An Analysis of State-of-the-Art Parallel File Systems for Linux

Martin W. Margo, Patricia A. Kovatch, Phil Andrews, Bryan Banister

San Diego Supercomputer Center
University of California, San Diego
San Diego, CA

mmargo@sdsc.edu, pkovatch@sdsc.edu, andrews@sdsc.edu, bryan@sdsc.edu

Abstract.

Parallel file systems are a critical piece of any Input/Output (I/O)-intensive high performance computing system. A parallel file system enables each process on every node to perform I/O to and from a common storage target. With more and more sites adopting Linux clusters for high performance computing, the need for high performing I/O on Linux is increasing. New options are available for Linux: IBM's GPFS (General Parallel File System) and Cluster File Systems, Inc.'s Lustre. Parallel Virtual File System (PVFS) from Clemson University and Argonne National Laboratories continues to be available. Using our IA-64 Linux cluster testbed, we evaluated each parallel file system on its ease of installation and administration, redundancy, performance, scalability and special features. We analyzed the results of our experiences and concluded with comparison information.

Introduction

Many high performance computing (HPC) applications read and write large quantities of data. For example ENZO[1], an astrophysics code, outputs data sets on the order of 10 TeraBytes (TB) in one computational job on a supercomputer. As Linux clusters proliferate and more applications port to them, the need is greater than ever for parallel file systems to assist with the I/O. More parallel file systems are available for Linux than ever before. Each of these parallel file systems have different characteristics: some are more robust, some perform better for large I/O, etc. We researched and tested three parallel file systems and compared them. These parallel file systems make new options available for all kinds of sites with differently purposed Linux clusters. Some sites may need a low cost parallel file system that's easy to install. Other sites may need a parallel file system that's highly redundant and scalable. We provide a comparison chart to help sites find the appropriate parallel file system for their needs.

Parallel Virtual File System (PVFS)[2]

PVFS was developed by Clemson University to provide an open source, high performance parallel file system for a large user community. It is currently supported by both Clemson University and Argonne National Laboratories.

PVFS stripes data across all its disks (storage targets). The storage targets supported includes SCSI, IDE, RAID arrays or Storage Area Network (SAN).

PVFS possesses three components: a client, a management server and an IOD (Input/Output Daemon) server. The client runs a daemon called pvfsd, which enables processes to perform I/O operations transparently through the kernel using standard UNIX file I/O. The management server is a single node that records all file attributes, modification dates and permissions. It runs the MGR daemon. The IODs interface directly with both the storage targets and the clients.

The client sends the initial read or write request to the management server. The MGR tells the client which IOD to request the I/O from. Then the subsequent I/O operations are performed directly between the client and the IOD server.

The PVFS daemons use TCP/IP to communicate; therefore they can and run over any network that supports TCP/IP interfaces including Ethernet and Myrinet.

PVFS writes to any file system supported by the Linux kernel. It can only read or write to files and not to block devices like SAN storage targets. Because of this, PVFS cannot make use of a fully connected SAN fabric. PVFS can use the SAN disks as local disks, but will serve the data from these disks over the Ethernet or Myrinet.

General Parallel File System (GPFS)[3]

IBM developed GPFS originally on AIX for SP systems. Recently, IBM ported GPFS to Linux.

GPFS stripes all of its data across all of the storage targets attached to its servers. GPFS is guaranteed by IBM to work on specific IBM disks. On Linux, GPFS operates in two modes: NSD (Network Shared Disk) and SAN direct-attached disk. For both modes, there is one daemon that runs on all the nodes for GPFS: mmfsd. However, there are additional daemons that support mmfsd. GPFS relies on RSCT (Reliable Scalable Cluster Technology). RSCT is a subsystem that monitors the heartbeats of GPFS member nodes. The heartbeat monitors the uptime, network and other attributes of the nodes through the RSCT daemons running on each of the nodes. The RSCT daemons consist of the ctrmc, cthags and cthats processes.

In NSD mode, only a subset of GPFS nodes is directly attached to storage targets. These nodes are called the NSD servers. The NSD nodes serve the storage to the rest of the cluster nodes via TCP/IP over Ethernet or Myrinet. In effect, each NSD node is a metadata server. There are primary and secondary NSD servers. In NSD mode, when a GPFS client receives a request to perform an I/O operation, it

passes the request to the primary NSD server and to the secondary NSD server if the former is unavailable. The NSD server then performs the actual I/O with striping across the storage directly attached to it and sends back an acknowledgement about the success of the operation. The clients and NSD servers communicate directly.

