LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-461793

Comparison of leading parallel NAS file systems on commodity hardware

R. Hedges, K. Fitzgerald, M. Gary, D. M. Stearman

November 8, 2010

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Comparison of Leading Parallel NAS File Systems on Commodity Hardware

Richard Hedges, Keith Fitzgerald, Mark Gary, D. Marc Stearman

Lawrence Livermore National Laboratory

***Abstract*—High performance computing has experienced tremendous gains in system performance over the past 20 years. Unfortunately other system capabilities, such as file I/O, have not grown commensurately. In this activity, we present the results of our tests of two leading file systems (GPFS and Lustre) on the same physical hardware. This hardware is the standard commodity storage solution in use at LLNL and, while much smaller in size, is intended to enable us to learn about differences between the two systems in terms of performance, ease of use and resilience. This work represents the first hardware consistent study of the two leading file systems that the authors are aware of.**

***Index Terms*—parallel file system, Lustre, GPFS, HPC I/O**

I. Introduction

High performance computing has enjoyed a long run of significant growth in high end systems capability. A decade ago we were celebrating the arrival of the first TeraFlop system (ASCI Red)[1]. Today we have PetaFlop/s systems in the Top500 and are busy debating the finer points of ExaFlop/s systems. TeraFlop to PetaFlop represents three orders of magnitude improvement in these ten years. In this time however, disk performance has experienced relatively little increase in performance, perhaps one order of magnitude[2]. Not only has disk performance not kept pace but the increasing computational capability has dramatically increased the data requirements, exacerbating the issue. The memory wall[3] that has bitten HPC so hard over the past decade and a half has gotten much of our attention while the I/O wall has been left with few resources.

It is also the case that it is no longer economically feasible to purchase sufficient I/O capability to be dedicated to each system obtained. Thus we are in a position where the only viable solution is that of a NAS (Network Attached Storage) running a capable, globally accessible, highly parallel file system. For our major systems at the Lawrence Livermore National Laboratory (LLNL), we have not fielded systems with dedicated storage since the installation of the ASC Purple system in 2005. The motive for this testing activity is to revisit the features and performance of the leading file system options we have as we look toward the next generation of large systems in the ASC Sequoia procurement.

About the time the first TeraFlop system was fielded, several DOE sites recognized the I/O bottleneck looming on the horizon and started to take action. One of the initiatives was that of providing funding for the Lustre Parallel File System via the ASCI Path Forward Program. LLNL has a strong interest in open source solutions that enable us to hunt and fix bugs as well as push development down a path beneficial to the high-end community. This has caused us to be a strong supporter of Lustre over the years and has resulted in our using Lustre as the current NAS solution. Lustre is our network-based file system that is shared among numerous large clusters including our big BlueGene/L and BlueGene/P (Dawn) systems.

During this same ten years, few other products have emerged in this space, Panasas' PanFS being the most notable. IBM's GPFS (Global Parallel File System)[4] has been around for a long while and in use as directly attached storage (storage nodes on the internet network) on IBM systems at Livermore including ASCI White. At the same time we have been encouraging and participating in the development of Lustre, we have been happily running a multi-PetaByte GPFS on Purple. In our environment, GPFS is an obvious candidate for a NAS.

We are currently planning the environment for Sequoia, a 20 PetaFlop/s IBM system with an I/O target of 512 GB/s, and a stretch goal of 1TB/s. In debating the file system options, we wanted to compare important aspects (ease of administration, performance, resilience, etc) of the two leading solutions with identical hardware. The intention was to get an apples-to-apples comparison between Lustre and GPFS. However, as you will see in the results presented here, the systems behave differently under even slightly different workloads. These differences translate in to widely varying levels of performance, especially on the metadata tests. As is always the case, your technical choice needs to be driven by your applications. This work shows this to be true in your choice of network attached parallel file systems as well.

This work was supported by the U.S. Department of Energy under Contract #NA27344

II. Background

High Performance Computing applications today typically consist of hundreds or thousands (even hundreds of thousands) of communicating MPI tasks. In the process of executing a scientific application it is reasonable that each task have the ability to efficiently create a file, open it, both write and read data, query the size of the file and finally close it. In the case of checkpoint activity, each task may create and write a separate file.

In the scientific simulation environment promulgated by LLNL, there is a smooth workflow consisting of data preparation, staging, and running the application on the computing resource, followed by post-processing the data. Because in most cases, it is not efficient to tie up the computational resource with the data handle activities, we provide network file systems to our users. The data preparation, staging and post-processing activities can all take place on smaller and cheaper equipment that frees up the big systems for the next simulation run. If the file system to be used is only mounted on the computational resource, then the big expensive system must be used in all activities involving handling the data.

