0", \
    ATTR{queue/max_sectors_kb}="$attr{queue/max_hw_sectors_kb}”

KERNEL=="dm-*", ACTION=="change", ENV{NCCS_DM_TABLE}=="multipath"      \
    ATTR{queue/scheduler}="deadline",              \
    ATTR{queue/nr_requests="192}",        \
    ATTR{queue/read_ahead_kb}=”0}",        \
    ATTR{queue/max_sectors_kb}=”8192”
```

- Multipathd (for an SFA14KX):

```
device {
        vendor              "DDN"
        product             "SFA14KX"
        prio                "alua"
        prio_args           "exclusive_pref_bit"
        path_grouping_policy   "group_by_prio"
        path_checker          "tur"
        path_selector         "round-robin 0"
        rr_weight             "uniform"
        failback              "2"
        no_path_retry         "12"
        user_friendly_names    "yes"
        dev_loss_tmo          "10"
        fast_io_fail_tmo       "5"
        max_sectors_kb         "8192"
    }
```

- LDISKFS/ZFS tunables:
  - options zfs metaslab_debug_unload=1 zfs_arc_max=150000000000 zfs_vdev_scheduler=deadline zfs_prefetch_disable=1 zfs_dirty_data_max_percent=30 zfs_dirty_data_max_max=60236916326 zfs_dirty_data_max=60236916326 zfs_arc_average_blocksize=2097152 zfs_max_recordsize=2097152 zfs_vdev_aggregation_limit=2097152 zfs_multihost_interval=60000



Lustre Client Tuning

- lctl set_param osc.*.**checksums**=0
  - You may already have network checksums enabled and don't need this
  - Performance penalty
- lctl set_param timeout=600
- lctl set_param ldlm_timeout=200
- lctl set_param at_min=250
- lctl set_param at_max=600
- lctl set_param ldlm.namespaces.*.**lru_size**=128
  - Might be ignored sometimes (current bug LU11518)
  - Low number for computes (generally), high number for login nodes
- lctl set_param osc.*.max_rpcs_in_flight=32
- lctl set_param osc.*.max_dirty_mb=64
- lctl set_param debug="+neterror"
  - Rick will talk more about this later

Lustre Router Tuning

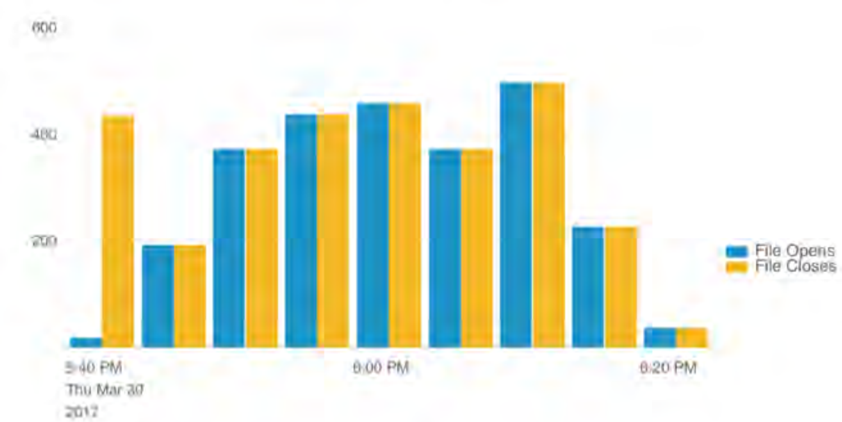
- Check if LNET routing is enabled on this node
  - cat /sys/kernel/debug/lnet/routes
- LNET router buffer sizes
  - Defaults are generally too small
  - Can be changed on the fly
  - How we tune it:
    - tiny: 8192
      - Zero-payload (signals and acks)
    - small: 131072
      - 4k payload (metadata, zero-length file, etc…)
    - large: 4096
      - 1m max payload (file data)
- Different credit sizes per interface?
  - Depends on the networks you are routing between

Performance Monitoring

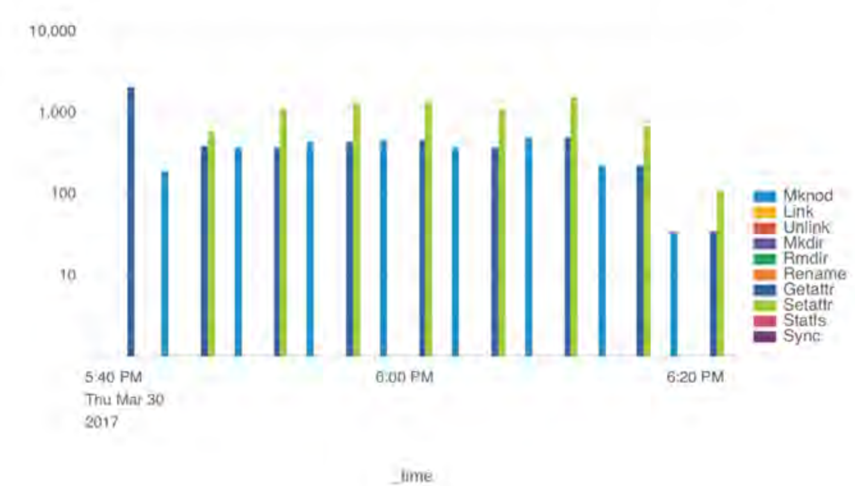
- Jobstats (job-level)
  - We assume single job per node and tag each lustre client with a job ID using the scheduler prologue/epilogue
  - You can gather this data as time-series or just have a report for what the total IO activity for the job was using tools like splunk/influx-Grafana