If each of the clients has direct access to common SAN disks, then SAN or direct-attached mode can be used. In this mode, each node acts as it's own metadata server. A Fibre Channel (FC) switch may be needed to connect the client nodes to the storage targets. In SAN mode, the metadata traffic travels over TCP/IP.

GPFS writes to any kind of block device and therefore can make use of a fully connected SAN fabric.

Lustre (Linux Cluster)[4]

Lustre is an open source parallel file system for Linux developed by Cluster File Systems (CFS). The name Lustre comes from combining the words "Linux" and "cluster". It integrates into an existing clustering environment by utilizing existing network, storage and computing resources. It uses many industry standard open source components, such as an LDAP server, XML formatted configuration files and the Portals[5] networking protocol.

The file system architecture consists of Meta Data Servers (MDS), Object Storage Target (OST) servers, and clients. The Meta Data Server keeps track of information about the files stored in Lustre. The OSTs are daemons that run on a node that is directly connected to the storage targets called Object Based Disks (OBD). The OBDs can be any kind of disks including SCSI, IDE, SAN or RAID storage array disks. OBDs can even be a combination of all these types. The clients send a request to the MDS to find where the data resides. After the MDS responds, the clients send the request for the data directly to the OSTs holding the data.

Lustre writes to supported block devices including SAN storage targets. However, since the OSTs and clients cannot exist on the same node, clients cannot perform direct I/O to the OSTs and OBDs. Hence, all data served from the OSTs travel over the Portals supported devices.

Lustre uses the Lightweight Directory Access Protocol (LDAP) to store Lustre XML configuration files. LDAP clients can easily probe configurations and discover the structure of a particular Lustre installation. The clients can discover the kind of network, the number of OSTs, host names of each cluster member, etc.

Lustre uses Sandia National Laboratory's open source networking protocol called "Portals" for communication between the MDS, OST and clients. This protocol is a lightweight layer that works directly with network-specific drivers, bypassing

TCP/IP altogether by using the Network Abstraction Layer (NAL). Supported devices include Quadrics and Myrinet with Infiniband support coming soon. It also works on Ethernet. In other configurations, Portals can use TCP/IP through the NAL as well.

	PVFS	GPFS	Lustre
Storage	Storage target	Storage target	OBD
Block device	Only file system device	Any block device	Block device with proprietary driver layer
Nodes directly connected to storage	IOD	NSD, every node in SAN mode	OST
Storage type	IDE, SCSI, RAID	SAN, RAID	RAID, SCSI, limited SAN
Daemon communication	TCP/IP	TCP/IP	Portal
Metadata server	MGR	NSD, every node in SAN mode	MDS

Table 1. Parallel File System Component Comparison

Other Parallel File Systems for Linux

There are other Linux parallel file systems, such as Sistina's General File System (GFS) and the Galley Parallel File System from Dartmouth University. Most of the other parallel file systems are still in development or in beta. We wanted to look at parallel file systems that could be deployed today. Also, GFS is not available for the IA-64 SuSE Linux at this time.

The Testbed

To explore the features of the three chosen parallel file systems, we built a test cluster consisting of 20 dual processor 1.5 GHz Intel Itanium 2 (Madison) nodes. Each node has 4 GB of RAM and two 73 GB SCSI IBM ESXS Model ST373307LC drives which operate at 10,000 RPM. The nodes run SuSE Linux Enterprise Server (SLES) 8 with a custom 2.4.21_144SMP kernel. We used the xCAT cluster installation suite from IBM to install this cluster.

Each node has a Myrinet M3F-PCIXD-2 adapter primarily used by applications for Message Passing Interface (MPI). Each node also has a dual-port on-board IntelPro 10/100/1000 Ethernet adapter. The first port connects to a Force 10 Gigabit Ethernet (GE) router for general-purpose networking. For the management network, the second Ethernet port is connected to a Cisco 10/100 switch.

Each node also has 2 Gb/s 64 bit 133 MHz PCI-X Qlogic QLA2340 Host Bus Adapter (HBA). A Brocade Silkworm 3900 Fibre Channel (FC) switch connects to each adapter on each node to form the SAN infrastructure. Five Sun StorEdge 3510 “Minnow” brick pairs containing a total of 17 TB of unformatted storage connected to the Brocade switch via 10 2 Gb/s fibre channel links (each brick pair has two 2 Gb/s connections to the Brocade switch).