A more cost effective approach is to utilize a Storage Area Network, or SAN. This approach has the SAN mounted by all the systems involved in the workflow and the user is able to utilize the best system for the task at hand. For example, we have dedicated visualization clusters that are outfitted with either GPUs, or large memory nodes. In all cases, these systems use the client-side software to globally mount the SAN file system. The workflow is for the setup to take place on a utility system, the computation run on the high-end system, the post-processing/visualization to happen on the visualization resource, and data archiving to be performed via data movement servers. Data need not move or be redundantly copied in this environment, easing the workflow significantly.

The cluster hardware used to drive the storage resource was Hyperion – an 1152 node X86_64 system with a full Fat Tree InfiniBand 4x DDR interconnect. One half of the nodes on Hyperion use Intel Harpertown processors, the other half use Intel Nehalem. All compute nodes are dual-socket, quad-core. As is the recipe for all our systems, I/O from Hyperion is routed via gateway (GW) nodes that take IB and produce 10GbE to our SAN infrastructure. The 10GigE network routes through a core router to 10 GigE storage edge switches to which we have connected four storage server nodes plus one metadata server node that front-end a Data Direct Networks DDN 9550 disk controller. The network is capable of delivering five GB/s of bandwidth from the cluster to the DDN RAID system. This is about 2X the throughput of the DDN system. This configuration matches the structure all of our systems use to communicate with the global shared file systems.

The LLNL SAN infrastructure is based on 10GbE with edge routers and a core switch. In this way all of the file systems on a network are available to all of the systems on that network. The path from a client to a storage server starts with the client calling the write or read command, which passes the request across the IB network to the GW node. There the message is translated from IB to IP and enters the client edge router near Hyperion. The edge router connects to the core router which forwards to the storage edge router and finally on to the storage server node (see Figure 1). The storage server node writes or reads the data to/from the DDN 9550 storage system and the acknowledgement or data travels back across the network to the client node.

III. The Lustre File System

The Lustre File System is an open source project recently acquired by Oracle. The Lustre architecture provides physical and logical separation of data and metadata. The metadata is information about a file or directory (name, access time, etc) and is stored in a single physical system called the Meta Data Server (MDS)[1]. The actual data portion of the file is split up and accessed in parallel across a (potentially large) group of storage units known as Object Storage Servers (OSS) with software servers called the Object Storage Targets (OST). In practice at Livermore, our file systems consist of up to 256 OSS nodes each with multiple OSTs (software) per single OSS (hardware) node.

As mentioned earlier, LLNL has long been a supporter of the Lustre open source project. In fact, we have several developers on staff that participate in development and are very familiar with the both codebase and operational aspects of the system. We also have a close working relationship with the Lustre developers and the Hyperion cluster is the primary at-scale test resource for the Oracle Lustre development team. For this work, we were using Lustre version 1.6.6 with numerous patches from our local development team.

IV. The Global Parallel File System (GPFS)

GPFS is a mature product available from IBM. GPFS is able to operate either in a direct-attached mode, where the disk nodes are directly attached to the same internal network as the compute nodes, or as NAS. Livermore currently has a multi-PetaByte direct-attached GPFS file system on our Power5+, 100 TFlop system named Purple.

GPFS also separates metadata and file data, however it does not use a dedicated MDS node like Lustre. With GPFS, each

[1] The clustered Metadata Server has been architected and implemented, but has yet to make it into the shipping product.

client node takes the role of the MDS node, and handles metadata requests for anyone on the network interested in the file. GPFS has an option to centralize the metadata itself, but it is architected to run the metadata service in parallel on the client nodes.

We are not as familiar with GPFS and have chosen not to gain access to the proprietary source code. Our experience with GPFS is in the directly attached mode – we have no experience with the system in a network setting as we are attempting to test here. In our discussions and surveys, we find that we are testing a usage model of GPFS that is supported, but not widely used. We used GPFS version3.2 for this work.

V. Setup for Testing

The hardware test environment is shown in Figure 1 and labeled for the Lustre configuration. The physical storage is a single DDN9550 system capable of delivering 2.4 GB/s with 4MB I/O requests or 2.0 GB/s with 1MB requests. On top of the storage hardware there are four OSS nodes which are Dell R610 systems with dual socket, quad core Intel Nehalem E5530 processors at 2.4 GHz. Each OSS node has a 10 Gb/s Ethernet interface to the storage network and an SDR InfiniBand (IB) connection to the DDN 9550. There is a pair of MDS (failover) servers (Dell R610) attached to a 16 bay SAS/SATA JBOD enclosure with sixteen 15K RPM SAS drives.