```
- job_id:          Titan-titan-login1
  snapshot_time:   1556021522
  read_bytes:      { samples:       2, unit: bytes, min:   32768, max: 1015808, sum:        1048576 }
  write_bytes:     { samples:       0, unit: bytes, min:       0, max:       0, sum:              0 }
  getattr:         { samples:       0, unit:  reqs }
  setattr:         { samples:       0, unit:  reqs }
  punch:           { samples:       0, unit:  reqs }
  sync:            { samples:       0, unit:  reqs }
  destroy:         { samples:       0, unit:  reqs }
  create:          { samples:       0, unit:  reqs }
  statfs:          { samples:       0, unit:  reqs }
  get_info:        { samples:       0, unit:  reqs }
  set_info:        { samples:       0, unit:  reqs }
  quotactl:        { samples:       0, unit:  reqs }
[root@atlas-oss1a1 ~]#
```

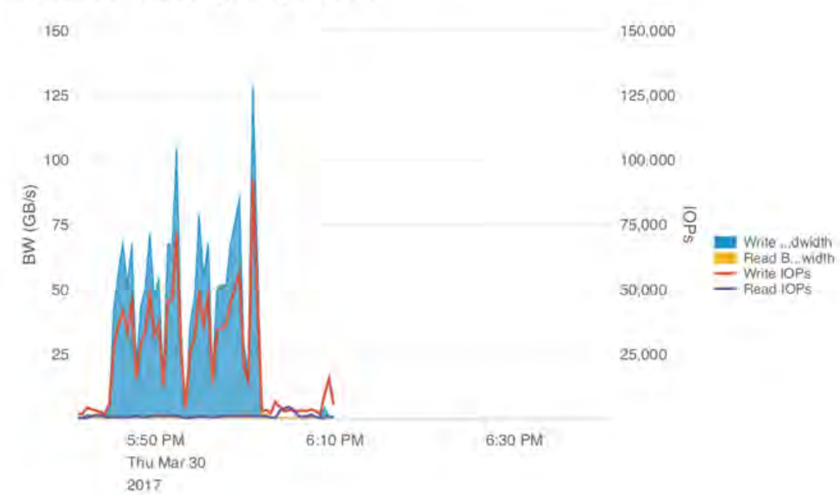
Job Specific I/O Statistics: File Opens & Closes



Job Specific I/O Statistics: Other Metadata Operations



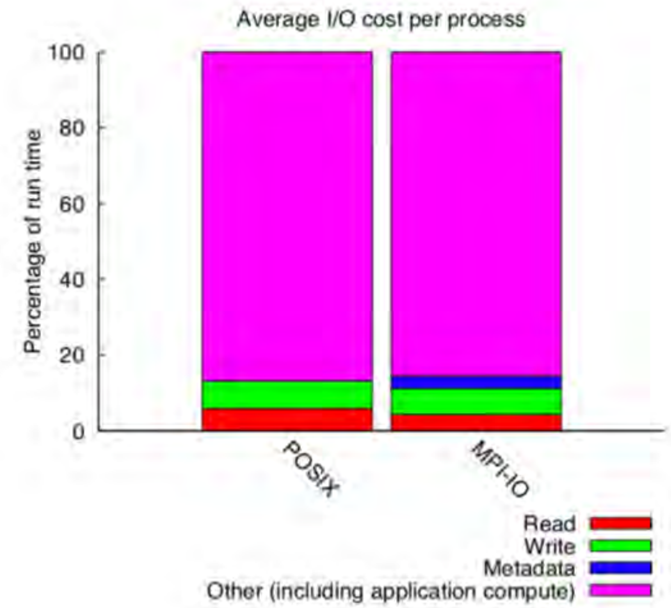
Atlas2: Bandwidth (GB/s) & IOPs

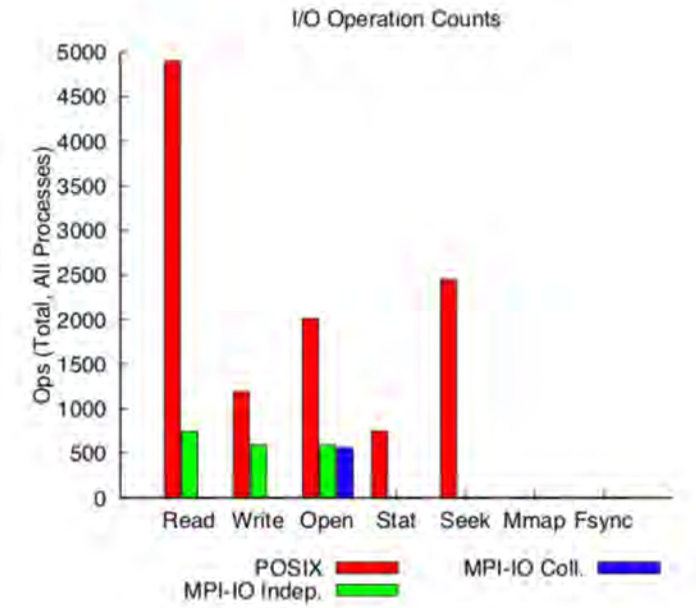


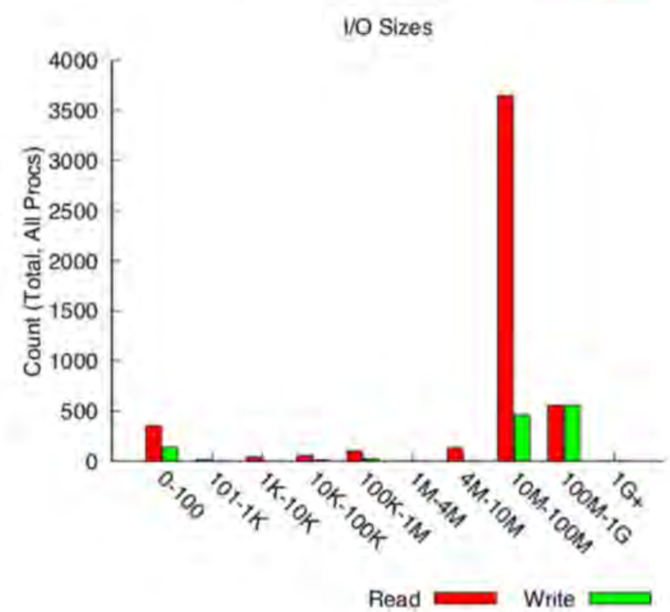
Job Specific I/O Statistics: Write BW & OST Usage

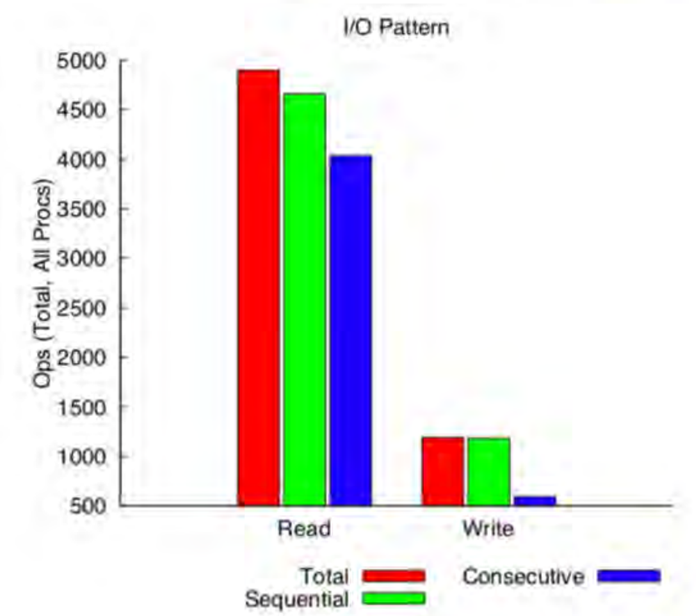
Performance Monitoring

- Darshan (job-level)
  - Load an environment module
  - Users compile code with darshan loaded
  - Darshan intercepts I/Os and gathers statistics
  - Tools exist to visualize data
  - Minimal performance impact









Average I/O per process

	Cumulative time spent in I/O functions (seconds)	Amount of I/O (MB)
Independent reads	10.719584	3342.922423
Independent writes	1.690095	787.716856
Independent metadata	0.001236	N/A
Shared reads	28.561105	10076.175192
Shared writes	46.037887	10014.234111
Shared metadata	0.125903	N/A

Data Transfer Per Filesystem

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/lustre/atlas1	194435.11740	1.00000	240439.05381	0.99543
/lustre/atlas	0.00000	0.00000	1104.70326	0.00457

Most Common Access Sizes

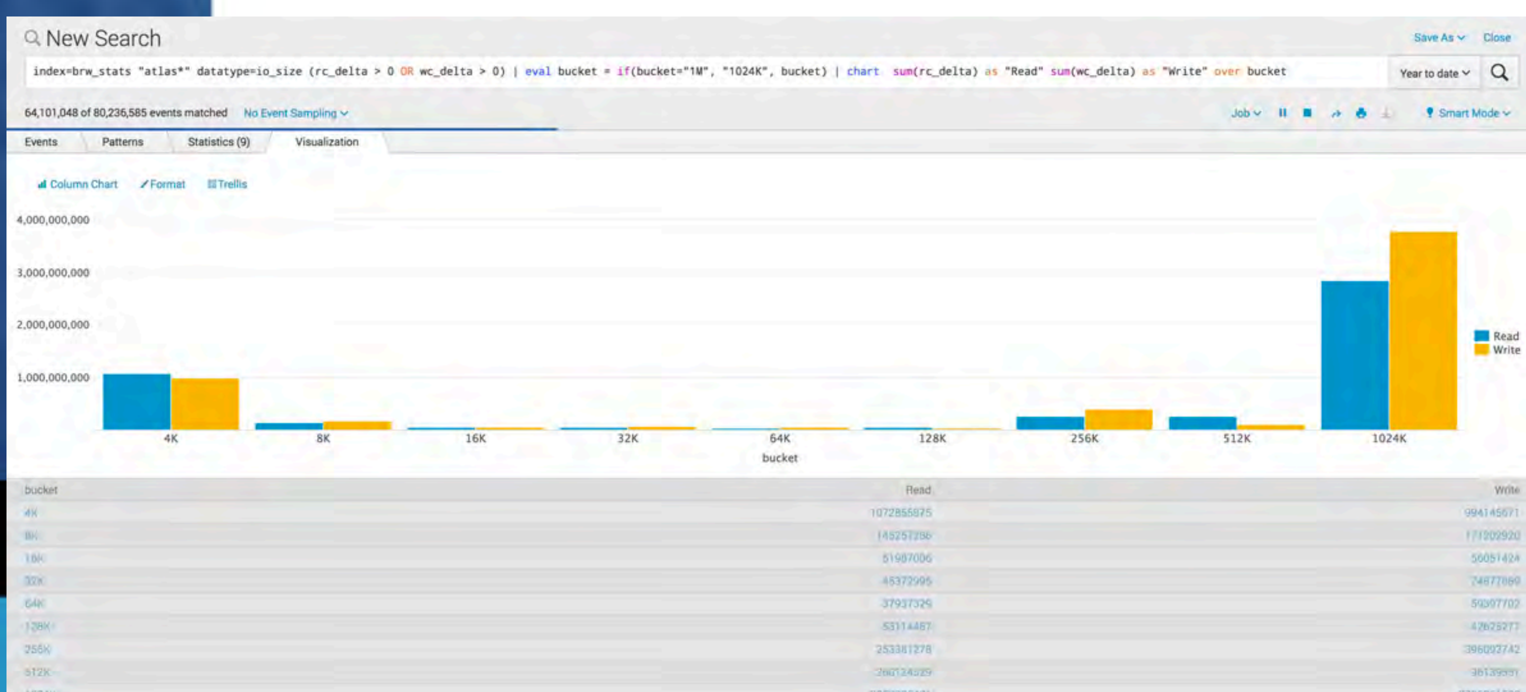
access size	count
14015004	2786
28030008	858
32391408	456
28	216

File Count Summary (estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	18	13G	177G
read-only files	5	8.0G	20G
write-only files	6	2.4G	4.7G
read/write files	1	177G	177G
created files	7	28G	177G

Performance Monitoring

- Brw_stats (OST/MDT target level)
  - Can use a data collector tool (cerebrod, telegraf, etc…) to collect the brw_stats data for each OST
  - Put this data into analytics tool (like splunk) to visualize
  - Dumps the following data:
    - pages per bulk r/w
    - discontiguous pages
    - disk I/Os in flight
    - I/O time (1/1000s)
    - disk I/O size




