Instrumenting IOR to Diagnose Performance Issues on Lustre File Systems

Doug Petesch
Performance Team
Cray Inc.
St. Paul, MN, USA
dpetesch@cray.com

Mark Swan
Performance Team
Cray, Inc.
St. Paul, MN, USA
mswan@cray.com

***Abstract*—Large Lustre file systems are made of thousands of individual components all of which have to perform nominally to deliver the designed I/O bandwidth. Many disk drives and switch ports and any redundant components may operate in a degraded state at times to provide resiliency. When the measured performance of a file system does not meet expectations, it is important to identify the slow pieces of such a complex infrastructure quickly. This paper will describe how Cray has instrumented IOR (a popular I/O benchmark program) to automatically generate pictures that show the relative performance of the many OSTs, servers, LNET routers and other components involved. The plots have been used to diagnose many unique problems with Lustre installations and help understand how unavoidable variations in component level performance affects file system level I/O performance.**

Keywords-IOR; Lustre; I/O

I. INTRODUCTION

A large external Lustre file system is a collection of thousands of semi-independant pieces of hardware, software and firmware. There are many opportunities for mistakes to happen when connecting all the parts and some of those components will fail or degrade during the installation/test phase. There are component level tests to check out the parts, but quite often issues don't appear until full scale is reached. Ultimately, customer acceptance depends on demonstrating the contracted performance with an application level benchmark. From experience over the last couple years, we have found that a properly instrumented user level application can quickly checkout components and also provide insight into full scale performance issues.

II. LUSTRE BASICS

Fig. 1 shows the basic components of an external Lustre file system attached to a Cray XE or Cray XC30 mainframe in sufficient detail for this paper. Any component not running at designed speed can keep the whole file system from meeting its I/O benchmark performance target.

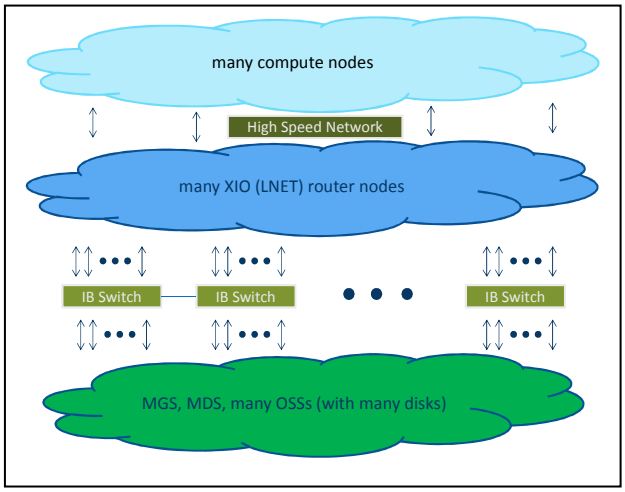

Figure 1. External Lustre file system components

All data travels across several network interfaces between compute nodes and disk storage. Each layer has resiliency features, but few components are fully redundant so there are many possible degraded states at these levels:

- Luster servers are connected to an Infiniband (IB) network
- Compute nodes are connected to the Cray High Speed Network (HSN)
- LNET router nodes provide a gateway between compute nodes on the Cray HSN and the Lustre servers on the IB network
- Each Lustre Object Storage Server (OSS) controls some number of disk arrays called Object Storage Targets (OSTs)

III. APPLICATION VS FILE SYSTEM PERFORMANCE

Measuring the I/O rate seems fairly simple at first. Take the amount of data moved divided by the elapsed time to get the rate.

$$\text{I/O RATE} = \frac{\text{DATA}}{\text{TIME}}$$

User applications usually have a certain amount of data to read or write so they naturally measure how long it takes to move the fixed amount of data. However, large Lustre file systems have thousands of moving parts (literally!) so there is always a period of time between when the first and last stream of I/O completes where not all of the file system hardware is being used. From a file system perspective, it makes more sense to keep all the parts busy moving data for a certain period and then measure how much data was moved in that fixed amount of time.

Since the "fixed data" rate is determined by the slowest component and always includes time when part, or most, of the file system is idle, it will always be slower than the sustained bandwidth of the file system measured over a "fixed time". Therefore, even if the customer specifies a fixed data measurement technique for acceptance, we often use fixed time measurements while debugging and tuning since that gives us a better idea if the file system is approaching its theoretical bandwidth.

