



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Comparison of Leading Parallel NAS File Systems on Commodity Hardware

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

September 20, 2010

Petascale Data Storage Workshop
New Orleans, LA, United States
November 14, 2010 through November 14, 2010

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Comparison of Leading Parallel NAS File Systems on Commodity Hardware

Keith Fitzgerald, Mark Gary, Richard Hedges, 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 than production systems, 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 seen increased performance in the last decade of about three orders of magnitude due to advances in processor speed, core count, and scale of networks. In the same time frame, disk performance has experienced relatively little increase in performance, perhaps one order of magnitude. The increased computational capability has dramatically increased the data requirements, exacerbating the issue. For HPC systems this leads to a requirement for a parallel file system: A shared disk file system, presenting a single name space which can be written to in a coordinated fashion by all nodes of a cluster, and potentially comprised of thousands of individual disk drives.

While we were familiar with the GPFS (proprietary) file system from IBM [1], when we began developing HPC clusters based on commodity hardware and open source software (Linux), there was no applicable open source solution. This led to funding development of the Lustre Parallel File System[2] via the ASCI Path Forward Program. LLNL has a strong interest in open source solutions that enable us to be involved in our own support and development as well as influencing a path beneficial to the high-end community. Thus we have been a strong supporter of Lustre over the years. Lustre is our current NAS solution: a network-based file system that is shared among many large

clusters including our BlueGene/L and BlueGene/P (Dawn).

A few other products have emerged in this space, Panasas' PanFS [3] being the most notable. IBM's GPFS (Global Parallel File System) 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.

Previously, our lab has published regarding the details and rationale for our testing protocol [4]. Several groups have investigated and compared the architectures of the existing parallel file system solutions [5-7]. For the last several years, there has been a popular tutorial comparing the file systems, including performance information, and practical advice regarding using the file systems[8].

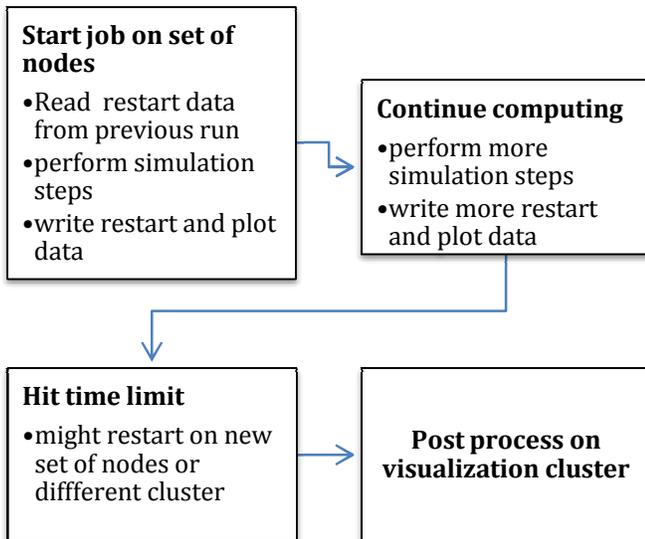
We are planning the environment for Sequoia, a 20 PetaFlop/s IBM system with an I/O target of 512 GB/s, and a stretch goal of 1TB/s. In considering file system options, we wanted to compare aspects (ease of administration, performance, resilience, etc) of the two leading solutions with identical hardware. The intention was to get an apples-to-apples comparison between Lustre and GPFS. At the scale we are investigating here, these tests (even performance tests) should be viewed as comparing functionality of the two file systems.

II. LLNL WORKFLOW

The HPC applications that we encounter in the Livermore Computing Center typically consist of hundreds or thousands (or hundreds of thousands) of communicating MPI tasks. In the process of executing an application it is common that each task have the possibility to efficiently create a file, open it, 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.

Here we will diagram the dominant workflow for LLNL simulation applications as seen from the perspective of the data created by the simulation code. It has been characterized as "write once, read seldom". "Restart data" describe the

complete state of a simulation, so that a simulation can be restarted and continued from the point represented by that data. “Plot data” may be sufficient for analysis (post processing) but without sufficient state to resume the simulation.