The network is a three-stage 10Gb Ethernet network with a Cisco Nexus 7018 as the core, an Arista 7148S as the edge switch on the storage side and a Cisco Nexus 5020 next to the 4 GW nodes. This network should be capable of providing 5 GB/s of bandwidth, or about 2X the capability of the DDN9550 RAID system.

The clients are the 1152 (8 core) nodes of Hyperion. Half the clients are nodes identical to the R610 described above, the other half contain Intel Harpertown LS420 processors at 2.5 GHz. The clients send file system requests via the gateway nodes across the 10 Gb/s Ethernet to the OSTs. Through the gateways, we expect to be able to provide an aggregate of 5 GB/s, as mentioned previously.

We are confident in our Lustre configuration skills, so we chose to provision the hardware and automate the file system configurations for these tests ourselves.

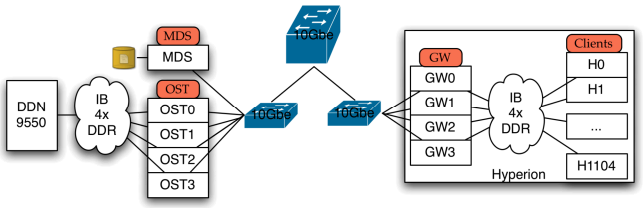


Figure 1: Hardware test environment - Lustre

Due to our inexperience with GPFS in the test environment, we invited IBM to install, tune, test and review the results of our testing activities. Over a period of about one month the system was tweaked, tested, and tweaked again. In the process we learned quite a bit about GPFS's architecture and the features of its administrative tools. One of our findings is that the differences in the architecture of GPFS and Lustre are so significant so as to directly and substantially impact the performance results achieved.

Overall, our experience in observing and assisting with the GPFS install and provisioning was strongly positive. The tools provided by IBM to work with GPFS are very advanced and mature. The file system itself was incredibly resilient. In fact, part of our testing beyond the scope of this document was to manually fail portions of the system to gauge the response. GPFS passed these harsh tests with ease and we did not see any failures for the duration of our multi-week experience with the system. At the same time and on production systems, we were gingerly attempting to use Lustre's manual fail-over with mixed results.

In Figure 2 we have the GPFS configuration for the tests. We have the same physical hardware as with Lustre. The main difference is that the MDS server that was used for Lustre no longer has that role and becomes simply a GPFS file server alongside the other 4 nodes (which were OST's in the Lustre configuration).

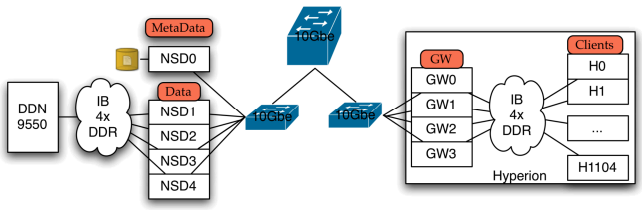


Figure 2 Hardware test environment - GPFS

The metadata server for GPFS consists of the client nodes themselves, operating in parallel with each responsible for the metadata it creates. The metadata does make its way to the disk on the left hand side of figure 2 and is then available via the NSD0 file server node. This architecture is flexible in that NSD0 can be either a set of systems, or run on the data server nodes alongside the data aspect. This is quite different than Lustre's single MDS and has a strong influence on metadata performance as described below.

VI. Benchmarking Strategy

Testing was done in an automated fashion with each test repeated at least twice with the best-case result shown. Tests were run on both a clean (< 5% full) file system as well as an aged file system (>80% data blocks used) which was prepared by filling up the file system and randomly removing files to bring it back to ~80% of the data blocks used). Between runs, the file system was completely removed and reinstalled to avoid any lingering issues caused by the last run.

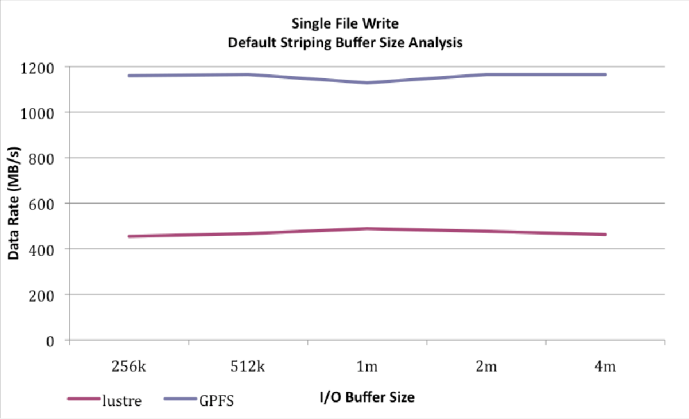
We used the Livermore developed test harness of IOR[5] (http://sourceforge.net/projects/ior-sio) and mdtest[6] (http://sourceforge.net/projects/mdtest/) which were fully automated via scripts.