There are 24 146 GB Seagate Cheetah disks per StorEdge brick pair. Each brick pair has two controllers: a primary and secondary controller. Within each brick pair, we configured four 4+P (Parity) arrays (using a total of 20 disks). This left four disks in each brick pair for hot spares. Each 4+P array represented one Logical Unit Number (LUN). A LUN is a name that allows nodes to identify a group of disks. There were four LUNs per brick pair for a total of 20 LUNs in the testbed. Each LUN is presented as a single block device on the host side.

The Qlogic and Myrinet adapters are on separate PCI-X busses.

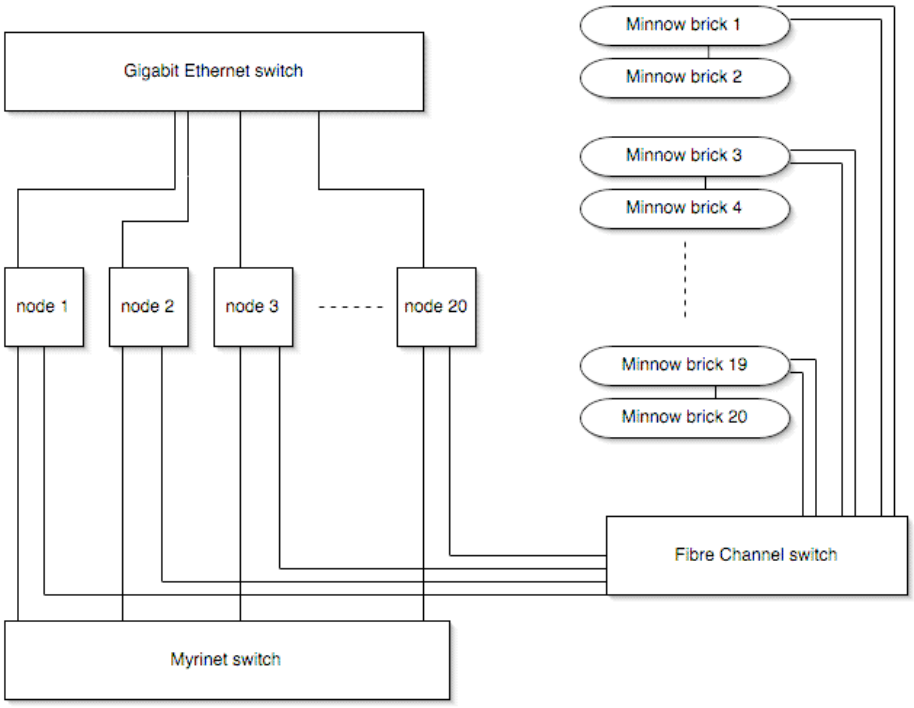


Fig. 1. Parallel File System Testbed

PVFS Testbed Configuration

Since PVFS could not directly manipulate the SAN block devices, we decided not use the SAN disks for PVFS. The only feasible way to use SAN disks with PVFS is to format the SAN disks with an ext3 or other Linux file system and build PVFS on top it. Unfortunately, the common Linux file systems don't support simultaneous multiple node connections to multiple storage targets for parallel I/O. Thus, we installed PVFS over a 35 GB partition on each node to demonstrate parallel I/O capability. PVFS's total storage capacity from 35 GB partitions on the 20 nodes in the testbed was roughly 660 GB unformatted.

We tested PVFS in two configurations. In the first configuration, we attempted to run 20 IODs and 20 clients. On one of the 20 nodes, we also ran the MGR. In the second configuration, we configured PVFS with two nodes running the IOD daemons only, one node running the MGR and a client and 17 nodes running as clients.

With PVFS we could change the stripe size with each I/O, but we used the default stripe size of 64 KB.

GPFS Testbed Configuration

We tested GPFS in two ways: SAN and NSD. In the first configuration, we made use of the direct-attached Sun SAN disks. In SAN mode, each node directly accesses all of the data stored by any of the other nodes through the Qlogic SAN HBA. All 20 nodes acted as servers for themselves. 10 brick pairs were connected to the Brocade switch. Each brick pair consists of 24 disk spindles and two redundant RAID controllers. One brick pair has four (4+ parity) Logical Unit Numbers (LUNs) plus 4 hot spares disks. That adds up to 24 disk spindles. Each LUN is 550 GB in size and appear on the node side as a SCSI disk from the Host Bus Adapter (HBA). Hence, our unformatted GPFS is 4*6*550 GB = 12 TB. GPFS stripe size was set to 512 KB since it is the optimal configuration for the 4+P disk arrays.