```
                            read        |     write
disk I/O size          ios   % cum %  |  ios          % cum %
1:                       0   0   0    | 13087   0   0
2:                       0   0   0    |  197   0   0
4:                       0   0   0    |  573   0   0
8:                       7   0   0    |  391   0   0
16:                      0   0   0    | 13275   0   0
32:                      1   0   0    | 7211   0   0
64:                      1   0   0    | 12145   0   0
128:                  4249   0   0    | 35584   0   0
256:                     1   0   0    | 29320   0   0
512:                     1   0   0    | 133477   0   0
1K:                      0   0   0    | 446480   0   0
2K:                      0   0   0    | 984317   0   1
4K:              105868354  35  35    | 3911623   3   4
8K:                1964299   0  36    | 12941202  10  15
16K:               8549687   2  39    | 4857143   4  19
32K:               8640700   2  42    | 3494314   2  22
64K:              10565910   3  45    | 1078249   0  23
128K:              4533522   1  47    | 1639481   1  24
256K:              4850565   1  49    | 1838083   1  26
512K:             37528509  12  61    | 1304059   1  27
1M:              113179958  38 100    | 87936619  72 100
```

Performance Monitoring

- LMT – ltop
  - Collects metadata, bandwidth, and other server-side stats
  - Puts data in a database via data collection tool (cerebrod)
  - Different interfaces to view the data (ltop and lwatch)

```
Filesystem: atlas1
    Inodes:   1024.000m total,    467.689m used ( 46%),    556.311m free
     Space:  14095.912t total,  10328.815t used ( 73%),   3767.097t free
   Bytes/s:      1.654g read,       1.533g write,             6698 IOPS
   MDops/s:  29188 open,     26767 close,     9751 getattr,    1356 setattr
                 0 link,        59 unlink,       4 mkdir,         1 rmdir
                 0 statfs,     104 rename,       0 getxattr
```

>OST S	OSS	Exp	CR	rMB/s	wMB/s	IOPS	LOCKS	LGR	LCR	%cpu	%mem	%spc
(7)	las-oss1a1	20058	0	11	1	45	57594	28	21	6	100	73
(7)	las-oss1a2	20058	0	8	2	35	54866	162	24	6	98	74
(7)	las-oss1a3	20058	0	8	2	31	53153	68	58	5	99	75
(7)	las-oss1a4	20058	0	6	2	36	57998	42	34	6	99	73
(7)	las-oss1a5	20058	0	60	0	79	55888	38	28	6	99	74
(7)	las-oss1a6	20058	0	6	4	39	55211	43	30	6	100	73
(7)	las-oss1a7	20058	0	51	2	69	56434	38	26	6	100	73
(7)	las-oss1a8	20058	0	6	1	42	56182	77	55	6	99	73
(7)	las-oss1b1	20058	0	7	2	29	60166	42	29	6	99	73
(7)	las-oss1b2	20058	0	49	2	82	55486	57	41	6	100	74
(7)	las-oss1b3	20058	0	6	31	64	56988	37	27	6	100	73
(7)	las-oss1b4	20058	0	7	1	27	54338	39	29	6	99	75
(7)	las-oss1b5	20058	0	7	41	69	58951	46	33	6	99	73

- Controller-local IO statistics
  - DDN, NetApp, Adaptec, etc… should present B/W, I/O size, IOPS, latency etc… for LUNs, PDs, host ports, etc…

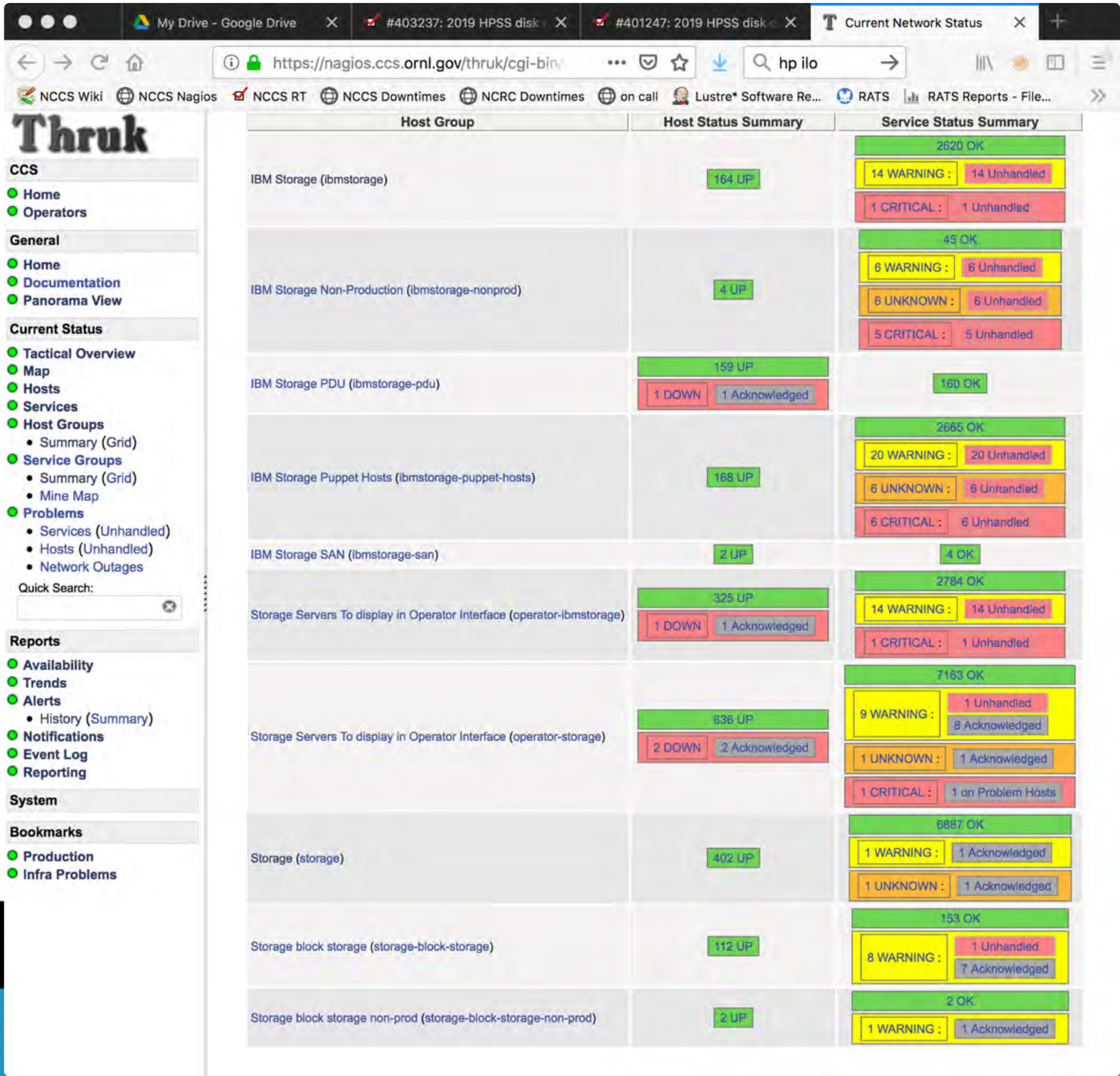
```
******************************************
*      Virtual Disk Rate Statistics      *
******************************************
```

Idx	Fwd Band-Width(KiB/s)	Forwarded IOPS	Avg Read Latency(ms)	Avg Write Latency(ms)	Read Band-Width(KiB/s)	Write Band-Width(KiB/s)	Total Band-Width(KiB/s)	Read IOPS	Write IOPS
0	0.00	0.00	9.09	0.54	1580.95	2663.77	4244.73	1.9	7.4
1	0.00	0.00	0.15	0.00	0.00	0.00	0.00	0.0	0.0
2	0.00	0.00	8.81	0.66	1624.55	2794.02	4418.57	2.1	7.6
3	0.00	0.00	0.11	0.00	0.00	0.00	0.00	0.0	0.0
4	0.00	0.00	8.59	0.55	1761.36	2761.23	4522.59	2.2	8.1
5	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.0	0.0
6	0.00	0.00	8.69	0.53	1595.09	2767.30	4362.38	2.1	7.8
7	0.00	0.00	0.08	0.00	0.00	0.00	0.00	0.0	0.0
8	0.00	0.00	8.89	0.54	1988.54	3230.76	5219.30	2.4	8.7
9	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.0	0.0
10	0.00	0.00	8.47	0.54	1929.38	2898.30	4827.68	2.3	7.8
11	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.0	0.0

OpenSFS

Health Monitoring

- At ORNL we use Nagios
  - Provides a dashboard for system health
- Monitoring
  - OK, warning, critical
- Alerting
  - Business hours
  - non-business hours
  - never page



Health Monitoring

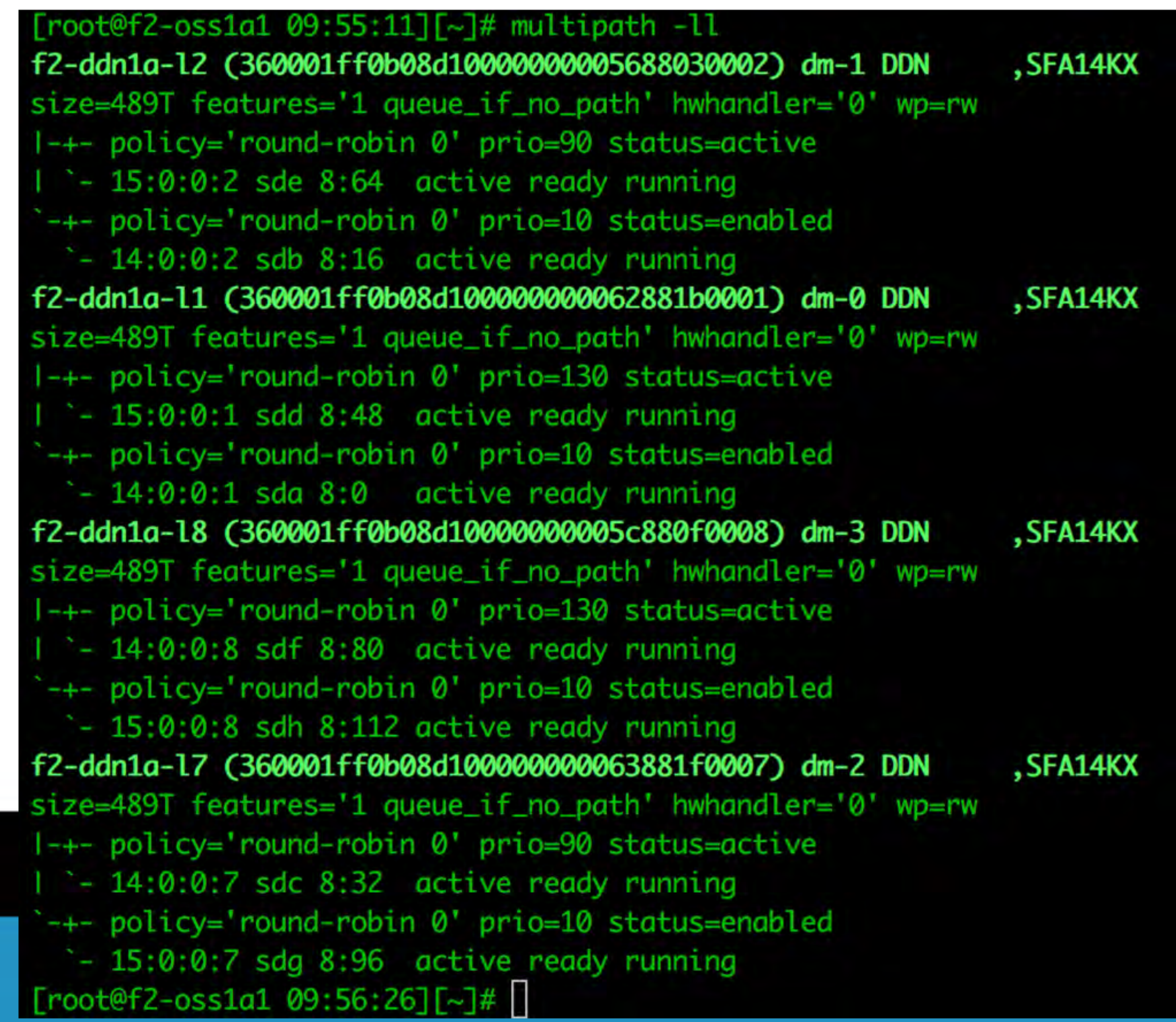
- block-device tuning checks
  - Need to make sure that the IO scheduler, nr_requests, timeouts, etc are tuned correctly.
  - These can be lost after an upgrade.
- Mounted devices check:
  - Make sure that all of your OSTs are mounted
    - sounds ridiculous, but this can happen at 2AM
- Server health (memory/processor/fan/power-supply)
  - Many hardware vendors provides tools already
    - OpenManage (Dell), iLO (HP), etc…
    - ipmitool sdr
      - Hundreds of sensors available