IV. IOR Basics

IOR[1] is a commonly used benchmark program. It has over 50 options to allow it to produce a wide variety of I/O workloads. IOR is an MPI program that can scale well, and is easy to build. Many customers specify that IOR be used to measure the I/O performance for acceptance. For these reasons, it is important to get as much information out of IOR as possible.

Since we're most interested in verifying that newly installed file systems behave as expected, we'll concentrate on Posix file-per-process (FPP) I/O. Other more complicated I/O workloads (shared files, MPI-IO, HDF5, etc.) introduce more variables. In a basic FPP run, each rank of the IOR program opens a file, writes, closes, reopens, reads and closes the file again. Each rank times each step.

By default, each MPI rank will write and read the same amount of data to simulate a "fixed data" application. IOR can also measure the file system performance over a "fixed time" by use of the deadlineForStonewalling (-D) option. After the specified number of seconds, each rank stops issuing new I/O requests. The program then waits for any outstanding I/O requests to complete and totals up the amount of data moved by all the ranks.

V. IOR Fixed Data Example

This IOR command will write 4 GiB of data to each of 100 files with direct I/O using a transfer size of 4 MiB.

```
aprun –n 100 IOR -a POSIX -C -B -F -t 4m -b 4g –k
```

When run with 100 ranks it produces this concise output:

```
Summary:
api                = POSIX
test filename      = testdir/IOR_POSIX
access             = file-per-process
pattern            = segmented (1 segment)
ordering in a file = sequential offsets
ordering inter file=constant task offsets=1
clients            = 100 (4 per node)
repetitions        = 1
xfersize           = 4 MiB
blocksize          = 4 GiB
aggregate filesize = 400 GiB

Max Write: 6015.63 MiB/sec (6307.84 MB/sec)
Max Read:  3046.21 MiB/sec (3194.19 MB/sec)
```

The Lustre file system was based on 3 NetApp E5400 controllers. There were 6 OSSs with 3 OSTs each. We expected higher rates, so added the verbose=3 (-vvv) option to get per MPI task (file) timings like:

```
Task=0, Time=1365558598.489247, write open start
Task=0, Time=1365558598.489978, write open stop
Task=0, Time=1365558598.496538, write start
Task=0, Time=1365558641.157996, write stop
Task=0, Time=1365558666.575858, write close start
Task=0, Time=1365558666.576329, write close stop
Task=0, Time=1365558666.597461, read open start
Task=0, Time=1365558666.597855, read open stop
Task=0, Time=1365558666.599108, read start
Task=0, Time=1365558754.811135, read stop
Task=0, Time=1365558801.056288, read close start
Task=0, Time=1365558801.056823, read close stop
```

Additional scripts were developed to automatically extract the time to write and read each file and generate pictures with gnuplot to quickly display much more detail of the run.

