Finally, A Way to Measure Frontend I/O Performance

Christopher Zimmer, Saurabh Gupta, Verónica G. Vergara Larrea
Oak Ridge Leadership Computing Facility
National Center for Computational Sciences
Oak Ridge, Tennessee
{*zimmercj,guptas1,vergaravg*}*@ornl.gov*

***Abstract*—Identifying sources of variability in the Spider 2 file system on Titan is challenging because it spans multiple networks with layers of hardware performing various functions to fulfill the needs of the parallel file system. Several efforts have targeted file system monitoring but only focused on metric logging associated with the storage side of the file system. In this work, we enhance that view by designing and deploying a low-impact network congestion monitoring system designed especially for the I/O routers that are deployed on service nodes within the Titan Cray XK7 Gemini network. To the best of our knowledge, this is is the first tool that provides a capability of live monitoring for performance bottlenecks at the I/O router level. Our studies show high correlation between I/O router congestion and I/O bandwidth. Ultimately, we plan on using this tool for I/O hotspot identification within Titan and guided scheduling for large I/O.**

I. INTRODUCTION

Performance monitoring within supercomputers is challenging. Supercomputers differ from traditional data-center and cloud computing systems in that the applications tend to be highly coupled, latency sensitive, and resource intensive. As a result, we often see supercomputers utilizing low-latency configurations across the system stack. These include tickless operating system (OS) scheduling, low-latency interconnects, OS-bypass functionality, among others. Significant effort has been put towards reducing noise in these environments. For this reason, it should be no surprise that high performance computing (HPC) system administrators tend to be wary of adding fine-grained system-monitoring tools to compute environments. Fine-grained monitoring tends to require sampling of the system state during periodic timing intervals. While generally the overhead of such sampling is low, it can add jitter to a system in which great effort has been put in to eliminate jitter.

The Oak Ridge Leadership Computing (OLCF) facility is home to Titan, the second fastest supercomputer in the world [1]. Titan is a Cray XK7 [2] system with 18,688 compute nodes, each with a 16-core 2.2 GHz AMD Interlagos processor and an NVIDIA Kepler K20X GPU accelerator. Spider 2 [3], [4] is a center-wide Lustre parallel file system at the OLCF is which has 32 PB of capacity and provides a 1 TB/s bandwidth [4].

Facilitation of the Spider 2 file system requires the connection of the Gemini torus network [5] within Titan to the storage I/O InfiniBand network (SION 2), a center-wide resource. This fact coupled with the complexity of the backend storage creates a system where performance variability is frequent and difficult to pinpoint. Unfortunately, from a system design standpoint, an understanding of system bottlenecks can be difficult to impossible without constant data-streams of performance information showing the machine under various workloads.

To date, there have been several projects at the OLCF that aim to improve visibility into the backend file system performance, and also to assist in identifying degradation. Many of the tools developed are currently deployed on the storage system backend and collect statistics from the different file system components. The DDNtool [6] utility, for example, continuously polls the DDN controllers and collects performance and fault information, such as the number of failed drives, and progress of rebuilds, and stores all the information in a database. Similarly, `brw_stats` is another tool that is used to monitor the backend systems. The `brw_stats` tool which provides I/O statistics from object storage targets (OSTs) to help identify I/O patterns that negatively impact the file system *e.g. small, random I/O*.

While the information collected from these tools can help identify performance degradation stemming from the backend, they do not capture information on bottlenecks that could originate from within Titan. To help address this gap, the OLCF developed the I/O Router Congestion Daemon (IORCD) which was designed specifically for the I/O routers within Titan's Gemini network, and is currently deployed on Titan. Our preliminary results using the OLCF's I/O test harness [7] show that introducing IORCD produces a negligible impact on runtime and I/O performance. Although Gemini congestion data from IORCD alone is not sufficient to definitively identify system wide sources of performance degradation, it has already been useful to gain a better understanding of the impact of Gemini network congestion on I/O performance. The IORCD tool, as well as the results obtained from using the tool in production to monitor I/O router congestion on Titan, can further extend I/O performance monitoring capabilities and can help gain a better understanding of I/O performance under production workloads.

II. BACKGROUND

A. Spider 2

Spider 2 is the primary file system resource available to users at the OLCF. As shown in Figure 1, Spider 2 in combination with the storage I/O network (SION 2) and the OLCF's compute and analysis resources form the OLCF end-to-end I/O path. Although Spider 2 was designed to primarily support Titan, it is a center-wide resource and, as such, other resources at the OLCF including Eos, a 736-node Cray XC30 system, and Rhea, a 512-node RHEL cluster, rely on Spider 2 for storage.