```
[root@f2-oss1a1 09:21:02][~]# ipmitool sdr
Temp             | 29 degrees C      | ok
Temp             | 29 degrees C      | ok
Inlet Temp       | 15 degrees C      | ok
DIMM PG          | 0x00              | ok
NDC PG           | 0x00              | ok
PS1 PG FAIL      | 0x00              | ok
PS2 PG FAIL      | 0x00              | ok
BP0 PG           | 0x00              | ok
BP1 PG           | 0x00              | ok
1.8V SW PG       | 0x00              | ok
2.5V SW PG       | 0x00              | ok
5V SW PG         | 0x00              | ok
PVNN SW PG       | 0x00              | ok
VSB11 SW PG      | 0x00              | ok
VSBM SW PG       | 0x00              | ok
3.3V B PG        | 0x00              | ok
MEM012 VDDQ PG   | 0x00              | ok
MEM012 VPP PG    | 0x00              | ok
MEM012 VTT PG    | 0x00              | ok
MEM345 VDDQ PG   | 0x00              | ok
MEM345 VPP PG    | 0x00              | ok
MEM345 VTT PG    | 0x00              | ok
VCCIO PG         | 0x00              | ok
VCORE PG         | 0x00              | ok
FIVR PG          | 0x00              | ok
MEM012 VDDQ PG   | 0x00              | ok
MEM012 VPP PG    | 0x00              | ok
MEM012 VTT PG    | 0x00              | ok
MEM345 VDDQ PG   | 0x00              | ok
MEM345 VPP PG    | 0x00              | ok
MEM345 VTT PG    | 0x00              | ok
VCCIO PG         | 0x00              | ok
VCORE PG         | 0x00              | ok
FIVR PG          | 0x00              | ok
Fan1A            | 5280 RPM          | ok
```

Health Monitoring

- Critical services monitoring:
  - Want to make sure that services that are required for system operation are “running”
  - Examples: srp_daemon/opensmd/crond/postfix
  - Simple script that parses `systemctl status <service>` output

- Multipath health
  - Script that parses `multipath –ll`
  - 2 paths: Healthy,
  - 1 path: warning,
  - 0 paths: critical



Health Monitoring

- Host IB health
  - Network link health (lane count and speed)
  - Check for card->PCI bus link health
  - Check counters changes over time
    - Symbol errors
    - LinkDownedCounter
    - VL15 Dropped

```
[root@f2-oss1a1 10:40:01][lustre]# ibstat
CA 'mlx5_0'
        CA type: MT4115
        Number of ports: 1
        Firmware version: 12.18.1000
        Hardware version: 0
        Node GUID: 0x506b4b03003956be
        System image GUID: 0x506b4b03003956be
        Port 1:
                State: Active
                Physical state: LinkUp
                Rate: 100
                Base lid: 1
                LMC: 0
                SM lid: 1
                Capability mask: 0x2651e84a
                Port GUID: 0x506b4b03003956be
                Link layer: InfiniBand
```

```
[root@f2-oss1a1 10:37:56][lustre]# lspci -s 3b:00.0 -vvv
3b:00.0 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4]
        Subsystem: Mellanox Technologies Device 0014
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx+
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Latency: 0, Cache Line Size: 32 bytes
        Interrupt: pin A routed to IRQ 53
        NUMA node: 0
        Region 0: Memory at ae000000 (64-bit, prefetchable) [size=32M]
        Expansion ROM at ab000000 [disabled] [size=1M]
        Capabilities: [60] Express (v2) Endpoint, MSI 00
                DevCap: MaxPayload 512 bytes, PhantFunc 0, Latency L0s unlimited, L1 unlimited
                        ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset+ SlotPowerLimit 75.000W
                DevCtl: Report errors: Correctable- Non-Fatal+ Fatal+ Unsupported+
                        RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+ FLReset-
                        MaxPayload 256 bytes, MaxReadReq 512 bytes
                DevSta: CorrErr+ UncorrErr- FatalErr- UnsuppReq+ AuxPwr- TransPend-
                LnkCap: Port #0, Speed 8GT/s, Width x16, ASPM not supported, Exit Latency L0s unlimited, L1 unlimited
                        ClockPM- Surprise- LLActRep- BwNot- ASPMOptComp+
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk-
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 8GT/s, Width x16, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
```