The file system strategy that we have devised to support this workflow is to utilize a Storage Area Network, or SAN. This approach has the SAN mounted by all the systems involved in the workflow: the user is able to utilize the best system for the task at hand. For example, we have visualization clusters that are outfitted with GPUs, or with larger memory nodes. In all cases, these systems use the client-side software to mount the SAN file system. Data need not move or be redundantly copied in this environment, easing the workflow significantly.

III. THE LUSTRE FILE SYSTEM

The Lustre File System is an open source project recently acquired by Oracle [6]. 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.

We presently have several developers on staff that participate in development and are very familiar with the both codebase and operational aspects of Lustre. 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.

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

For this work, we were using Lustre version 1.6.6 with numerous patches from our local development team. While this might reduce the relevance of these results for those readers considering a more standard Lustre installation, testing with these patches is relevant to our primary goal of planning the Sequoia environment.

IV. THE GLOBAL PARALLEL FILE SYSTEM (GPFS)

GPFS is a mature product available from IBM [5]. 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 client node takes the role of the MDS node for a subset of files, and handles metadata requests for any process on the network interested in one of those files. 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 internals 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 version 3.2 for this work. For the purpose of these tests, We had IBM personnel configuring the software, review results, adjust configuration, etc.

V. SETUP FOR TESTING

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 our present 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 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. Specifically, 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. 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 a 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.

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. There are 48 tiers of 250GB SATA disks in 24 8+2 RAID3 luns per controller. 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.

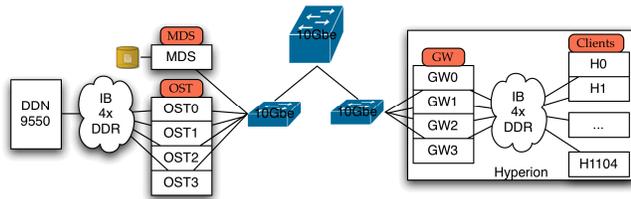


Figure 1: Hardware test environment - Lustre

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 that they directly and substantially impact the performance results achieved.

In Figure 2 we have the GPFS configuration for the tests. We use 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). Metadata operations are still isolated on the high performance disks used exclusively for metadata operations. This was to insure that file data operations do not impact metadata operations.

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.

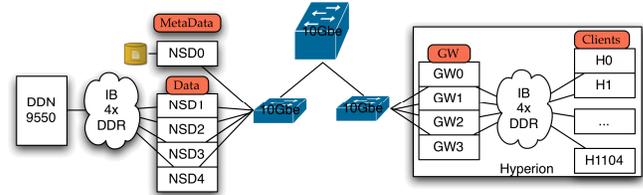


Figure 2 Hardware test environment - GPFS

VI. BENCHMARKING STRATEGY

Testing was done in an automated fashion with each test repeated at least twice with the best-case result shown. While this approach has been called into question, our reasoning is that is the method we use for testing on production systems, where we are assuming that the best case that we can measure is one without contention from other file system activity.

We used a Livermore developed test harness with IOR (<http://sourceforge.net/projects/ior-sio>) for throughput measurements and mdtest for metadata performance measurements. (<http://sourceforge.net/projects/mdtest/>).

In selecting test conditions, it is our basic assumption that data will not be read by the node that wrote it. When reading a restart file, for example, it would be unreasonable to assume that subsequent accesses would be by the same node, given our workflow. Those accesses could even be coming from an entirely different cluster. Another case for file stats is "ls -l" from a single interactive node. In these cases the stats, accesses, etc come from nodes or even clusters other than those creating the files. For this reason, we have run all of the tests where subsequent access (reads or stats for the metadata testing case) are performed by a different node. This will defeat the client caching mechanism and cause the data to be read from disk in each case.

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. IOR 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.

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

At LLNL, the most common way for applications to do I/O is for each process to open a file to write its portion of the data to its own file. Here we see GPFS peak of 2668 MB/sec at 4 nodes, but Lustre does trail GPFS by only about 10% with a peak of 2286 MB/sec 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 the corresponding read performance, we see that both file systems are well-matched at the limits of the test hardware capabilities.

Working with LLNL in the past, IBM has made an effort to match GPFS's file per process performance by tuning the shared file throughput. GPFS quickly reaches peak of about 2500 MB/sec at 4 nodes and remains reasonable consistently near the 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 hitting a plateau of ~2000MB/sec at 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. 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. Both achieve ~2000MB/sec at the highest scale.