Spider 2 [3] is a Lustre-based parallel file system with 32 PB of capacity and provides an aggregate bandwidth of 1 TB/s. Spider 2 is partitioned into two file systems, *atlas1* and *atlas2*, each one with access to 1,008 object storage targets (OSTs). This design allows applications that use single-shared files to take advantage of wide-striping to achieve better I/O performance.

Troubleshooting performance bottlenecks on a parallel file system of the scale of Spider 2 can be challenging. In addition to having a complex architecture, Spider 2 also relies on several system software components. Both changes in Spider 2's system software stack or hardware can potentially impact I/O performance. In order to minimize negative impacts on I/O performance, the OLCF regularly tests new versions of system software (e.g. Lustre server/client software, InfiniBand firmware) using a combination of kernels, I/O benchmarks, and real-world applications [7]. However, due to the limited amount of time that can be reserved for testing, and the different behavior observed at-scale in comparison to that observed on test systems, performance regressions do occur in production. Given that performance degradation can originate from any part of the OLCF end-to-end I/O system, monitoring tools throughout the end-to-end I/O path are necessary.

B. Frontend I/O

Titan uses the Gemini 3D torus network, built upon tightly coupled custom ASICS. The network contains 9,600 Gemini routers where routing decisions are dimension ordered in the absence of faults. Titan is composed of a total of 19,200 nodes in the 3D torus, and 18,688 of those are compute nodes. Many of the remaining nodes are service nodes that are used for various purposes. One such purpose is to serve as login nodes, used for deploying applications and various other activities. Other service nodes are used for I/O forwarding. These nodes, called I/O routers, perform the action of routing traffic between the Gemini and InfiniBand networks, and are less sensitive to latency.

Due to the striping nature of Spider II, the OLCFs Lustre parallel file system, when data is written from a compute node to Spider II, it is packetized and sent through several I/O routers. This data is picked up by the recipient I/O router, where a checksum is performed and then it is forwarded onto the InfiniBand network. The most expensive operation performed is the data checksum. In our own analysis, we have observed the work of the file system routing client to be mostly memory bound under heavy load.

C. I/O Performance Bottlenecks

As sections II-A and II-B emphasize, there are significant differences in the hardware capabilities of the storage backend versus that available on Titan's I/O routers. These differences result in an egress path with support for up to 56 OSTs, each capable of up to approximately 400 GB/s, and a lower ingress bandwidth at the I/O router interface. Furthermore, due to the high number of Titan compute nodes that map to each I/O router, in combination with the additional load the I/O router must endure in order to perform checksums, the likelihood that over-subscription at the I/O router interface can occur is high.

Standard monitoring tools like `DDNtool` and `brw_stats` rely on information obtained from the backend systems and can assist system administrators in detecting issues impacting object storage servers (OSSs), metadata servers (MDS), object storage targets (OSTs), and metadata targets (MDTs). However, these kind of tools do not capture information about performance bottlenecks that can occur at different sections of the end-to-end I/O path, including those occurring at Titan's I/O routers. In order to detect I/O performance bottlenecks that stem from congestion between Spider 2 and Titan's Gemini network, additional logging is needed at the I/O router level. To monitor potential over-subscription of I/O router nodes, as well as assist system administrators in identifying performance bottlenecks, the IORCD tool was deployed on Titan's I/O router nodes.

III. I/O ROUTER CONGESTION DAEMON

The I/O Router Congestion Daemon (IORCD) is a tool developed to give insight into I/O performance bottlenecks that may exist within the Titan's Gemini network. As discussed in the background of this work, all prior tools on Titan have focused on the backend storage controllers and InfiniBand fabric associated with the Spider II file system. Unfortunately, such a limited view could result in missed opportunities to identify hot spots such as build-up at I/O Routers. For the first-phase development of IORCD, we decided to employ simple Gemini monitoring but ultimately plan on extending this further. While this would result in a lack of fidelity, we wanted to ensure that the first pass of the tool would have negligible overhead impacts when deployed on our production systems.