```
[root@f2-oss1a1 10:41:16][lustre]# perfquery
# Port counters: Lid 1 port 1 (CapMask: 0x5A00)
PortSelect:......................1
CounterSelect:...................0x0000
SymbolErrorCounter:..............0
LinkErrorRecoveryCounter:........0
LinkDownedCounter:...............0
PortRcvErrors:...................0
PortRcvRemotePhysicalErrors:.....0
PortRcvSwitchRelayErrors:........0
PortXmitDiscards:................0
PortXmitConstraintErrors:........0
PortRcvConstraintErrors:.........0
CounterSelect2:..................0x00
LocalLinkIntegrityErrors:........0
ExcessiveBufferOverrunErrors:....0
VL15Dropped:.....................0
PortXmitData:....................4294967295
PortRcvData:.....................4294967295
PortXmitPkts:....................4294967295
PortRcvPkts:.....................4294967295
PortXmitWait:....................4294967295
[root@f2-oss1a1 10:41:19][lustre]#
```

Health Monitoring

- Switch-to-switch IB health
  - `ibdiagnet` is insufficient for finding all switch-to-switch IB link issues
    - It will help you find unhealthy links
    - Links can go dark and will not be detected
  - Down IB links can cause performance issues
    - If using AR, they can even make the network re-route which causes unavailability for a small time
    - Non-symmetric routes can cause ~3-5% performance drop
  - Script that knows IB network topology and checks for it to be sane
    - Knows that the switch cables are connected to the correct port on the correct switch
      - Impacts network routing
    - Knows that the host cables are connected to the correct port on the correct switch
      - If improperly cabled can impact FGR
    - Check link speed
    - Check link width

Health Monitoring

- Lnet_stats
  - Monitor changes in ` lnetctl stats show` to show LNET congestion or errors
  - Set threshold to report on changes in backlogged messages "msg_alloc"
    - Example 30000
  - Can set a threshold for downed routes, dropped messages, etc…
- Lustre_health
  - Simple script that checks the status of `lctl get_param -n health_check`
  - Tells you if a OST is mounted read-only, is slow, corrupt, etc…
- Ls timer
  - Inside of each cluster network (if routed), check to make sure that you can `ls` inside of a lustre directory within a certain timeout
  - Will tell you if lustre is being slow or not
    - Helps to get in front of users complaining

15-minute break

Open Scalable File Systems, Inc.
3855 SW 153rd Drive Beaverton, OR 97006
Ph: 503-619-0561 | Fax: 503-644-6708
admin@opensfs.org | www.opensfs.org

When Things Go Wrong

- Lustre Recovery
- Gathering debug information
- Network debugging
- Repairing file system issues

Lustre Recovery

- Lustre's recovery mechanism is designed to deal with node/network failures and keep the file system running in a consistent state

- Some of the failures it is designed to handle are:
  1. Client failure
  2. MDT failure
  3. OST failure

- MDS and OSS failures require methods to recover or replay outstanding I/O requests from clients

Client Failure - Detection

- It is important to detect client failure early so that remaining clients can continue accessing the file system
- Two main ways to detect client failure:
  1. Client fails to respond to a blocking lock callback from the Distributed Lock Manager (DLM)
  2. Client fails to “ping” server in a long period of time
- These conditions may occur even if the hardware itself has not actually failed (e.g. – network link failure), but it is still treated the same

Client Failure - Recovery

- When a client failure is detected, Lustre tries to ensure that other clients can continue working
  - Can't afford to have one or more clients waiting to perform I/O while they are trying to acquire a lock held by a dead client
- When a client is evicted:
  - All client locks are invalidated
  - All cached inodes on client are invalidated
  - All cached data on client is flushed
- When client recovers, it may reconnect to the file system and continue operations

MDT Failure - Detection

- Clients may detect MDT failure by timeouts of in-flight requests or from Imperative Recovery
  - Client MDC will attempt to connect to failover node if configured
  - Only clients connected during the failure are permitted to reconnect during the recovery window
- Client state will need to be communicated to MDT once connection is reestablished

MDT Failure - Recovery

- Lustre uses the Metadata Replay protocol to ensure that MDS can re-acquire necessary state information from client transactions that have not been committed to disk
- The protocol uses transaction numbers to ensure operations are replayed in the correct order
- Clients also communicate existing lock state to MDS

OST Failure - Detection

- If an OST fails to respond to a client in a timely manner, the corresponding OSC on the client will treat the OST as having failed
  - Outstanding I/O requests will block until the OST has recovered
  - OSC will try to reconnect to OST through a failover OSS node (if one has been configured)

- Same logic applies if the "client" is the MDS
  - MDS will note that OST is unavailable and skip it when assigning objects to new files

OST Failure - Recovery

- OSC-to-OST recovery protocol is the same as the MDC-to-MDT Metadata Replay protocol
  - Bulk writes usually have been committed to disk so server just needs to reconstruct the reply
  - For other cases, normal replay/resend handling is done
  - Client still has copy of data until it receives acknowledgement
- When OST is in recovery mode, all new client connections are refused until the recovery finishes
  - Recovery finishes when all previously-connected clients have replayed transactions, or a client times out

Metadata Replay Protocol

- Every client request contains a unique, monotonically increasing XID to track order of requests
- Each request processed by server is assigned a unique, increasing Transaction Number (TN)
  - Reply to client's request contains TN for the request along with the last committed TN
- Server maintains last_rcvd file with list of connected clients
- During recovery
  - Request with only XID → resend
  - Request with TN → replay

Viewing Recovery Status

- To view recovery status of all OSTs

```
lctl get_param obdfilter.*.recovery_status
```

- To view recovery status of MDTs

```
lctl get_param mdt.*.recovery_status
```

- Example output:

```
status: COMPLETE
recovery_start: 1553204504
recovery_duration: 0
completed_clients: 1/1
replayed_requests: 0
last_transno: 94574301709
```

Aborting Recovery

- In some cases, it may be known that recovery will not complete properly, or perhaps recovery is not really necessary
  - Previously connected client may currently be down
  - File system was brought down cleanly, but there was an idle client connected at the time
- Recovery can be aborted in two ways:

```
mount -t lustre -o abort_recov <dev> <mnt_point>

lctl --device <dev_num> abort_recovery
```

Gathering Debug Information

- When something goes wrong with Lustre, there are several ways to grab useful information
- Some of these methods are useful for sys admins, and others are primarily of use to developers
- Sources of debug information include:
  - Syslog / dmesg
  - Lustre internal debug logs
  - Crash dumps
  - Debugfs
  - Wireshark

Syslog / dmesg

- Things to look for in log messages:
  - Lustre / LustreError / Lnet / LBUG
  - rc -30 (EROFS)
  - Timeouts / evictions
  - Messages that contain NIDs (10.1.2.3@o2ib, etc.)
- It's impossible to enumerate all the Lustre errors you might see, so let Google be your friend
- Sometimes general pattern of messages can be just as useful (or perhaps more useful) than the content of the messages

Lustre Internal Debug Log

- Lustre maintains an internal circular debug buffer
- A debug mask is used to control what info gets logged
  - Query using "`lctl get_param debug`"
  - Set using "`lctl set_param debug=<mask>`"
  - Can also be set using "`sysctl lnet.debug`"
- Size of the buffer can be modified using

```
lctl set_param debug_mb=<size>
```

- Contents of buffer can be dumped to a file using

```
lctl debug_kernel <filename>
```

- See Lustre manual for info about debug mask options

Crash Dumps

- Use `kdump` (via `kexec`) to capture kernel info when LBUG is encountered
  1. Set kernel to panic on LBUG
     ```
     lctl set_param panic_on_lbug=1
     ```
  2. Install `kexec-tools` package
  3. Add the following parameter to the kernel boot options:
     `crashkernel=<size>` (or “auto”)
  4. Modify `/etc/kdump.conf` if desired
     - For example, send crash dumps over the network to another host
  5. Start the kdump service
     ```
     systemctl start kdump
     ```
- Use a program like `crash` to analyze output

debugfs

- When using ldiskfs for backend Lustre storage, you can inspect the contents of the file system in two ways:
  1. Mount the device with “-t ldiskfs” instead of “-t lustre”
  2. Use the debugfs command

- One benefit of using debugfs is that you can view the contents while Lustre is up and running
  - In that case, it is best to use “`debugfs -c`” so that the device is opened in read-only mode

- Even if there are no problems, spending some time looking at the file layout can provide some insight into how Lustre works

Network Debugging

- Many Lustre issues can ultimately be traced back to network connectivity problems
  - Disruption of client-server communication leads to timeouts or dropped requests
  - Clients see this as a server failure
  - Servers see it as a client failure and evict clients
- Error messages might not make the issue obvious
  - Client syslog message may complain about being unable to process config from MDS, but the real reason is that it can't even contact the MDS
  - Sporadic network problems make debugging even harder

Initial Troubleshooting

- Is the firewall enabled?
- Does every node have the proper NID configured on the correct interface?
- If LNet routing is used, does the client and server have the correct routes?
- Do any nodes have duplicate IP addresses?
- Can you `ping` between nodes? Both ways?
- Can you `lctl ping` between nodes? Both ways?
- Do the servers have MDTs/OSTs mounted?

Infiniband Issues

- Debugging Infiniband issues can get complex, but there are some simple steps that often lead to results
  - Is IPoIB configured?
  - Is the installed version of Lustre built against the correct IB stack (in-kernel vs. MOFED, version, etc.)?
  - Is the IB firmware too old? Too new?
  - Do IB bandwidth tests give expected results?
  - Do IB HBA counters show any errors? What about counters on the IB switch?
  - Does output from ibnetdiscover, ibstat, etc. match what you expect?

Lnet selftest

- Lnet selftest is a useful tool for testing connectivity and measuring network performance
- Can be used to test pairs of nodes or entire clusters
- To run Lnet selftest:
  - Load `lnet_selftest` kernel module on all nodes
  - Use `lst` command to add groups of clients and servers
  - Use `lst` command to specify type of test to run
  - Initiate test from any host on the fabric

Example: Lnet selftest