Testing took several weeks of time for each of the candidates as we discovered issues that caused us to want to invalidate and rerun the tests. As mentioned above, we learned things about GPFS that necessitated some changes in the tests themselves.

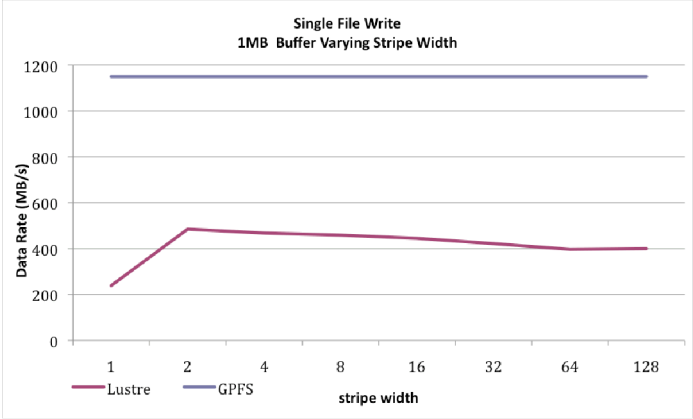
VII. Throughput Testing

Throughput or bandwidth testing is intended to provide insight into the ability of the file system to deliver a significant fraction of the peak bandwidth of the hardware. As mentioned previously the peak for our setup was 2.4 GB/s. The IOR test harness is an MPI-coordinated bandwidth test capable of testing single and multi-node performance for file-per-process and single-shared-file access and both contiguous and noncontiguous data patterns. IOR is good at producing a high, sustained I/O load on a parallel file system. The tests run with IOR included "transfer size scaling", "block size Scaling", "file per process scaling", "segmented, shared file performance", and "interleaved, shared file performance".

In the first test, we focused on performance for a single client. This is the case where one MPI task handles all of the I/O for an application, or the case where we are streaming data to tertiary storage from a single client. With a peak capability of 2.4 GB/s, we see GPFS getting about 50% and Lustre about 25% of that bandwidth.



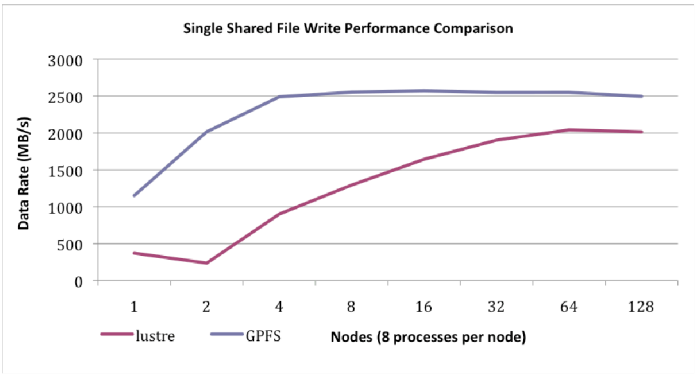
We believe GPFS is better able to take advantage of multi-threading and lays data for a single file onto all servers in a 4MB round-robin fashion. Lustre will stripe 2 way by default, but this does not turn out to be the performance limiting factor as further testing with larger stripes still did not achieve parity with GPFS as seen in the next graph.



We had previously discovered that Lustre's single file performance is limited to the performance of a single core due to it's design which does not take full advantage of the number of cores and threads in modern processors.

We intentionally ran the read tests in a "cache-busting" mode. That is, we arranged the tests such that the file is not read by the process that wrote it. This will defeat the client caching mechanism and cause the data to be read from disk. This approach does not work on a single node so the results for single node performance are not indicative of real performance and are not presented.

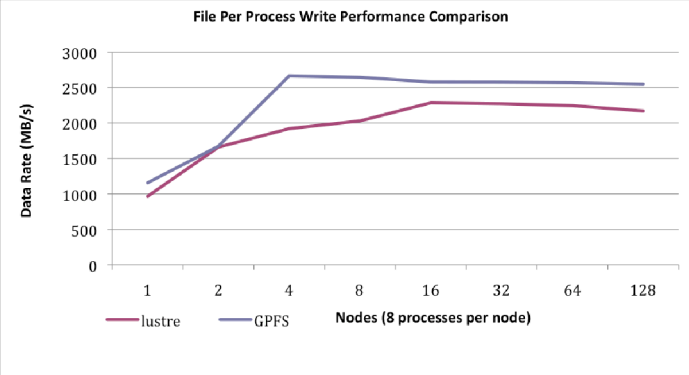
We next move onto testing parallel jobs I/O performance. In the first test, all nodes (ranging from 1-128) write to the same shared file. This is a common mode of application I/O where each task will do a seek-write to position its data at the proper off-set within a file.