In order to make the I/O Router Congestion Daemon collect data, we integrated the sampling with the Gemini-Performance Counter Daemon (GPCD). GPCD effectively employs a kernel module that enables the sampling of both

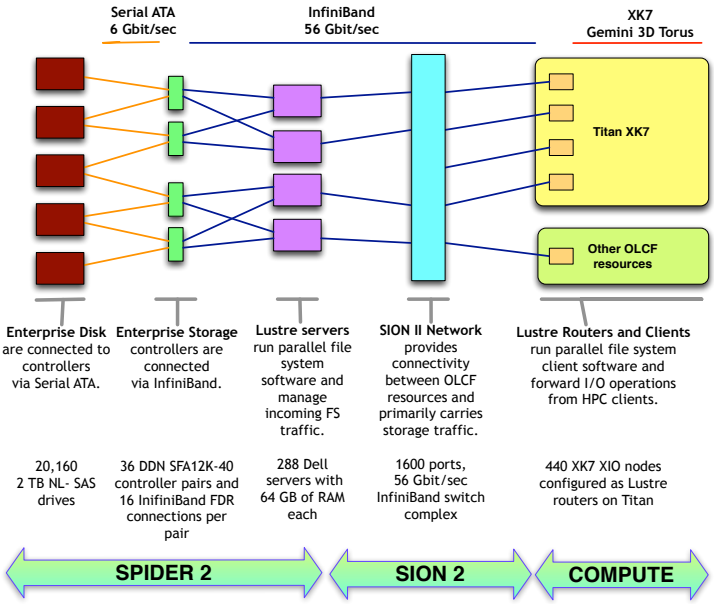

Figure 1: OLCF Spider 2 end-to-end I/O path.

Gemini Router and Gemini network interface card (NIC) performance counters through an IOCTL to the kernel. This mechanism allows us to get back a per-tile set of counters related to network performance. Each Gemini router is composed of 48 tiles, and each tile provides 6 performance counters for the link associated with the tile. For our effort, we were mostly interested in the tile which corresponded to host-side transfers. On Gemini routers there are eight links associated with two hosts, with messaging overhead we see roughly 5.8 GB/s of bandwidth to each host. These links are used by IORCD to establish a measure of congestion of traffic flowing into and out of the I/O Router.

Each Gemini Router tile provides six fixed performance counters (MMRS) [8], [9] associated with their bound link. These performance counters measure request/response phits/packets and congestion. Congestion is represented as INQ and OUTQ congestion. INQ congestion manifests during tile internal routing. In the case of the host-link this counter would increase when transferring data from the host-tiles to the directional-tiles that will send the packets off of the Gemini Router. On the opposite side OUTQ congestion manifests when there is a backup transferring off router. In the case of host-links this would be when transferring data from the Gemini router into the host-NICs of the service nodes. Ideally, INQ congestion should manifest during file-read operations and OUTQ congestion should manifest from file-write perspectives. Figure 2 shows an evaluation of these performance counters taken under controlled experiments on the Chester test system. In this example we use IOR from 8 processes to generate an 8GB file, after termination of the first phase a waiting period is employed and then we use IOR to read back the file. During this time we sample performance counters using a 30 second interval. The results show the congestion impacts across the 4 I/O routers used for these transfers. The first phase of write shows the OUTQ congestion, between the Gemini Router and Service node, becoming congested. In the worst case the congestion cycles increase to roughly 210,000,000 cycles. This indicates a good traffic flow but is by no means a high-measure of congestion. During the read phase the INQ counters on the same routers experience congestion. In all steps both congestion counters are being measured and shown. There are several takeaways from this example, the main one being that the INQ and OUTQ counters can respectively be used to identify read and write I/O. Since no significant MPI processes are running on these nodes, the bulk of traffic can be assumed to be I/O related, and the counters are fine grained to be able to measure even small I/O actions.

Through our previous experiment we were able to show that the congestion counters can provide a directionality of the I/O flow being processed. However, it is also important to be able to establish a reasonable relationship between the magnitude of the flow and the measurement of congestion. With such a large sampling rate as 30 seconds, we are limited in the granularity of data that we can obtain. For our first effort, we decided to focus on detecting high levels of congestion which would result in several seconds of full congestion during a 30 second sampling window. Essentially, at heavy uses of the I/O system, this will help us to better characterize the relationship between congestion