```
export LST_SESSION=$$
echo LST_SESSION=$LST_SESSION

lst new_session io_test
lst add_group clients 10.10.20.31@o2ib0
lst add_group servers 10.10.1.7@o2ib0 10.10.1.8@o2ib0
lst add_batch bulk
lst add_test --batch bulk --concurrency=8 --distribute 1:2 --from clients \
        --to servers brw write size=1M
lst run bulk
lst stat servers & sleep 30; kill $!
lst end_session
```

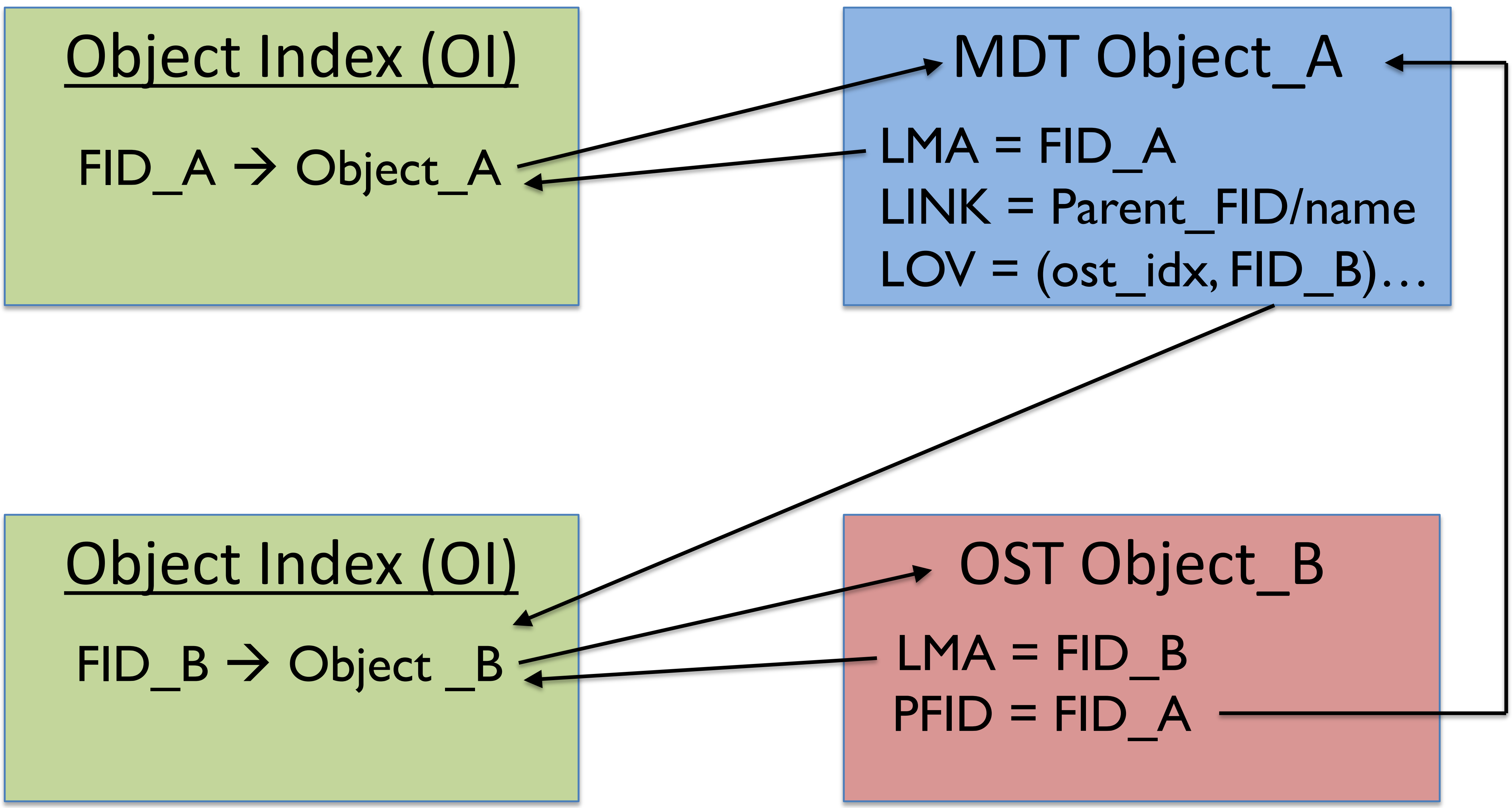
Repairing File System Issues

- Sometimes file system data structures can get into an inconsistent state
- Causes can include:
  - Power failures
  - Hardware failures
  - Software bugs
- Inconsistency could be with Lustre's internal data or with the data structures of the backend ldiskfs/zfs file systems used on the MDTs/OSTs
- Each layer has its own tools to deal with the problem

LFSCK (Lustre File System Checker)

- Lustre provides a tool for checking the consistency of its internal state and repairing any problems
- Prior to Lustre 2.3, performing a full file system check was slooooooow and painful
  - Had to take Lustre offline to generate the needed databases of inode information
  - Best bet was just to run e2fsck on underlying ldiskfs file system and hope it fixed enough of the problems
- LFSCK has been re-engineered to run with the file system online (and in use)

File Identifiers and Objects



LFSCK (Phase 1)

- Maintain consistency of Object Index on MDT
- Iterate through all objects on the OSD
  - Make sure inode number in OI matches with FID from inode's LMA xattr
- Can be triggered manually or automatically
- Maintains checkpoint file (scrub_status) on MDT
  - Allows restart if scan is interrupted
  - Contains stats about current scan
- Supports rate limiting

LFSCK (Phase 1.5)

- Maintain consistency between the FID-in-Dirent info and LMA/LINK xattrs in objects
- Iterate through each object on OSD
- If it is a directory, check each file entry
  - Compare FID listed in dirent with LMA xattr of inode
  - Compare file name from dirent with name from inode’s LINK xattr
  - Compare FID from LINK xattr with FID of parent directory
- Supports checkpoint restart (lfsck_namespace) and rate limiting
- This check is not automatically triggered

LFSCK (Phase 2)

- MDT-OST consistency checking
- MDT object for a file contains list of child OST objects
- Child OST object contains FID for parent MDT object
- Check 4 different cases:
  - Dangling reference – mdt_obj1 points to ost_obj1, but ost_obj1 doesn't exist or doesn't have PFID xattr
  - Mismatched reference – mdt_obj1 points to ost_obj1, but ost_obj1 points to mdt_obj2. mdt_obj2 doesn't exist or recognize ost_obj1 as child.
  - Multiple references – mdt_obj1 and mdt_obj2 both point to ost_obj1
  - Unreferenced object - ost_obj1 points to mdt_obj1, but mdt_obj1 doesn't exist or recognize ost_obj1 as child. No other mdt_obj points to ost_obj1.

LFSCK (Phase 2)

- Fixes ownership inconsistency between MDT and OST objects (MDT ownership takes precedence)
- Will track errors, and if threshold is reached, will trigger full lfsck for file system
- Supports checkpoint restart and rate limiting

LFSCK (Phase 3)

- Implements MDT-MDT consistency check for DNE
- Similar to MDT-OST consistency check in many ways, but also more complicated
- Too many cases to list here
  - Check http://wiki.lustre.org for design docs
- Supports checkpoint restart and rate limiting

Running LFSCK

- Full file system check is initiated via

  `lctl lfsck_start -M ${MDT0} -A -t all -r`

- The `-t` option is used to specify which checks to run
  - scrub – Run OI scrub
  - namespace – FID-in-Dirent, LinkEA consistency
  - layout – MDT-OST object consistency
- Other useful options
  - `-n | --dryrun`
  - `-c | --create_ostobj`
  - `-C | --create_mdtobj`
  - `-o | --orphan`

Example: Running LFSCK