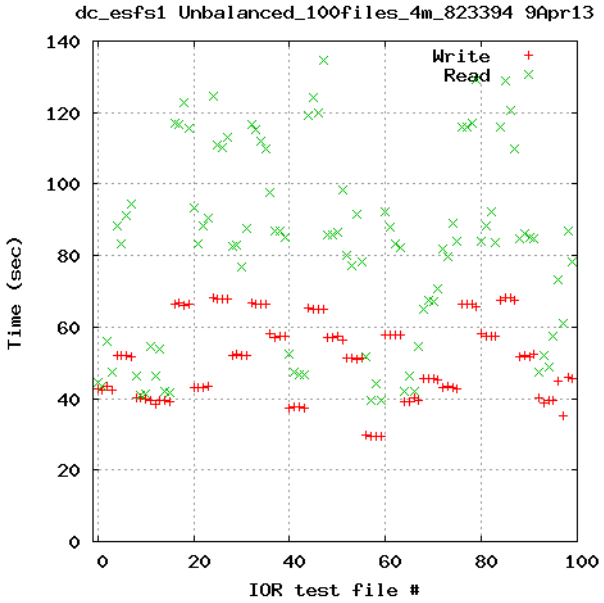

Figure 2. Time to write and read individual files

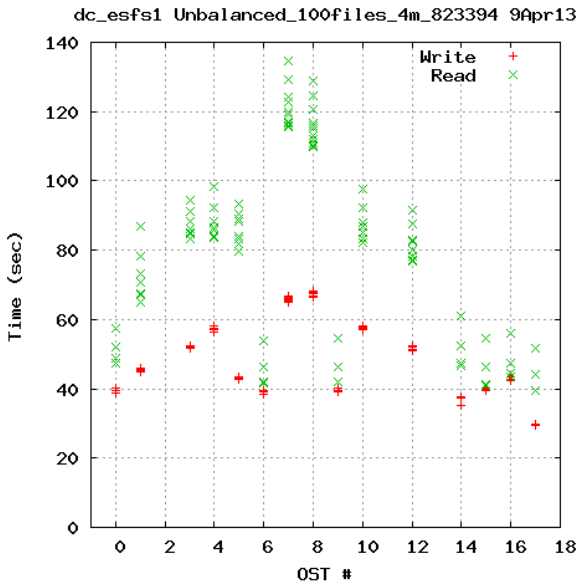

Figure 3. Write and read times sorted by OST index

Fig. 2 shows there is a wide variation in the time it takes to write and read the various files. Next we use the "lfs getstripe -i" command to find the OST index of each file.

Fig. 3 groups the times for all files on each of the 18 OSTs. The OSTs that took the longest (7 and 8) each had 12 files. OSTs 3, 4, 5, 10 and 12 each had 8 files. Several OSTs (2, 11, 13) did not have any files and the rest had 4, 5 or 7 files. This distribution of files on OSTs is unusually poor, but any imbalance hurts fixed data IOR rates.

Fig. 4 shows the times from a later run after forcing Lustre round robin file allocation with qos_threshold_rr =100.

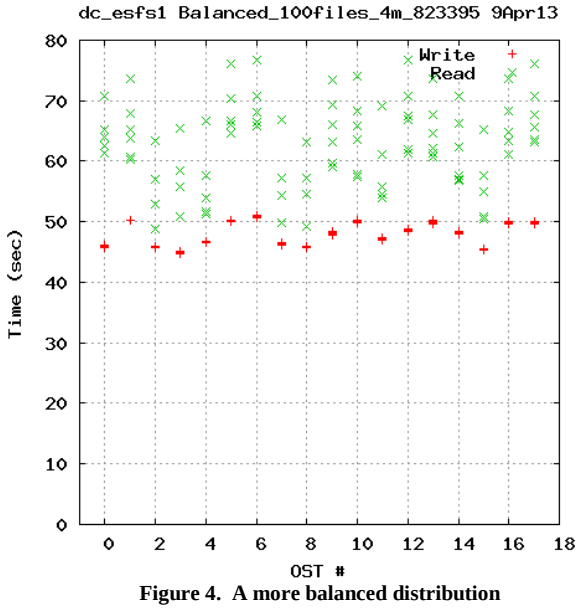

Figure 4. A more balanced distribution

Since the maximum write and read times were reduced, the average rates reported by IOR for moving the same fixed amount of data increased.

```
Max Write: 8029.20 MiB/sec (8419.23 MB/sec)
Max Read:  5335.16 MiB/sec (5594.32 MB/sec)
```

The number of files is still not a multiple of the number of OSTs so some OSTs had 5 files and some had 6. To get the best fixed data IOR performance it is critical to keep every I/O path as balanced as possible to spread the workload evenly over all components.

VI. Disk Position

Even after ensuring files are evenly distributed on OSTs and running in a dedicated environment there can be variation in performance among the OSTs. A common source of variability is the physical position of the files on the disk drives. Typical server disks rotate at a constant rate. The data is stored at a constant linear density so the tracks at the outer edge hold more data than those near the inner edge.

Most OSTs are RAID6 (8+2) disk arrays. Each 1MiB block of data is striped over the disks in the array. The physical addresses of the data blocks in the OST start at the outer (fastest) edge of the disks with the highest address at the inner most (slowest) zone. We use the linux filefrag command to associate a physical address with each IOR file.

Fig. 5 is the timing information from an IOR run using 1152 files on a file system with 144 OSTs. Similar to the previous plots, but now time is on the horizontal axis. The 8 (3 GiB) files written to each OST take up 24 GiB which is much less than 1% of the available space on the nearly 16 TB OSTs.