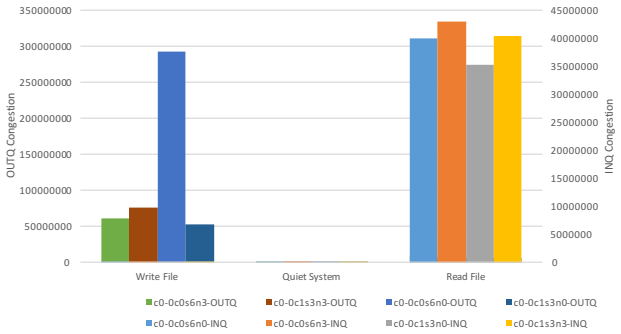

Figure 2: Per IO Router Congestion Counters by I/O Type

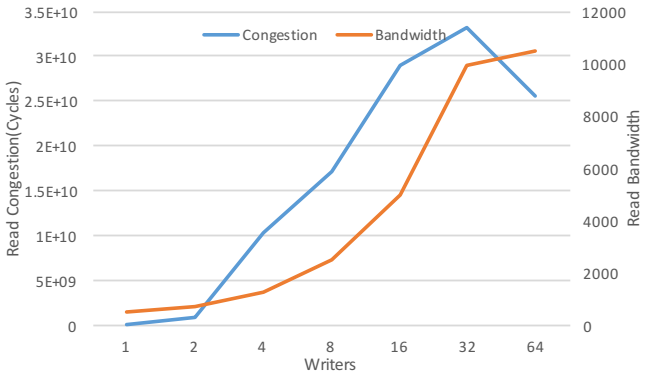

Figure 3: Read Congestion vs IOR Bandwidth

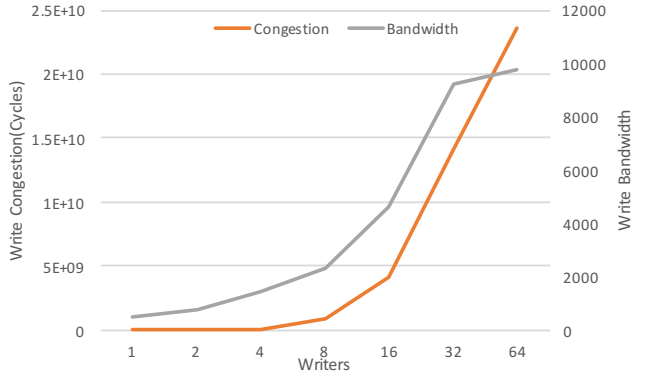

Figure 4: Write Congestion vs IOR Bandwidth

and I/O bandwidth.

For this experiment we used IOR to read and write files. Our IOR input parameters were selected to generate I/O from the application to fill the 30 second sampling window. Then, we extracted the average bandwidth that was achieved during this period to show the relationship between congestion and bandwidth. To fully achieve peak bandwidth it was necessary to scale the number of participating compute nodes and the number of ranks per node. Too many ranks per node would generate a bottleneck at the compute node. For these experiments, we scaled up to 64 participating ranks limiting 4 ranks per node. Figures 3 and 4 show these experiments for reads and writes. In the case of writes, the bandwidth and congestion trends correlate well, with a Pearson's coefficient of 0.94. This indicates that for higher levels of congestion we can infer an I/O router reaching a capacity where additional I/O added to a router would likely result in diminished performance. The correlation with reads however was lower, a 0.85 Pearson's coefficient. Additionally, we observed diminished congestion values between 32 and 64 ranks. This was due to credit starvation between the clients and the compute endpoints that resulted in data throttling.

Based on these results we view write congestion as being a more accurate indicator of I/O router workload. This makes sense when reasoning about network flows into and out of the I/O router. Traffic exiting the I/O router can be blocked along the Gemini path resulting in varying congestion levels based on end-points, as reads are a fan-out pattern from I/O routers to compute nodes. However, write traffic follows a fan-in pattern, where the congestion measurement is taken at the end-point. Any slow down in the backend or on the I/O router will result in increased congestion at the I/O router. Thus, any overflow at the compute nodes is blocked at ingress prior to reaching the I/O router. This makes measuring the flow/congestion at the point of the I/O router more representative of the flow moving through the I/O router. This is important from the perspective of detecting over saturated resources and ultimately will be instrumental in designing mitigation techniques.

One of the major reasons we did not deploy IORCD globally on all compute nodes is the sampling cost of GPCD. Sampling requires an IOCTL call through the GPCD kernel daemon, which in the past has shown to require up to 600ns to return. In our own profile, we measured results around 250ns dependent upon the data we were collecting. By reducing our samples to 8 tiles associated with the HH link and only two performance counters from each, we are able to make these calls with minimal overhead. In our preliminary version of IORCD we are using a sampling period of 30 seconds. We rely on the global NTP service utilized across all OLCF systems to keep the daemons within reasonable synchronization.