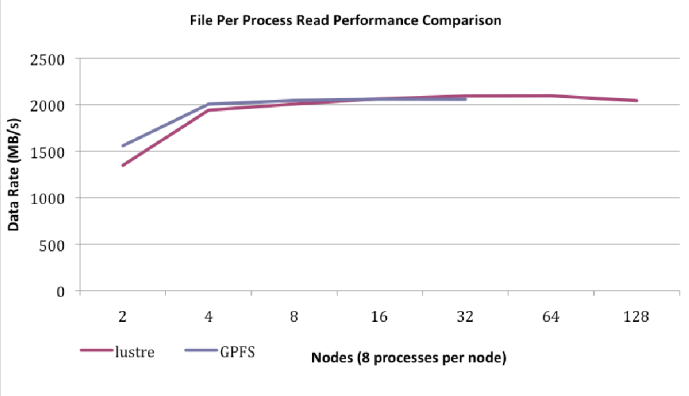
Working with LLNL, IBM has made an effort to match GPFS's file per process performance by tuning the shared file

throughput. As the above graph shows, GPFS quickly reaches peak at 4 nodes and remains consistently at peak up to the 128 nodes used in the test. GPFS also reaches the half-peak performance level with a single node. Lustre performs less well, increasing performance on a more gradual basis and not hitting a plateau until 64 nodes. Lustre doesn't reach the half-peak performance level until 4 nodes and has a dip in performance from 1 to 2 nodes.

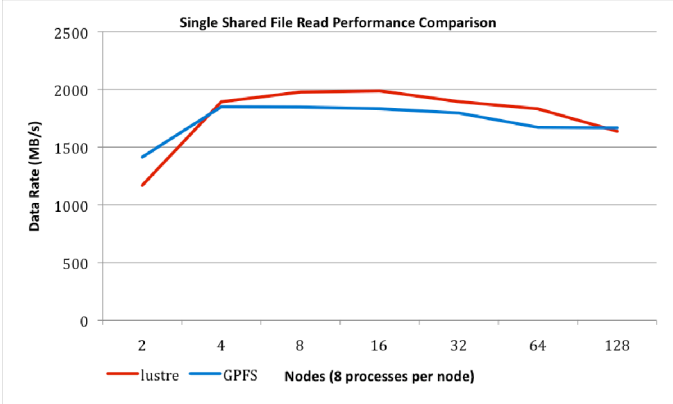
Another common way for applications to do I/O is for each process to open a file to write its portion of the data. This is also a common checkpoint mechanism used by our applications. Here we again see GPFS peak at about 4 nodes, but Lustre does much better trailing GPFS by only about 10% and peaking at 16 nodes. This is as expected due to the fact that GPFS does better load balancing and Lustre has the additional overhead of calculating checksums.



For parallel read performance, in the next graph, we see that both file system are well-matched at the limits of the test hardware capabilities.



For reading a single shared file, GPFS outperforms Lustre by a slim margin at first, then the opposite is true as the test scales up. They end up equal at the highest scale for this test.



In bandwidth testing, we had expected GPFS to have a wider margin of advantage due to its larger network blocksize (4 MB vs Lustre's 1 MB blocksize). The tests here do not appear to show this due to the DDN's architecture which provides sufficient disk bandwidth behind the RAID controller to saturate the controller with only 1 MB blocks. To explore this area, we disabled a portion of the RAID devices. GPFS is able to saturate the controller's bandwidth in this configuration with only 24 tiers of disk where Lustre required 48. This is a subtle but important result that could dramatically impact the cost of an installation by reducing disk requirements.

VIII. Metadata Performance Testing

The metadata test driver used was mdtest, developed at LLNL and available as described above. Logical separation of metadata and file system data is one of the key features of parallel file systems. As we learned in this exercise, the performance of metadata is very dependent upon application behavior. The tests we ran turn out to be worst case for GPFS (with its distributed MDS) while seemingly favoring Lustre (with it's single MDS).

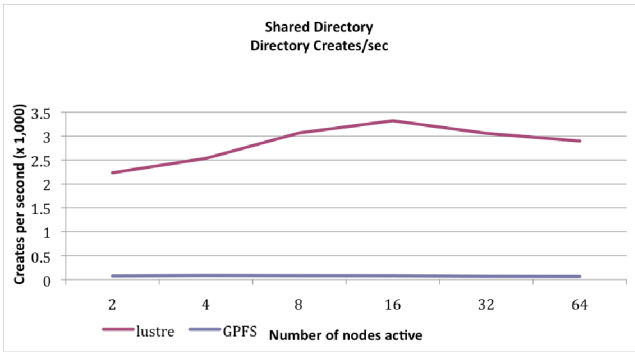
At a high level we tested two situations. The first is the situation in which an application does its file system activity entirely within a single directory on the file system. The other extreme is where the application creates a directory for each MPI tasks and the tasks subsequently performs all of its file system work in its own directory separate from the other tasks. While there is plenty of middle ground, we find that testing these two modes is indicative of performance for our applications. For these metadata tests files were zero length (contained no actual data).