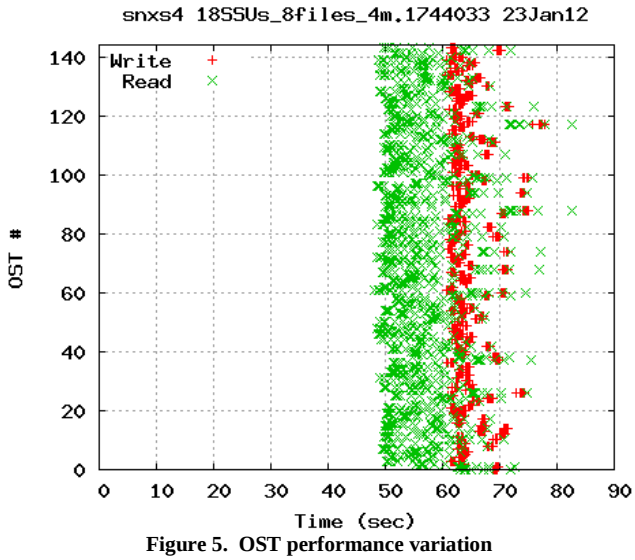

Figure 5. OST performance variation

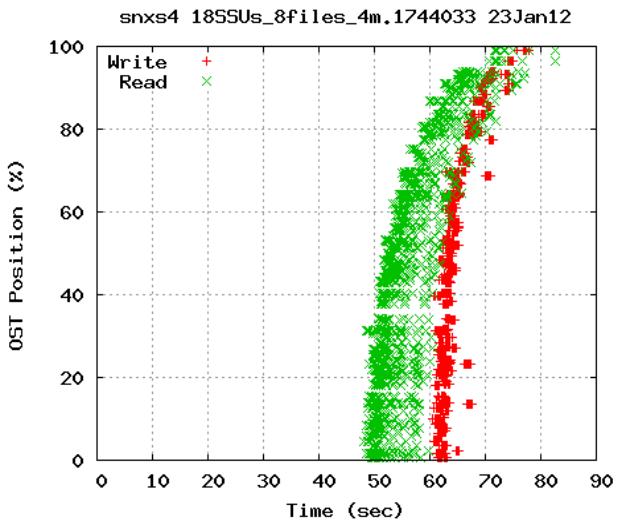

Figure 6. OST performance relative to disk position

Fig. 6 contains the same timing information as Fig. 5, but the files are sorted by their position within their respective OSTs rather than by their OST index. The 8 files written to a particular OST are interleaved and tightly packed, but when viewed across all OSTs, the allocation position of the groups of files appears random, even though the file system was essentially empty. This is typical of production Lustre file systems.

VII. I/O Rates Over Time

Since it takes more disk rotations to transfer 24 GiB of data near the inner edge than at the outer edge, it makes sense that it would take more time too. The IOR timing information confirms that, but it is not clear yet if the rates are constant over time.

The next step was to instrument IOR to record its activity over time. At the end of the run, each rank prints out how much data it moved every 0.25 seconds.

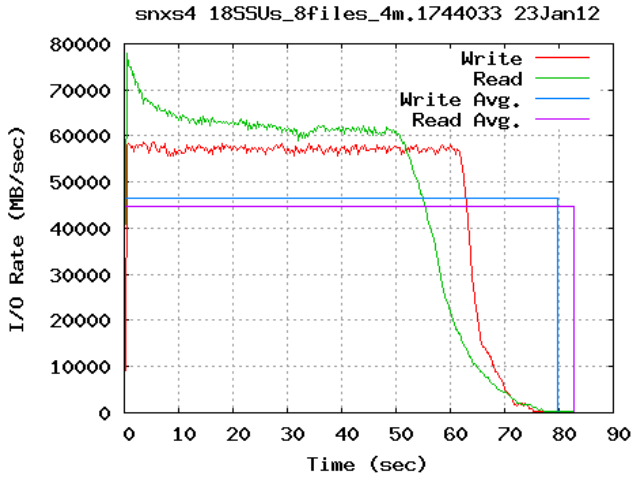

Figure 7. I/O rates over time

Fig. 7 shows the aggregate I/O rates for the 1152 files over the time it takes to write and read the files. The two phases do not actually overlap. The reading starts after the writing ends, but both results are plotted together for comparison.