IORCD and the Lustre LNET client co-exist within a service node. Working locally on our test system, we evaluated the Lustre LNET client under load. While multi-threaded, the Lustre LNET client appeared to be memory bound with CPU loads between 50-70% for a heavily loaded client. This indicated that IORCD needed a small memory footprint but that the CPU cost of data collection should fit without perceptible overhead. We evaluated this during our first pass deployment discussed in the next section.

Ultimately, we plan to decrease this sampling period but as a first pass of the production viable daemon we wanted to ensure negligible overhead in order to add additional data collection functionality.

Performance data after collection on the individual routers is collated into the Cray System Management Workstation (SMW) via the `syslog` utility. From the SMW, individual log entries are forwarded into our Splunk infrastructure. Splunk is a log collection and analytics framework. The Splunk infrastructure enables visualizations of time-series data, automatic field extraction, and the ability to generate alerts based on pattern based analysis. In our first pass implementation, the log entry data contains the internal hostname, a timestamp epoch, write congestion values, and read congestion values. Time granularity from `syslog` was insufficient and could be delayed, so we included the timestamp from the time the sampling occurred in IORCD. This improves analytics by eliminating any gap in the indexed record between the sample time and the time the SMW received the message.

IV. Deployment

For our initial deployment we evaluated IORCD on our debug cluster, Chester. Chester like Titan, is a Cray XK7, containing 96, 80 of which are compute nodes. Chester contains 8 LNET routers for I/O. In our first evaluation, we wanted to make sure that the I/O performance from an application's point of view was not negatively impacted. To perform this evaluation, we ran experiments from our I/O test harness. The three benchmarks we use are configured with different I/O patterns that impact the file system in different ways. HACC-IO reads and writes single shared files, whereas GTC has a single terminal write phase during which each rank creates and writes independent files in an N:N write strategy. Finally, we use the IOR benchmark, using a file-per-process, to stress the I/O Routers under heavy write traffic. From the graph in Figure 6, we can see the performance ratio of runtimes of these applications with and without IORCD running. As expected, the results demonstrate the negligible impact of monitoring using IORCD. Variance between measurements is consistent with and without IORCD monitoring. This is exactly as expected and will allow us to build in further monitoring and increase sampling rate.

A. Performance Diagnostic

Shortly after the deployment of IORCD to Chester, we got our first opportunity to analyze performance anomalies in the I/O system from the perspective of IORCD. During some experimental runs of the I/O test harness used to characterize the performance overhead of an I/O profiling tool, we ran into a two hour period during which some of our evaluations of the GTC application experienced up to 600% performance degradations in their runtimes. From

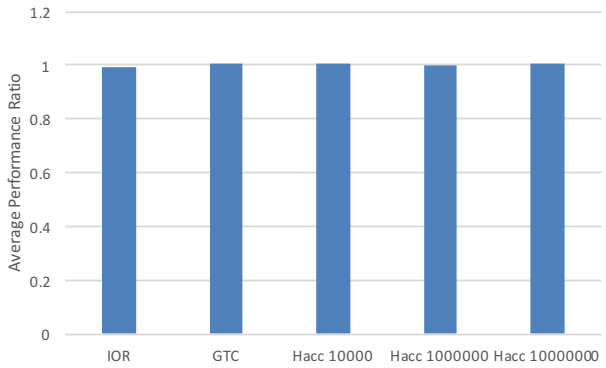

Figure 6: I/O Router Congestion Daemon: Performance Impact Study

investigation of the experimental outputs, we identified that the jobs were experiencing significantly increased write times. In IORCD, this manifested as shown in Figure 7. Over several runs of GTC the I/O system was able to handle the burst of I/O from the compute nodes in a small time frame. Healthy runs spike quickly with significant magnitude and the I/O write phase ends within a single sampling period of the daemon. In unhealthy phases, shown around 2:27:38, the congestion duration extends through approximately 5 samples with low congestion during the entire duration. In all of the degraded runs, the problem manifested as reduced congestion with long duration in more than two samples. More than two samples is an important aspect here because an application run of GTC could have simply startled two sampling periods without manifesting irregular performance. Extending congestion beyond two sampling periods strongly indicates that the duration of the I/O was being impeded. It is also important to point out the low congestion values. Lustre LNET uses a credit exchange mechanism to handle network data exchanges. If a component within the parallel file system became bottlenecked due to under-performance or the manifestation of errors, interfaces would become throttled due to lack of credits, resulting in low-congestion over an extended period of time, as was shown in IORCD. Ultimately, an analysis of OSS logs showed a locking issue in the system shortly after 3:00 PM that was resolved automatically. The performance degradations also stopped shortly before 3:00 PM. This example was important for two reasons, it demonstrated a new way for us to start investigating degradations in the file system performance and also demonstrated to us a new set of information that we could use from the I/O routers in Titan to refine our analysis of congestion. This will be discussed in the next section.