```
[root@haven-mds1 ~]# lctl lfsck_start -M haven-MDT0000 -A -t all -r

[root@haven-mds1 ~]# lctl lfsck_query -M haven-MDT0000
layout_mdts_init: 0
layout_mdts_scanning-phase1: 1
layout_mdts_scanning-phase2: 0
…
layout_osts_scanning-phase1: 30
layout_osts_scanning-phase2: 12
…
namespace_mdts_init: 0
namespace_mdts_scanning-phase1: 1
namespace_mdts_scanning-phase2: 0
```

Repairing ldiskfs corruption

- Since ldiskfs is based on ext4, journaling helps keep the file system in a consistent state
- If a problem occurs that cannot be fixed by the journal, it will be necessary to run `e2fsck`
  - One possible symptom of this is when the logs contain “-30” (EROFS) errors
  - Only need to run `e2fsck` on the device(s) that contain errors
- General procedure:
  1. Replay journal
  2. Run `e2fsck` in non-fixing mode
  3. Run `e2fsck` to fix problems

Example: Running `e2fsck`

NOTE: Always use latest e2fsprogs from Whamcloud
https://downloads.whamcloud.com/public/e2fsprogs/latest/

```
# Unmount affected device
root# umount /mnt/ost

# If possible, use logger to capture output
root# script /tmp/e2fsck.sda

# Replay journal
root# mount -t ldiskfs /dev/sda /mnt/ost
root# umount /mnt/ost
```

Example: Running `e2fsck` (cont.)

```
# Run e2fsck in non-fixing mode
root# e2fsck -fn /dev/sda
…[output]…

# Fix the errors
root# e2fsck -fp /dev/sda
…[output]…
```

- Might need to follow-up with LFSCK if there are lots of problems

ZFS maintenance

- ZFS handles consistency issues differently from ldiskfs
- Admins should periodically scrub zpools
  - Can be done while zpool is online and Lustre is running
  - Causes I/O to disk which could have some affect on the file system
  - Recommended interval = 1 month (?)
- Example:

```
zpool scrub <pool_name>
```

- Can reduce impact from scrub by adjusting sysctl parameter `vfs.zfs.scrub_delay`

Other Useful Admin Info

- Striping Considerations
- OST allocation (Round-robin vs. Weighted)
- Advanced file layouts
  - Progressive File Layout (PFL)
  - Data on MDT (DoM)

Striping Considerations

- Basic file striping is pretty straightforward
  - Most of the time, just choose a stripe count
  - Sometimes you might adjust the stripe size
  - Other options probably used even less
- For user, striping is usually about performance

Knowledge of application IO pattern

↓

Customized striping parameters

↓

Less contention, better IO performance

- But admins have additional concerns…

Default stripe count

- Choosing the default stripe count for a file system can be a tricky proposition
  - Too low → Fill up OSTs with large files
  - Too high → Consume more inodes on OSTs than needed
  - Progressive File Layouts can help with this
- Choice of default stripe count might also affect how you choose to format the MDTs/OSTs
- In any case, it's a good idea to have some general guidelines for users
  - Ex – At least 1 stripe for every 100 GB of file space

Improperly striped files

- Whatever striping guidelines you choose, users still won't listen…
- May need to track down large files with small stripe counts that are filling up OSTs
- Options:
  1. lfs find (could take a while)
  2. Robinhood (if you already have this tool)
  3. OST usage distribution (quick, but limited)

- The last option is handy, but sometimes requires a little work

Searching for Improperly Striped Files

- Look at distribution of OST usage
  - Run “lfs df <filesystem>| sort -nk 5”
  - Look for anomalies at the tail end
- Find the user(s) with the most usage on OST
  - Run “lfs quota -I <ost_idx> -u <user> <filesystem>” command for each user
  - Look for one or more users with abnormally high usage
  - These are your initial candidates for investigation
- Try to locate the offending files

Example

haven-OST000e_UUID x x x 40% /lustre/haven[OST:14]

haven-OST001a_UUID x x x 41% /lustre/haven[OST:26]

haven-OST0017_UUID x x x 41% /lustre/haven[OST:23]

…<snip>…

haven-OST0019_UUID x x x 51% /lustre/haven[OST:25]

haven-OST0005_UUID x x x 52% /lustre/haven[OST:5]

haven-OST0008_UUID x x x 52% /lustre/haven[OST:8]

haven-OST0013_UUID x x x 53% /lustre/haven[OST:19]

haven-OST001b_UUID x x x 65% /lustre/haven[OST:27]

filesystem summary: x x x 46% /lustre/haven

Inode Calculations

- Default stripe count and average file size are important factors for planning a new file system
  - These factors help determine the number of inodes needed on MDTs/OSTs which in turn can affect formatting options and device size requirements
- Number of MDT inodes:
  - num_osts * ost_size / avg_file_size
  - Recommend doubling this to allow for future expansion or smaller than expected file size
- Number of OST inodes:
  - num_mds_inodes * default_stripe_count / num_osts
  - Recommend 2x-4x padding

Inode Calculations (cont.)

- ZFS has variable number of inodes
  - MDT still needs enough space to allow about 4KB per inode
- ldiskfs creates fixed number of inodes during format
- Defaults for Lustre 2.10:
  - inode size = 1KB
  - MDT will have 1 inode for every 2.5KB
  - OST will have 1 inode for every 1MB (if OST size > 8TB)
- Can alter inode ratios by adding option to mkfs.lustre:

```
--makefsoptions="-i <bytes-per-inode>"
```

- Adjust this option to get desired number of inodes

Inode Disparity

- Primary goal of these calculations is to have parity among MDT and OST inode counts
  - Ideally, inode and space usage track each other
- Disparity can show up in non-obvious ways

```
# lfs df -i /lfs01
UUID       Inodes      IUsed      Ifree       IUse% Mounted on
MDT0000   2402287616  46560885   2355726731    2%   /share/lfs01[MDT:0]
OST0001   24117248    22883788   1233460      95%  /share/lfs01[OST:1]
OST0003   24117248    22903308   1213940      95%  /share/lfs01[OST:3]
OST0004   24117248    22895442   1221806      95%  /share/lfs01[OST:4]
OST0006   24117248    22890201   1227047      95%  /share/lfs01[OST:6]

summary:    51457138   46560885    4896253     90%   /share/lfs01
```

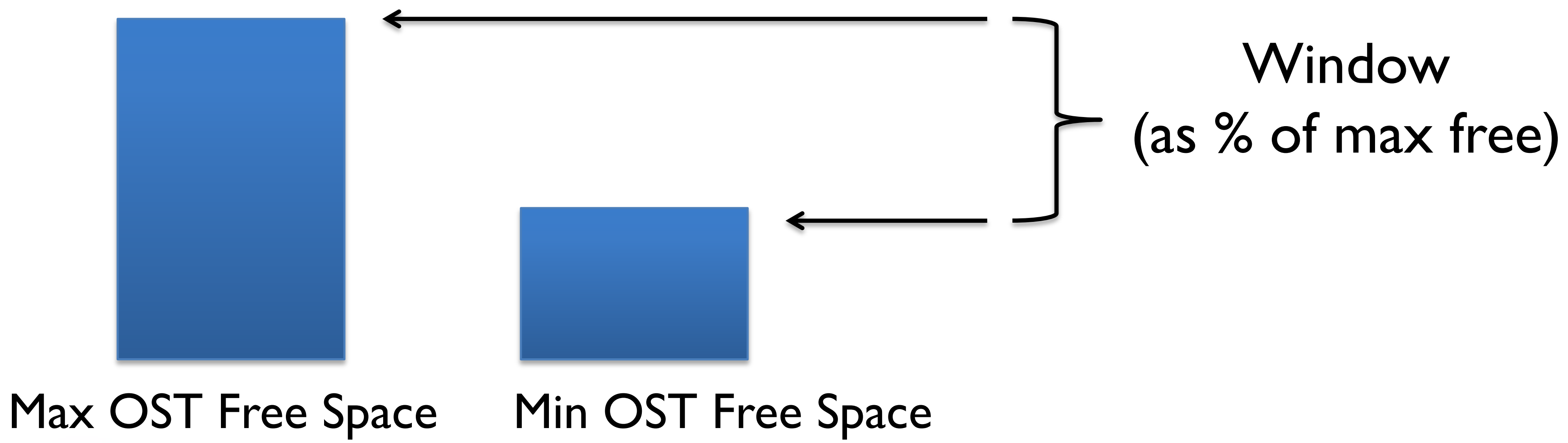
OST Object Allocation

- When a new file is created, Lustre allocates objects on OSTs according to desired stripe count