In running the tests in this straightforward manner, we were surprised to see GPFS outperform Lustre by a wide margin. As we learned about the GPFS architecture, we understood the distributed metadata design and realized the speedup was a result of the client nodes acting as metadata servers and caching file operations locally. We decided this mode was not reflective of our applications so changed the tests to cause the "stat" command to be issued from a neighbor node.

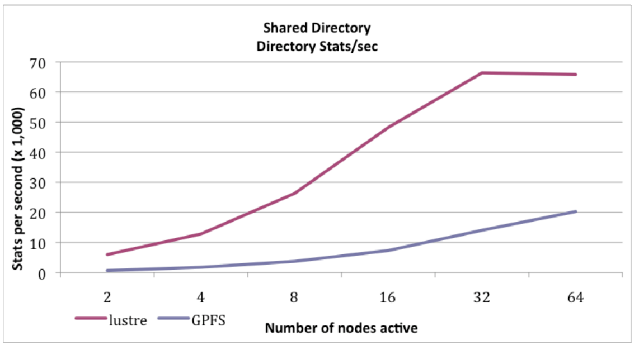
This seemingly subtle change neutralized the advantage GPFS had and the tests were rerun. The GPFS results changed to be a fraction of those from Lustre and several orders of magnitude less than they had been. What we learned (and what follows in the results) is that in GPFS, a stat operation from a node other than the creator forces the creating node to flush to disk, which is a very expensive operation. The result is that GPFS is penalized quite significantly. The penalty of the flush is more than would occur in a situation where a file is not stated immediately but some time afterward when the file system flush has an opportunity to occur naturally.

We decided to continue on this path because these are the test parameters dictated by our applications that have been developed and use the Lustre file system. Had our applications been optimized to work with (in a NAS environment as opposed to directly attached), they would have to behave differently and the mdtest script would reflect this. It was communicated to us that simply having task zero (or any single task) do the metadata operations would reduce this problem and bring the performance up at least to the level of Lustre. It is important to note that this is the main factor in the results. The different architectures produce widely different performance profiles based on these seemingly simple changes.

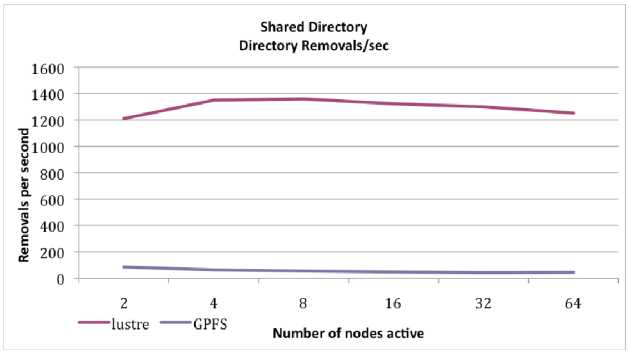
Tests were run on multiples of 2 nodes up to 128 with 8 processes per node. In the test, each process creates, then stats a different node's files and finally deletes the 100 files or directories it created.



For the first case of creating directories within a shared directory, Lustre was significantly faster than GPFS. This is due to the fact that in creating a directory, GPFS must add the parent directory ".." pointer. This requires exclusive lock the parent directory for each create effectively serializing the process and adding overhead. Lustre, with it's single MDS looks much better for this particular test, though it shows signs of tailing off past 16 nodes.

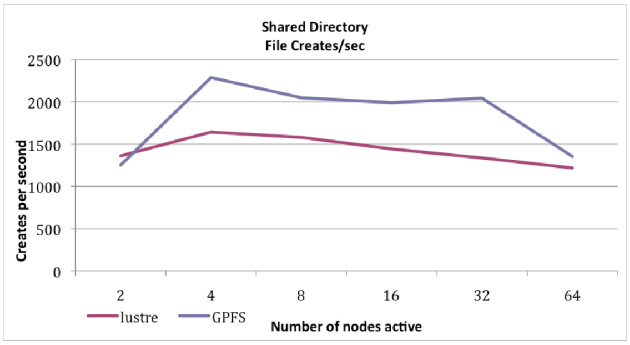


Similarily for stats, the single MDS is a significant advantage for Lustre resulting in over 3X the performance of GPFS. GPFS does better in this test, but our conclustion is that distributed architecture of GPFS needs a heavier load to show its benefits over the single MDS for Lustre – for smaller systems and with this method of activity, Lustre appears to have an advantage.