IORCD was deployed to Titan in February of 2016. Due to the binding of two service nodes per Gemini router, IORCD was deployed to half of the service nodes in the system making sure to attain coverage for each Gemini router. The

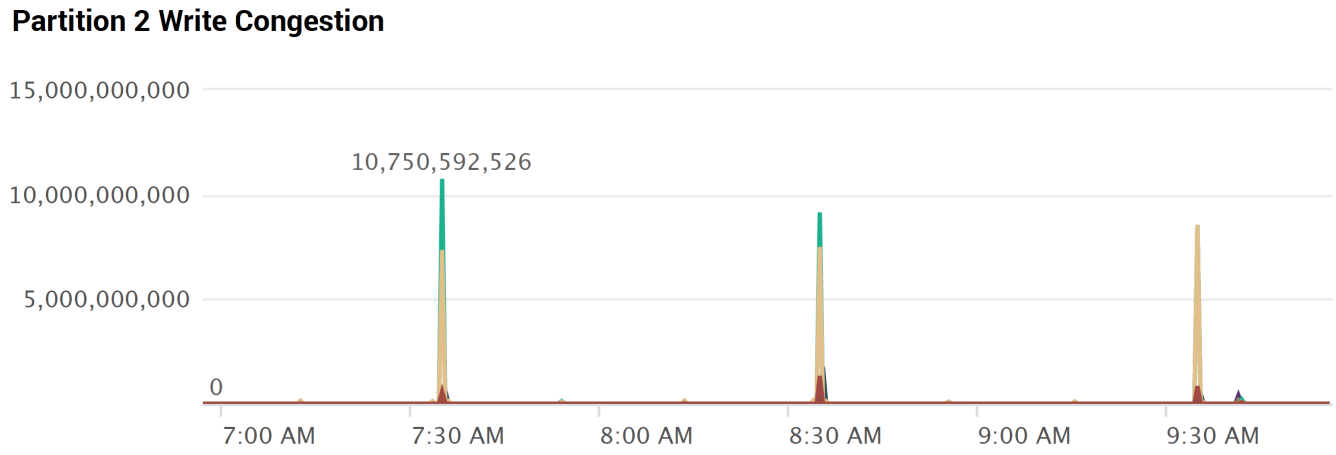

Figure 5: Splunk Dashboard IORCD Interface

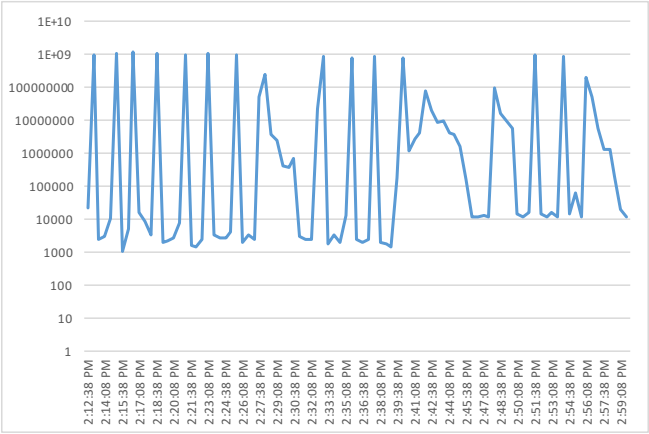

Figure 7: January 21, 2016: GTC apruns including several highly degraded runs

deployment created a significant increase in the amount of data being pushed into the SMW, and ultimately Splunk, but each of the systems responded well and were able to handle the new volume of data. To logically organize the data streams from 220 I/O routers, we leveraged the organization inherently built into Titan through Fine-grained routing. Fine-grained routing creates a static organization of routes from compute nodes to I/O routers to top-of-rack switches in the file system backend. This mapping reduces hop counts in the Gemini network and also congestion in the backend network. This also serves as a logical organization in which to break up I/O routers into differing groups represented as 12 I/O partitions on Titan. Figure 5 is a screen taken directly from the Splunk UI of Partition 2 on Titan. Partition 2 contains 36 I/O routers, representing 36 LNETs. This provides access to all of the OSSs within Spider II, and nodes within each LNET's geometric partition on Titan will communicate through this set of I/O routers to reach the parallel file system.