. In the second test, we configured two nodes to be the NSD servers. These two nodes served the data to the other 18 nodes over Gigabit Ethernet. The disk and storage configuration were the same as the first configuration.

Lustre Testbed Configuration

Lustre was installed on the same SAN infrastructure as GPFS, with exactly the same storage configurations as GPFS. Hence, the total size was 17 TB unformatted. We have one Lustre MDS server, two OST servers and 17 Lustre clients.

Each OST server was configured to handle 10 LUNs. The Logical Object Volume (LOV) manager, which manages the file striping on the OSTs, was setup to interleave the LUNs and the OSTs. For instance the first OST server handled the odd numbered LUNs and the second OST server handled the even numbered LUNs. A default stripe count was set to 1 so that each LUN would get a portion of the file. The stripe size was set to 1 MB.

Ease of Installation and Administration

Setup and administration are a crucial part of managing any system; therefore we considered the clarity of the documentation and ease configuration and installation of each parallel file system. We also looked at how easy it was to manage

and monitor the system, apply updates to the software and other typical administration tasks that would be performed on the file system.

PVFS - Ease of Installation and Administration

For our testing, we used PVFS version 1.6.2. Our PVFS installation was very smooth and straightforward except for the installation of the pvfs-kernel package. The pvfs-kernel packages need to be compiled with the kernel source matching the running kernel so this took some time to complete. It took time to figure out because we were using a specialized custom kernel. This may not be an issue with an IA-32 kernel. Since it's modular, it's portable to other systems with the same configurations. Once the kernel source was installed and built from the current running configuration, we installed the pvfs-kernel package with little trouble. It took several hours to setup PVFS the first time. With some practice we were able to install it in an hour. It is well documented. Overall, the PVFS installation was simple and painless.

In PVFS, members can be a metadata server (MGR), I/O server (IOD), and client. The PVFS users guide specifies what needs to be done for the MGR, IOD and client nodes. We followed the instructions for installing the metadata server and client for one node, while on the rest of the nodes we followed the instructions for installing the IOD server and client. In our configuration, most nodes are IOD server and client servers, although other configurations are possible.

Based on our experience, it was better to set MGR and IOD daemon priorities higher than normal to ensure stability. When the load on management or IOD machine got too high, the Linux Virtual Machine (VM) starts killing off processes. Nicing prevents MGR and IOD daemons from being killed and ultimately prevented PVFS from crashing. We also periodically monitored the daemons and the status of the mount points. We wrote a Perl script that runs every half an hour to make sure the MGR and each of the IODs were still alive. It also checks to make sure the client nodes were still mounting PVFS. We wrote this script so we could monitor the status of the daemons because occasionally they were unresponsive.

Disks that are part of a RAID can be added to increase the total capacity of the storage target. New disks that are part of a new RAID cannot be added. IOD servers can't be expanded on the fly, since this requires the file systems to be restriped with the new IOD server.

PVFS has several administrative commands that help manage the system. For instance, the `pvfs-ping` command contacts all of the IODs and the MGR.

GPFS - Ease of Installation and Administration

We used GPFS version 2.2 for Linux in this test. After each host identified all 24 LUNS through the HBA, we installed all the necessary RSCT and GPFS RPMs and compiled the GPFS kernel extension. The GPFS kernel extension module was compiled as a module against a running kernel. Since it's modular, it's portable to other systems with the same configurations. The GPFS installation process was straightforward. This process was documented in the GPFS Redbook. The Redbook is a good resource for information and has step-by-step instructions to help beginners to install GPFS quickly. The first time we installed GPFS it took several hours. Now, we can install our 12 TB GPFS file system in about an hour and a half unless we encounter disks or node connectivity problems.

We were able to expand GPFS by growing the file system with new disks. Because GPFS stripes its data and each disk possesses an equal portion of the data, we were able to manually initiate GPFS to rebalance the file system after we added a new disk.

We were also able to easily add new client nodes. We executed a few commands to add the nodes and then started up a daemon called mmfsd. Once the daemon was started and active, we mounted GPFS on these nodes.