The file removals, much like creates, are in favor of Lustre with Lustre directory removal performance over 12X that of GPFS. This is due to the synchronization overhead, again, in GPFS' distributed algorithm.

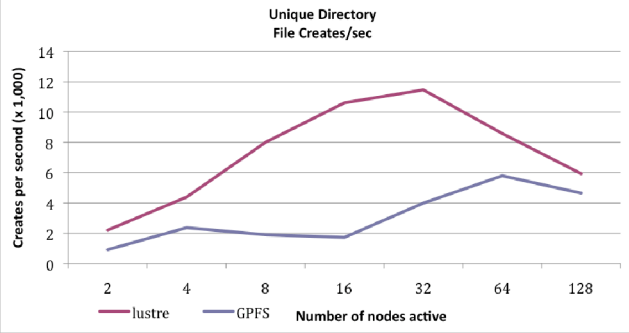
In testing file creates, we find an interesting difference in GPFS' performance. Here, GPFS does quite well, outperforming Lustre by 30% in places. The reason for this is that creating many files in a shared directory is common practice and the GPFS developers have provided a well-performing solution. This is because, after listening to feedback from users, the developers of GPFS have implemented shared directory file creates with a parallel algorithm enabled via a fine grained directory lock.



The testing results for file stats and removals within a shared directory are substantially similar to that of the directory case. For that reason, we are not including them here. [but may update the graphs to overlay for the final submission]

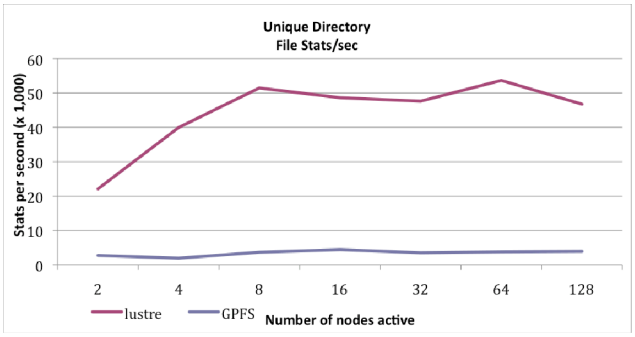
Thus far, we have focused on metadata tests that create, stat, and delete files and directories within a single flat directory. The other main alternative is to have a directory per MPI process, partitioning the activities in a manner that should enable the file systems to proceed in parallel.

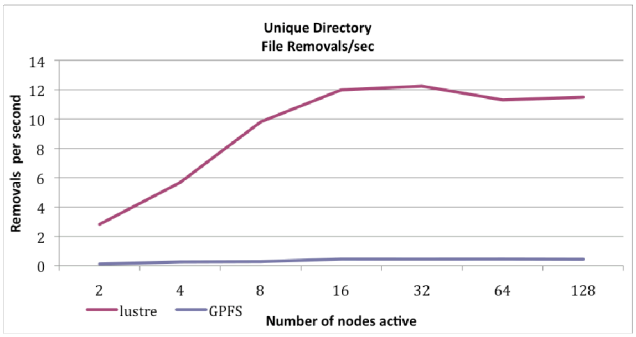
In our first test for the unique directory method, we see that Lustre's performance scales nicely up until about 32 nodes where it starts to drop off. The GPFS performance is not as good as Lustre, only catching up as Lustre declines at 64 nodes and tailing off in a similar way past that point.



File and directory stat operations within a unique directory are not significantly different from that of a shared directory. For GPFS, both situations force a node to flush to disk to answer the stat operation since it is initiated from a different node.

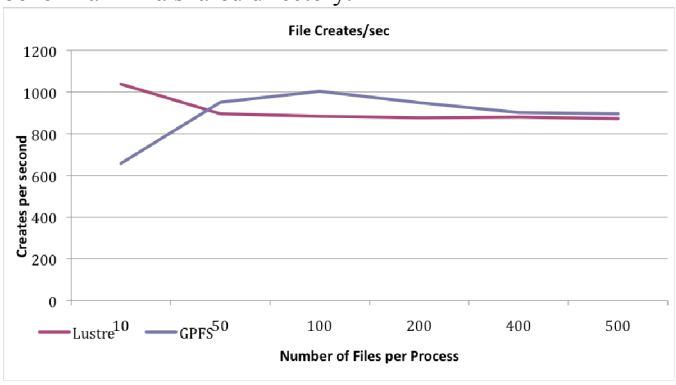
Finally, and for completeness, removal of files and directories within a unique directory show poor performance for GPFS relative to Lustre.