V. Future Work

In this work, we have discussed our first implementation of a new mechanism for monitoring parallel file system performance on Titan. Up to this point we have focused our efforts on mining the Gemini performance counters as a means of inferring I/O performance at the routing layer. Due to the granularity of sampling and the limitations of the performance counters, the fidelity of this data only provides loose estimations of our view into operations. In this section, we will discuss additional work that is currently being undertaken to improve upon our ultimate goal of identifying Gemini side impacts to I/O performance.

A. LNET Credit Integration

In the previous evaluation of congestion vs. bandwidth, we demonstrated that increasing the write bandwidth results in an increase in service node ingress congestion. While this inference works as a reasonable analogue for assessing workload on I/O routers, we found that additional information would be necessary to increase the accuracy of bottleneck detection. This weakness of inference became apparent when debugging the GTC performance degradations on Chester. With the knowledge that applications were experiencing significant performance degradation, we were able to pinpoint the data from the IORCD logs. However, in different circumstances, and on a busier system that same pattern could simply result from several small files being written over an extended period of time.

The difference between the two situations lays within the credit exchange method that is used for LNET routing in Lustre. In the first case of data throttling, a common OST was under-performing. This resulted in a credit depletion between the I/O router and the OST, due to work being enqueued for transfer to the OST faster than it could be

```
lctl get_param -n nis
nid                      status alive refs peer  rtr   max    tx   min
0@lo                         up     0    2    0    0     0     0     0
16798@gni101                 up     0  839   16    0  4096  4096  3734
16798@gni102                 up     0  750   16    0  4096  4096  3865
16798@gni103                 up     0  748   16    0  4096  4096  4015
10.36.229.1@o2ib201          up     0    4   63    0  1280  1280  1091

lctl get_param -n peers
nid                      refs state  last   max   rtr   min    tx   min queue
7616@gni102                 1    NA    -1    16    64    64    16    15 0
15431@gni101                1    NA    -1    16    64  -156    16   -40 0
6197@gni103                 1    NA    -1    16    64    64    16    15 0
13724@gni101                1    NA    -1    16    64  -219    16   -30 0
10.36.225.100@o2ib201       1    up    69    63    64  -126    63 -27802 0
...
```

Figure 8: Lustre LNET Client Stat Commands

stored. This created back-pressure that extended to the credits used for transfers between the compute-hosts and the I/O router. This, in turn, forced the compute hosts to transfer at a lower rate, and block at ingress. This resulted in low congestion at the Gemini interface due to credit exhaustion at the LNET layer. In the inverse case of many small transfers being made over an extended period of time to a healthy OST, there would be remaining credits and low Gemini congestion. From this example, it became clear that there would be benefit to integrating LNET credits into IORCD.

The Lustre LNET client provides several ways of measuring congestion at the higher level I/O routing layer. Lustre LNET routing is accomplished using credit exchanges to limit overloading. Credits are employed at the network interface level. Information about the LNET counters can be found in the Lustre Operations Manual [10]. The network interface level is queried as shown in Figure 8. These counters show the particular network interface and a series of credit counters. Of interest to us are the `max` and `tx` credits. In this case, `max` is the combined number of send credits for this interface and `tx` is the number of send credits currently available for this interface. Therefore $MAX - TX$ is the number of active sends. Using these counters, we could track active sends over successive sampling periods, where large values of active sends or increasing values over measured time periods may be indicative of problems. Combining this data with the Gemini congestion may provide better insight into overall routing performance and aid in identification of bottlenecks at the routers.

Credits are also applied at the peer level, shown as the second command in Figure 8. These counters reflect peer exchange credits and are used to limit a single peer from monopolizing resources. As with the NIS counters, the peer counters reflect the peer credit availability. Gemini side peers are denoted by `nid@gni` and OSS peers are reflected by `IP@o2ib`. From these counters, we would likely be interested in the `RTR` and `TX` counters. The `RTR` counter represents the number of routing buffer credits that are currently available for use by the peer. The `TX` counter, as on the NIS metrics, represents the number of send credits available to the peer. Credit integration into IORCD would enable the tracking of individual peers, and, as mentioned before, the value of these counters can enable an understanding of the instantaneous in-flight messages for a single host. This is represented by $RTR - TX$, additionally, the ratio of $(RTR/TX) > MAX$ where MAX is the number of concurrent sends from this peer, indicates that operations are blocking.