GPFS comes with several administrative commands that made it easy to administer. These commands allowed us to check on the status of the daemons and the disks. We also were able to add clients (`mmaddnode`), NSD servers (`mmaddnode`) and replicate the metadata (`mmchconfig`).

Lustre - Ease of Installation and Administration

For this test, we installed Lustre 1.0.4 with one MDS. Two nodes were OST server nodes. The rest of the nodes were clients. The OST nodes were connected to Brocade FC switch that served 20 SAN disks for Lustre. We setup Portals to work over the Gigabit Ethernet network for Lustre connectivity. We opted to use Logical Object Volume manager (LOV) to achieve striping between our OBD targets from a single OST.

Lustre required a custom kernel with multiple Lustre patches. We encountered great challenges installing Lustre, particularly with the kernel patches. We speculate that this might be because we are running a highly optimized and customized kernel. Moreover, the kernel patch modifies other kernel components such as the Qlogic HBA, network driver, SCSI, etc. There is no module that made the installation portable. We spent several days getting Lustre installed. Even after practice, it is still a challenge.

A possibly easier way to install Lustre is to download the pre-patched kernel provided on Lustre's FTP site. These are available for IA-32 RedHat systems with certain standard components. However, if you have specific hardware components, it could be difficult to install. In our case we have Myrinet cards and Qlogic adapters that weren't included in the standard pre-patched kernel. We had great difficulty getting the pre-patched kernel to work with our adapters. CFS offers custom kernel patches and RPMs to customers with support contracts.

Once this obstacle was overcome, the Lustre packages installed easily. Lustre documentation is still in beta mode and was incomplete. For example, some commands that should have patched the kernel failed. The SAN instructions were incomplete.

It is possible to grow a running Lustre file system by adding new disks and new OST servers. This is done by editing the XML configuration and applying this configuration to the new nodes.

Lustre is administered with one command: `lconf`. This made it easy to remember how to manipulate the file system.

Redundancy

System stability and data integrity are important to all the users of any system. System failure is inevitable since there are so many hardware and software pieces involved. Being able to recover from single disk failures, single node (and multiple node) failures as well as daemon and metadata server failures is critical to running a robust system. We examine these aspects of the parallel file systems to see how well they handle different types of failures.

PVFS Redundancy

To avoid data loss, the PVFS users' guide suggests that multiple disks on each IOD can be combined with a hardware RAID for redundancy. The RAID guarantees recovery for single disk failures. Another way to make PVFS more reliable is to dedicate a set of nodes for IOD servers only. This keeps other processes from starving the with the IOD server resources. Keep the packages on these IOD servers as limited as possible and monitor their uptime carefully for as much stability as possible.

Since PVFS stripes data to each disk in the cluster, if one node (or even just the IOD daemon) serving the disk crashes, then PVFS is unavailable to all nodes. For

example, in our case, losing an IOD server is like losing every 10th data block on a disk. The Manager server is another single point of failure of PVFS. If a manager server node crashes, no metadata operations can be performed and as a result, no I/O operation can be completed.

GPFS Redundancy

In SAN mode, each node was a server and accessed all of the disks directly. This meant that a one node failure did not affect the availability of the parallel file system. In fact, multiple nodes can fail and GPFS will still be available to the remaining nodes. We rebooted nodes and were able to put them back into the GPFS pool.

Using a RAID configuration keeps GPFS safe from single disk failures. For multiple disk failures, both SAN and NSD modes have the ability to replicate the data and the metadata. To replicate data, GPFS performs the I/O operations twice into different disks and/or failure groups if it is configured. Although this doubles the space used in the file system, it provides redundancy if you don't have RAID or other spares. We don't use this option in our system since our SAN storage is RAID-5 and each brick pair has 4 hot spare disk spindles, which automatically rebuild the volume to the hot spare disk when a disk fails.

A failure group is a collection of disks that are defined to handle disk failures. For instance, a failure group setup as a RAID can include disks that reside on different controllers. This allows a controller to fail without affecting access to the file system.

In NSD mode, every I/O operation goes through the primary NSD server. If this server fails, the secondary NSD server takes over. This provides redundancy in case of a single server failure.