The file system was a CRAY Sonexion 1300 Data Storage System. It had 18 Scalable Storage Units (SSUs). Each SSU has 2 OSSs and a total of 8 OSTs. The expected sustained I/O rate is 3 GB/sec per SSU. Fig. 7 shows that the write and read rates are both greater than 54 GB/sec while the full file system is in use. As the fastest files complete, the aggregate bandwidth drops until the last (slowest) file finishes.

Based on the final finish time, IOR reports:

```
Max Write:44386.53 MiB/sec (46542.65 MB/sec)
Max Read: 42745.48 MiB/sec (44821.89 MB/sec)
```

The average write and read rate lines in Fig. 7 represent what the rate profile would look like if all OSTs performed at the average rate for a fixed time equal to the actual times required to write (79.73 sec) and read (82.79 sec) the fixed amount of data (3456 GiB) in the actual run. The difference in the sustained rate before any file finishes and the average rate of the fixed data job due to disk position variability is often on the order of 15 to 30%.

Plotting the aggregate I/O rate as seen from an application over a long period of time has helped identify transient issues such as the Lustre ping effect[2] that were missed when only looking at the average rate. Separating the rates over time for individual OSTs or OSSs has been useful to identify failing disk drives or intermittent rate drops due to error recovery from poorly seated IB cables. The basic idea of instrumenting I/O calls in applications is quite old[3].

VIII. Finding Degraded IB Links

Resiliency is critical for the robustness of a large Lustre file system. However, it can cause headaches for the benchmarker. Many components will automatically drop to a lower speed if they encounter too many errors at their nominal speed. The degradation may be logged somewhere, but that information might not be available to a normal user.

A common example is IB links. There are many possible rated speeds (FDR, QDR, DDR, SDR). For a variety of reasons, such as a cable connection vibrating loose or a firmware mismatch between the devices on both ends, one or more of the links in a complex IB fabric may be running at a speed lower than expected. The system will still run correctly, and many jobs could still run at expected performance, but a full file system benchmark would be affected. The performance impact of a degraded IB link may affect writes and reads differently due to the presence of LNET routers.

Most Cray systems with external Lustre file systems use some form of Fine Grained Routing (FGR) where access to a small set of OSSs is through a small set of routers. The recommended FGR ratio for connecting a CRAY Sonexion 1600 Data Storage System to a CRAY XE6 mainframe is 4 routers to 3 OSSs. That means there are IB links from 4 XIO nodes and from 3 servers going to the same switch.

When writing, the Lustre client on a compute node round robins the data across the 4 routers which each forward the data to the OSS controlling the destination OST. When reading, the OSS round robins the requested data across the 4 routers on its way to the compute node. A slow link between any of the LNET routers and the switch will affect both reads and writes for any of the OSTs controlled by the 3 OSSs. A slow link between an OSS and the switch will only affect reads from OSTs owned by that OSS, but would affect writes to all the OSTs on all 3 OSSs. All 4 routers will be slowed down trying to get data through the slow link impeding their ability to send data to the OSSs also.

Fig. 8 shows this affect. This individual file timing information came from an IOR run on a 48 cabinet Cray XE6 system with a 14 SSU Sonexion 1600 file system with 7 SSUs in each of 2 cabinets. Each cabinet has four 4:3 FGR groups for the first 6 SSUS and two 2:1 groups for the last SSU. At the time of this run, the link between the first odd numbered OSS and the top of rack (TOR) switch was degraded to SDR speed. All other links were at QDR speed.

The IOR job used 896 ranks on 224 nodes to write and read 8 (3 GiB) files on each of the 112 OSTs. Each OSS controls 4 OSTs so there is 96 GiB of data sent to and from each OSS. The reads to the OSS with the slow link took 105 seconds which equates to 96*(1.024)**3/105 = 0.98 GB/sec which is close to the 1 GB/sec SDR speed. The writes to all 3 OSSs in that FGR group are also limited to that speed.