The next steps of integrating Lustre credit congestion data into IORCD enables several options for improving the view into I/O performance from Titan. The NIS credits would potentially require a higher sampling interval but the backups in these credits have a directionality associated with them that when coupled with Gemini interface congestion, enable us to determine if the backup were on the Gemini or the InfiniBand side. The value here is the additional cost to include this in IORCD should be relatively low as there are very few counters that would need to be considered.

A stronger potential outcome of this would be the integration of peer credits. Tracking individual peers representing Lustre OSSs and compute nodes in Titan would enable us to accurately track individual specific backups. On the OSS, this could aid us in detecting under-performing OSTs. Using NIS credits for this would show backup on the InfiniBand side, but would not enable us to identify the specific under-performing OST. Another possible use of this data is in detecting performance degradation through geometric digression. Geometric digression is an artifact of blocking communication on the Gemini torus. Essentially, when multiple compute nodes communicate with the same

I/O router, if the path from their respective nodes to the I/O router overlaps, then bandwidth will be reduced. Without tracking flows, it is extremely difficult to detect geometric digression. However, using peer credits, it would be possible to identify GNI peers that engaged in adversarial traffic flows. Since the GNI peers themselves have static locations in the network as does the I/O router. This information, coupled with the common XYZ dimension order routing that is most commonly used, can enable us to infer if there is potential overlap in the traffic pattern. This information would be valuable to libraries such as ADIOS or LibFGR in phasing I/O timing to attempt to mitigate these bandwidth reductions. Techniques like these are already used inside applications to improve I/O performance, but they are unable to make inferences about traffic patterns from other applications in adjacent locations. This data would be application agnostic and thus provide a deeper insight into the network traffic.

VI. Conclusion

In this work, we have presented a new tool for identifying congested I/O within the Gemini network on OLCF's Titan. The I/O Router Congestion daemon is the first tool, to our knowledge, to run on the service nodes with the Gemini network, where all other efforts have been focused on the backend of the parallel file system. Using this tool we can identify and detect I/O routers that are over-provisioned causing reduced performance in I/O bandwidth that is invisible to the backend of the storage system. We also outlined our plans to extend the data collection of IORCD to integrate more detailed knowledge of Lustre congestion to improve the understanding of I/O performance at this level. We believe this effort will enable us to detect non-ideal write patterns from multiple compute hosts and develop mitigation strategies to reduce the impact of geometric digression on network performance.

References

[1] J. Dongarra, H. Meuer, and E. Strohmaier, "Top500 supercomputing sites," http://www.top500.org, 2015.

[2] "Cray XK7." [Online]. Available: http://www.cray.com/products/computing/xk-series

[3] D. A. Dillow, D. Fuller, R. Gunasekaran, Y. Kim, H. S. Oral, D. M. Reitz, J. A. Simmons, F. Wang, G. M. Shipman, and J. J. Hill, "A Next-Generation Parallel File System Environment for the OLCF," Jan. 2012.

[4] S. Oral, D. A. Dillow, D. Fuller, J. Hill, D. Leverman, S. S. Vazhkudai, F. Wang, Y. Kim, J. Rogers, J. Simmons *et al.*, "Olcfs 1 tb/s, next-generation lustre file system," in *Proceedings of Cray User Group Conference (CUG 2013)*, 2013.

[5] R. Alverson, D. Roweth, and L. Kaplan, "The Gemini System Interconnect," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, Aug. 2010, pp. 83–87.

[6] Ross Miller, Jason Hill, David A. Dillow, Raghul Gunasekaran, Galen M. Shipman, and Don Maxwell, "Monitoring Tools for Large Scale Systems," in *In Proceedings of Cray User Group Meeting*, 2010.

[7] Veronica G. Vergara Larrea, Sarp Oral, Dustin Leverman, Hai Ah Nam, Feiyi Wang, and James Simmons, "A More Realistic Way of Stressing the End-to-end I/O System," in *In Proceedings of Cray User Group Meeting*, 2015.

[8] K. T. Pedretti, C. T. Vaughan, R. F. Barrett, K. Devine, and K. S. Hemmert, "Using the Gemini Performance Counters," Napa Valley, May 2013.

[9] "Using the Cray Hardware Counters." [Online]. Available: http://docs.cray.com/books/S-0025-10//S-0025-10.pdf

[10] "Lustre* Software Release 2.x: Operations Manual." [Online]. Available: http://doc.lustre.org/lustre_manual.pdf