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 subtle but important result could dramatically impact the cost of an installation by reducing disk requirements.

VIII. METADATA PERFORMANCE TESTING

mdtest can measure the rates of file and directory creation, stat'ing and removals for the situation where all processes act in the same directory and the situation where each process works in its own directory. We tested both situations. The situation where an application does all of its file system

activity within a single directory of the file system is the most common case for our user applications. The other case where the application creates a directory for each MPI task and the task subsequently performs all of its file system work in its own directory is less common, but is occasionally encountered.

For these metadata tests files were zero length (contained no actual data). 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.

GPFS metadata performance where stats come from a neighboring node, were a fraction of those from Lustre, by a factor up to an order of magnitude. For 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.

When the stats are performed by the creating node, GPFS outperforms Lustre by a wide margin. As we understood the distributed metadata aspect of the GPFS architecture, we realized the speedup was a result of the client nodes acting as metadata servers and caching file operations locally.

We selected that subsequent operations not come from the node creating the files/directories because these test conditions are most appropriate for our applications and workflow. We acknowledge that the applications have been adapted to work well with the Lustre file system. Had our applications been optimized to work with GPFS they would have to operate differently and the mdtest measurements would reflect this.

It was communicated to us that simply having task zero (or any single task) do all of the metadata operations would reduce this problem and bring the performance up at least to the level of Lustre. The different architectures produce widely different performance profiles based on these seemingly simple changes.

For 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 its single MDS looks much better for this test, though it shows signs of tailing off past 16 nodes.

Similarly 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 conclusion 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's distributed algorithm. The testing results for file stats and removals within a shared directory are substantially similar to that of the directory case.

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.

In our 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.

The previous metadata tests used a constant 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.

IX. CONCLUSIONS

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 with about equal performance on reads. The data here also indicate that GPFS is much better at making use of multiple cores in a system when compared to Lustre. Further, due to its larger message size, GPFS can saturate the RAID controller capacity with fewer backend disks.

For metadata performance, GPFS shines for stat operations being performed by the node that created the file or directory since metadata operations can be effectively cached. When stats are not from the creating node, Lustre is faster in all but the case of statting a large number of files in a shared directory. In most metadata test cases, the performance of Lustre is significantly better than that of GPFS. In the same tests, Lustre is permitted to cache the results and artificially appears to have a large advantage.

Metadata operations are 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 writing and reading files vs. the management of metadata. Unless the application is using the file system as a communication device (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.

REFERENCES

1. Frank B. Schmuck, Roger L. Haskin, GPFS: A Shared-Disk File System for Large Computing Clusters, Proceedings of the Conference on File and Storage Technologies, p.231-244, January 28-30, 2002
2. "Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System", white paper available at <https://www.sun.com/offers/details/LustreFileSystem.xml>
3. David Nagle, Denis Serenyi, Abbie Matthews, The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage, Proceedings of the 2004 ACM/IEEE conference on Supercomputing, p.53, November 06-12, 2004
4. 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.
5. Margo, M. W., Kovatch, P. A., Andrews, P., and Banister, B. "An Analysis of State-of-the-Art Parallel File Systems for Linux." The 5th International Conference on Linux Clusters: The HPC Revolution 2004, Austin, TX, May 2004. This paper compares GPFS, Lustre, PVFS on an IA-64 linux cluster.
6. J. Cope, M. Oberg, H. M. Tufo, and M. Woitaszek. Shared parallel file systems in heterogeneous Linux multi-cluster environments. In Proceedings of the 6th LCI International Conference on Linux Clusters: The HPC Revolution, Chapel Hill, North Carolina, Apr. 2005. The paper provides an investigation of PVFS2, GPFS, Lustre, and TerraFS across multiple Linux clusters.
7. M. Oberg, H. Tufo, and M. Woitaszek, "Exploration of Parallel Storage Architectures for a Blue Gene/L on the TeraGrid," in 9th LCI International Conference on High-Performance Clustered Computing, March 2008. The paper investigates storage cluster configurations IBM Blue Gene/L computer using GPFS, Lustre, and PVFS.
8. Robert Ross, Robert Latham, Marc Unagast and Brent Welch, "Parallel I/O in Practice", Proceedings of Supercomputing 2009