Fig. 8 also shows how job placement within the large Cray XE Gemini 3D torus network can affect bandwidth to individual FGR groups. This job is trying to use all the bandwidth of the file system from less than 5% of the compute nodes in the system. The default placement clumps those nodes to maximize HSN communication bandwidth between the nodes of the job. Since the LNET router nodes are scattered around the torus, the available number of paths (and therefore bandwidth) between the group of compute nodes and the router groups varies.

If the same job was assigned a different set of compute nodes or a smaller or larger number of nodes, the available bandwidth to individual routers could change. Reads are affected more than writes since when reading, all the data from an FGR group of servers enter the torus at a few points so the diversity of communication paths is limited.

IX. Failed LNET Router

Having 4 routers in an FGR group gives some redundancy. If one fails the clients will round robin their transfers across the remaining 3. The Cray XE router nodes have a bandwidth of about 2.6 GB/sec while the OSSs of a Sonexion 1600 can write up to about 3 GB/sec. The normal 4:3 ratio has some router bandwidth to spare. If one router is down, however, the bandwidth of the FGR group may be limited by the routers instead of the OSSs.

Fig. 9 shows the write times for an IOR job that wrote 12 (3 GiB) files to each of the first 48 OSTs of a Cray Sonexion 1600 file system. The OSSs that finished in ~58 seconds wrote at about 2.7 GB/sec, but the FGR group including the

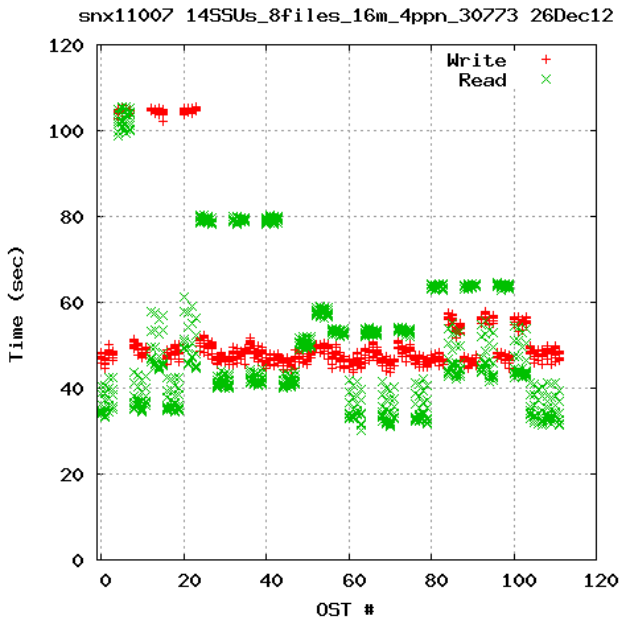

Figure 8. Degraded IB link

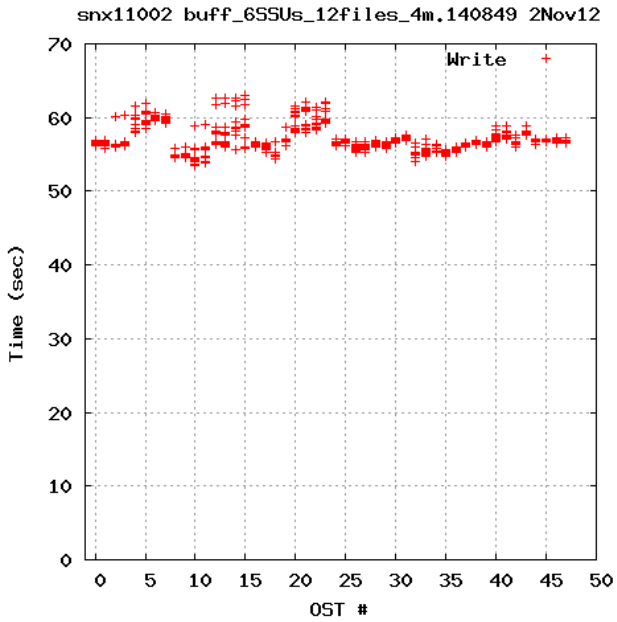

Figure 9. Write performance with a failed LNET router

failed router and the first 3 odd numbered OSSs (OSTs 4-7, 12-15, 20-23) was a little slower (~2.5 GB/sec/OSS) due to the bandwidth limit of the 3 remaining routers.