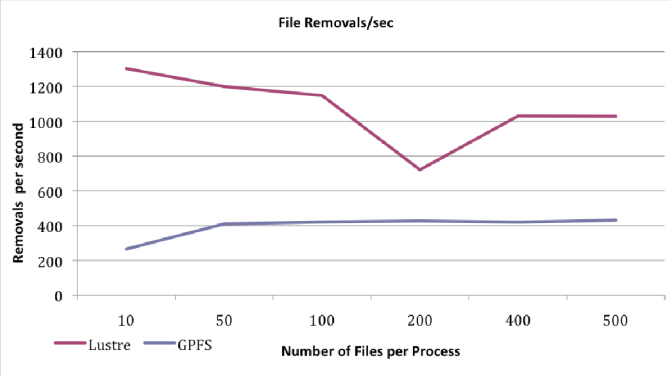


IX. Per Process Metadata Scaling Study

The above tests used a static number of operations (100) per node and varied the number of nodes used. Lustre outperformed GPFS in these tests. In order to understand scaling up, we chose to hold the number of processes static at 128 and vary the number of file and directory operations for each process from 10 to 500. Here we again used the mdtest benchmark in a shared directory.



The file stats per second in a unique directory gives hints that the GPFS architecture may have advantage if put under a heavier load. It is not easy to imagine a situation where every process needs 500 files, but perhaps if the number of files itself is the key to performance, large systems with hundreds of thousands of processes may show GPFS to be better suited. Unfortunately we are not able to test at these levels and know of no site with a NAS-based GPFS installation and a hundred-thousand plus node system.



X. Conclusion

For the throughput performance tests, both file systems were able to drive the small backend hardware at high rates, with GPFS outperforming Lustre for the most part by 20% or more for writes and they were about equal on reads. GPFS is also able to more quickly saturate the available storage hardware due to its larger network block size. The data here also indicate that GPFS is much better at making use of multiple cores in a system when compared to Lustre.

With respect to metadata performance, Lustre is faster in all but the case of creating a large number of files in a shared directory. In most metadata test cases, the performance of Lustre is significantly faster than that of GPFS. This is somewhat counter to expectations given GPFS' distributed metadata architecture. As noted, the tests were very harsh toward the way GPFS operates and very friendly toward Lustre's method. To thwart the caching behavior of GPFS, we perform stat operations from a different node than the one holding the metadata, which has the effect of forcing GPFS to sync to disk. In the same tests, Lustre is permitted to cache the results and artificially appears to have a large advantage. The truth here lies somewhere between these extremes and the thing that matters most is what your application does or can be made to do in the way of being GPFS-friendly and/or Lustre-friendly.

Metadata operations are logically associated with creating files and directories, querying them for information and deleting them. For the most part, our expectation is that the majority of application activity involves reading and writing files vs. the management of metadata. Unless the application is using the file system as a communication device (which is a bad idea these days), metadata operations are likely a tiny fraction of the file system activity. Yet performance is critical as we have learned with BlueGene/L and BlueGene/P where applications running on ~132,000 MPI tasks can each create a file for results or checkpoint data.

As we had multiple weeks of testing with each system, we had the opportunity to perform many more tests than presented here. We also studied the two systems with an eye toward administrative tools, resilience and total cost of ownership. These non-performance metrics are important to the usability and reliability in the day-to-day use of the system. LLNL has many years of experience with Lustre and we found ourselves attracted to the administrative tools provided with GPFS. It is no small effort to keep a production center running and GPFS administrative tools are mature and offer simple, intuitive interfaces designed to handle the day-to-day tasks that are necessary in a large production environment. These considerations are significant and should be taken into account when selecting a solution. GPFS is much further ahead than Lustre in this area.

LLNL has a long history with both Lustre and GPFS. We have pushed hard to separate the disk from the system and have shown that network based file systems are the right answer for large file systems in an environment with multiple supercomputers. We have worked closely with the Lustre community to help make this solution viable for the community. We are happy to see more sites using GPFS in this way and hope to participate in the growing community supporting these architectures.

References

[1] http://www.sandia.gov/ASCI/Red/RedFacts.htm
[2] John May, "Parallel I/O for High Performance computing", Morgan Kaufmann 2001, p 22.
[3] Hitting the Memory Wall: Implications of the Obvious, Wm. A. Wulf, Sally A. McKee, Computer Architecture News, 1995
[4] GPFS: A Shared-Disk File System for Large Computing Clusters: Frank Schmuck and Roger Haskin, IBM Almaden Research Center. http://www.usenix.org/publications/library/proceedings/fast02/full_papers/schmuck/schmuck_html/index.html
[5] Richard Hedges, Bill Loewe, Tyce McLarty and Chris Morrone, "Parallel File System Testing for the Lunatic Fringe: The Care and Feeding of Restless I/O Power Users," Mass Storage Systems and Technologies – MSST 2005.
[6] Ibid