```
-bash-4.2$ lfs getstripe testfile
testfile
lmm_stripe_count:   4
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  8
        obdid       objid               objid               group
          8       231338244           0xdc9f104               0
         39        20273590     0x13559b6     0
         30        20441490     0x137e992     0
         38        20549867     0x13990eb     0
```

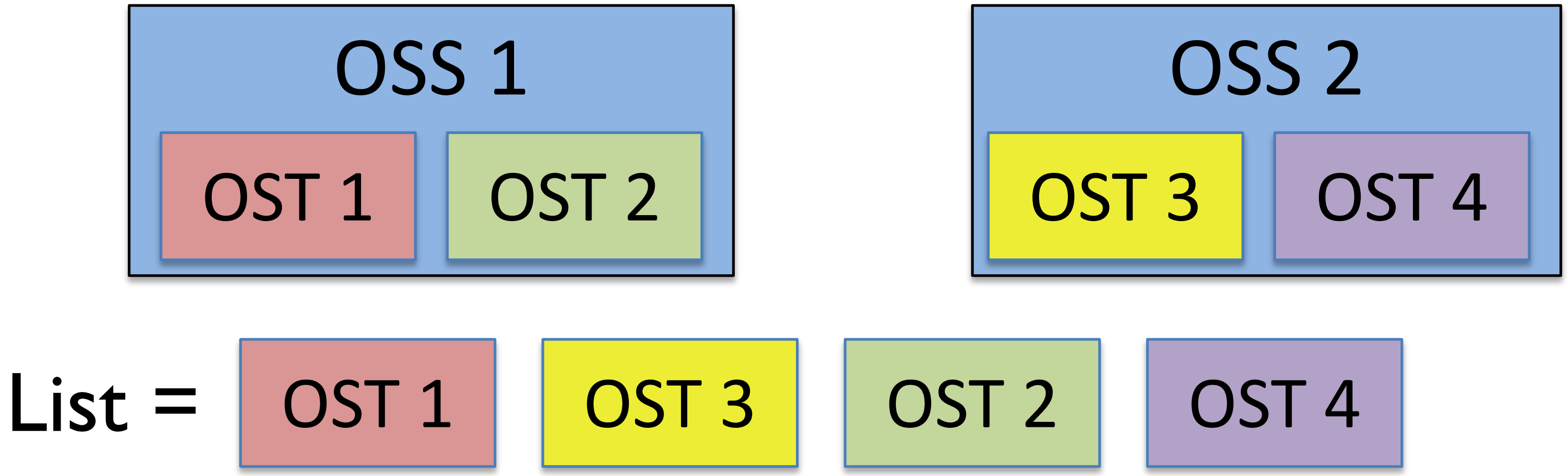
OST Allocators

- How does Lustre decide which OSTs to assign to a file?
- Two different allocators
  - Round-robin
  - Weighted
- Choice based on how "balanced" usage is (as defined by the target window)



Round-Robin Allocator

- Round-robin allocator is used if the OST usage is balanced (i.e. – all OST free space falls within target window)
- OSTs are assigned sequentially from an internal list
- List is not necessarily sequential with regards to OST index
  - Accounts for things like OSTs being on different nodes



Weighted Allocator

- Weighted allocator is used when OST usage is not balanced
- OSTs are assigned a weight based on the amount of free space and their location
  - Emptier OSTs have a higher weight and are more likely to be selected
- Algorithm makes random selection based on weights
  - Even OSTs with the least free space still have some chance of being selected
- The goal is to divert more I/O to OSTs with the most free space while still utilizing other OSTs to some extent

Adjusting Allocator

- Admins have some control over which allocator is used and how the Weighted allocator assigns weights
- Control size of window used to determine if OST usage is balanced
  - /proc/fs/lustre/lov/<name>-MDT0000-mdtlov/qos_threshold_rr
  - Default value is 17%
  - If set to 100%, round-robin is always used
- Control how much weight is affected by free space
  - /proc/fs/lustre/lov/<name>-MDT0000-mdtlov/qos_prio_free
  - Default value is 91%
  - If set to 100%, weights are based solely on free space

Advanced File Layouts

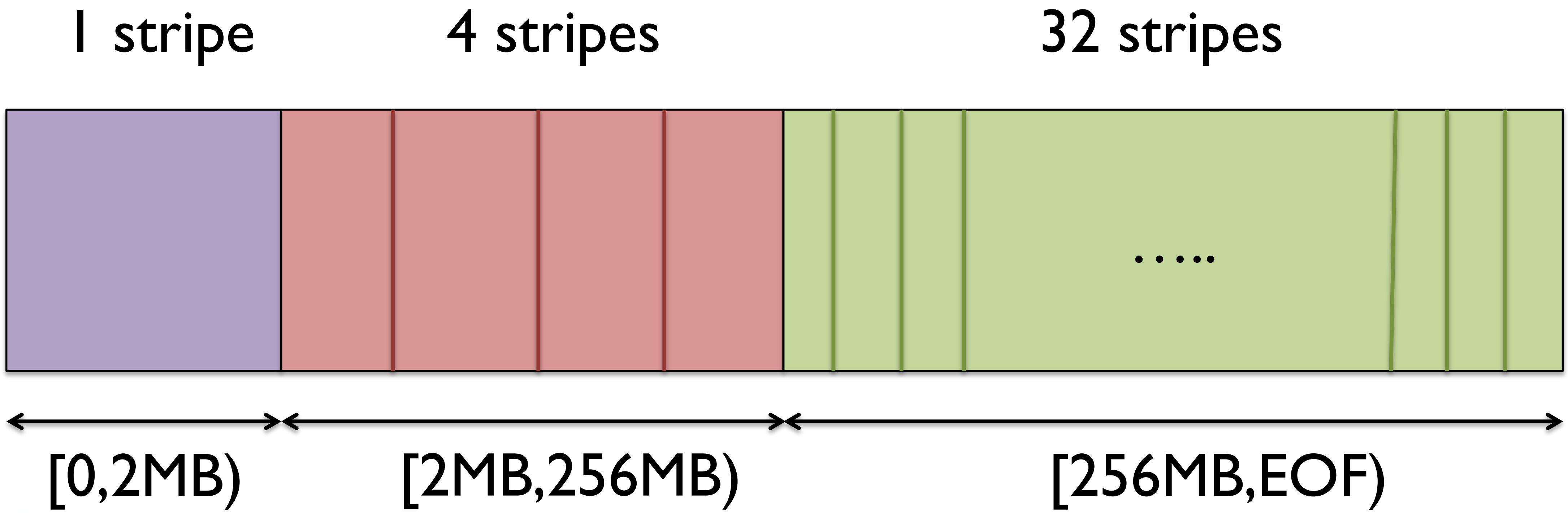
- Recent versions of Lustre have added some features that provide more options beyond current basic layout

1. Progressive File Layout
   - Provides ability to adjust file layout as the size of the file grows.
   - Essentially creates different basic layouts for different sections of a file

2. Data on MDT
   - Store some (or possibly all) file contents on the MDT itself

Progressive File Layout (PFL)

- Introduced in Lustre 2.10
- A PFL file is essentially an array of basic layouts (components) that cover different non-overlapping sections of a file



PFL Benefits

- Fine-grain control of layout could provide performance improvements
- File layout can be adapted on-the-fly
  - Only need to define initial component
  - Add components when needed
  - Don’t use more OST inodes than necessary
- Choose a default PFL for all users that gradually increases stripe count as the file size increases
  - No more full OSTs! (maybe…)
- Underlying composite layout structure forms basis for other layout options

PFL Examples

Create PFL for previous figure

lfs setstripe -E 2M -c 1 -E 256M -c 4 -E -1 -c 32 <file>

Create starting layout, then add component

lfs setstripe -E 2M -c 1 -E 256M -c 4 <file>

lfs setstripe --component-add -E -1 -c 32 <file>

Display all components of file

lfs getstripe <file>

NOTE: Will only see OST objects for instantiated components

Data on MDT (DoM)

- Introduced in Lustre 2.11
- Designed to improve file I/O by placing small files (or the first part of a larger file) directly on MDT
  - Helps eliminate extra RPCs to OSTs
  - Advantageous if MDT storage is faster than OST storage
- This is a special case of PFL in which the first component has a single stripe that resides on the MDT
- Example:

```
lfs setstripe -E 1M -L mdt -E 256M -c 4 -E EOF -c 10 <file>
```

DoM Settings

- Some care must be taken when allowing users to place data directly on the MDT
- Admins can limit the size of the file's first stripe that resides on the MDT
- Controlled via dom_stripesize parameter (default=1MB, disabled=0):

```
# Query value
lctl get_param lod.*MDT0000*.dom_stripesize
# Set value temporarily
lctl set_param lod.*MDT0000*.dom_stripesize=<value>
# Set value permanently
lctl conf_param <fsname>-MDT0000.lod.dom_stripesize=<value>
```

Panel Session

Questions?

Open Scalable File Systems, Inc.
3855 SW 153rd Drive Beaverton, OR 97006
Ph: 503-619-0561 | Fax: 503-644-6708
admin@opensfs.org | www.opensfs.org