X. Failed Over Server

Occasionally a Lustre server will crash and its OSTs will fail over to its partner server. Since the other server now has twice as many OSTs to manage, the performance of both groups of OSTs will be reduced until the failed server can be returned to service.

Plotting the time to write each of the files during a fixed data IOR job on a file system with a failed over server would show the files on OSTs involved in the fail over would take roughly twice as long. Since the files on any particular OST always have a range of completion times, it is hard to assign a rate to an OST or OSS based just on those times. Now that we have instrumented IOR to record the rate of each file over time, we can calculate the average I/O rate for each OST or OSS based on the elapsed time up to when the first file finishes.

Fig. 10 shows the rates of each of the OSSs on a 180 SSU (360 OSS) Cray Sonexion 1600 file system. At the time of the run, the servers for OSSs 199 and 214 were down so OSSs 198 and 215 were each controlling 8 OSTs instead of the usual 4. The script that extracts the performance information from the IOR output simply aggregates the rates for an OSS from a sequential group of 4 OSTs. Therefore the plot shows both OSSs in each pair as slow when in reality one server in each pair is not doing anything.

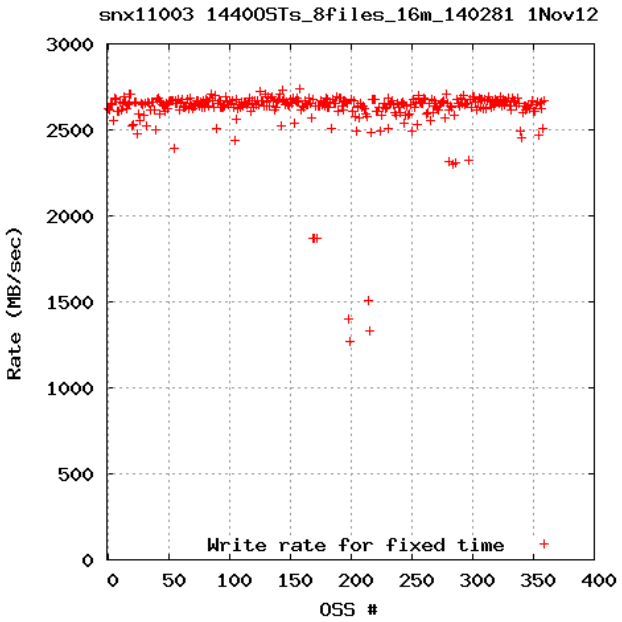

Figure 10. Server write rates while failed over

There was also one FGR group of servers (OSSs 168, 170 and 172) that were performing poorly that day, but most of the OSSs were writing at the expected rate of about 2.7 GB/sec or 5.4 GB/sec per SSU.

XI. Summary

Instrumenting IOR to provide rate information as a function of time makes it a very useful tool for diagnosing I/O performance problems on large Lustre file systems. It has uncovered software scaling issues like the Lustre ping effect and transient issues that turned out to be firmware defects or poor cable connections. Seeing how the I/O performance varies over time gives much more insight into the health of a file system than usual single average rate output of IOR.

Using IOR as a file system test and debugging tool has some advantages over component level test tools like dd or obdfilter-survey or LNET self test in that IOR:

- Scales from a single thread to thousands of nodes
- Can generate a wide variety of I/O patterns
- Is often specified as the official measurement method for acceptance
- Can be run by unprivileged users

A single IOR job can gather performance information on all the OSTs and OSSs, or just a subset, under conditions that can be made to match almost any application workload. Automatically turning the expanded information into graphs with gnuplot makes it easier to quickly identify which parts of the file system are operating nominally and which components are not.

Besides being a valuable debugging tool during the installation of a complex Lustre file system, the IOR program can also help identify the impact that inevitable component failures will have on user applications. Finally, it can be useful to explore the potential benefit of changing the I/O pattern of an application through its ability to simulate so many types of I/O.

References

[1] The source for the IOR benchmark application, version 2.10.3, is available at “http://sourceforge.net/projects/ior-sio”.

[2] C. Spitz, N. Henke, D. Petesch, J. Glenski, “Minimizing Lustre ping effects at scale on Cray systems,” Proc. Cray User Group, 2012

[3] D. Petesch, J. Bauer, “An application independent intelligent I/O layer,” Proc. Cray User Group, Spring 1993