Lustre Redundancy

Lustre offers redundancy for both MDS and OST daemons. When a Lustre client discovers that the primary MDS is unreachable, the client queries its LDAP server and asks for a secondary MDS to be started on a different host. The secondary MDS replaces the primary MDS and allows the Lustre client to complete its I/O. When an OST server fails or times out, a standby OST server will take over the original OST server to continue access to the shared OBD. This can only happen with a redundant path to the storage targets. If there are no available standby OST servers, Lustre will report an I/O error and prevent any additional write operations to that OST

server. Lustre will automatically direct a write operation to another OST server. In the case of a read operation, failure will occur if the client requests data from a failed OST server. RAID can be used to help with single disk failures.

Performance

Performance is a key component in analyzing parallel file systems. If simple file sharing between nodes is the goal, NFS is an easier file system to setup and manage. To find each parallel file system's performance characteristics, we had all clients read and write to a unique portion of one file.

To benchmark each parallel file system we used the Interleaved-Or-Random (IOR) program version 2.7.4 from Lawrence Livermore National Laboratory (LLNL). IOR is open source code and freely available from LLNL public ftp site. Its main goal is to test parallel file system performance using various network interfaces and access patterns. IOR relies on MPI to synchronize its I/O operations. We ran IOR over the Myrinet network with MPICH-GM-1.2.5..10[6].

We did two different comparisons of the file systems: in the first comparison, we setup the file systems that could have servers and clients on the same node. For this test, we used GPFS in SAN mode (all nodes accessing the disks directly over the fibre channel) and PVFS (all nodes accessing the disks directly locally on each node and over the GE). We did not include Lustre in this test since Lustre could not have clients and servers on the same nodes. For this comparison, we setup all 20 nodes to act as clients and servers.

In the second comparison, we setup all of the file systems with two servers and 18 clients. For Lustre we used 17 clients since one server had to be dedicated to the MDS. We used the GPFS NSD mode for this test.

For each comparison we performed two different IOR tests. For the first IOR test, each client wrote a 1 GB file to the file system. During this test, we varied the block size from 16 KB to 4 MB. We did not use block sizes larger than 4 MB because performance started to plateau. In the second IOR test, we took the optimum block size from the first test at and wrote files with this block size varying from 1 MB to 8 GB in size.

Our tests were done using out-of-the-box parameters, meaning we didn't tune each file system specifically to produce the optimum performance. We intended to test the file systems as a novice.

Storage Hardware Characteristics

The local IBM disks in each node that were used for the PVFS tests had a disk cache of 8 MB each. Using Bonnie[7], we tested a write speed of 103 MB/s to the local formatted IBM SCSI disk. We achieved a read speed of 344 MB/s to this same disk. The local SCSI disk cache was 8 MB.

For the GPFS and Lustre tests, we used the Minnow bricks in the SAN infrastructure. Each Minnow brick had a maximum read speed of 192 MB/s and a write speed of 172 MB/s. The aggregate maximum peak read speed was 1.9 GB/s and the write speed was 1.7 GB/s.

The first Minnow brick in the pair had two 2 Gb/s connections to the Brocade switch. The second Minnow brick in the pair had one 2 Gb/s connection to the first Minnow brick. Each "minnow" brick had a disk cache of 1 GB. There were two controllers in the first brick. Both controllers managed all I/O to the brick pair. The raw I/O rate from each brick was 2 Gb/s and 4 Gb/s for the brick pair.

Each brick pair had a command queue limit of 100. The command queue is a separate I/O request. Therefore, for the five bricks pairs in our test, we had a total of 500 initiators. To figure out the queue depth (number of outstanding I/Os) each node could request, we divided the total number of initiators by the total number of nodes (500/20 = 25). We set the queue depth to be to 24 to avoid blocking from the controllers in the disk.

Since each brick has 1 GB disk cache, any I/O 1 GB or less (per brick) fits into the local disk cache and shows exceptionally good performance. This is because as soon as the data is in the cache, the controller signals the HBA that the I/O is completed. This doesn't measure the actual effective I/O performance. This caching effect is seen in the graphs for GPFS and Lustre for I/O 1 GB or less per node. Once the I/O exceeds 1 GB, the controller flushes the cache and forces the data to be written to the disks. This is indicative of the actual I/O rate.

On the host side, each Qlogic adapter was connected to the Brocade switch. The channel rate for the connection for each Qlogic adapter was 2 Gb/s. This was limited within the node by the rate of the PCI-X bus which was 400 MB/s.

Maximum Theoretical Performance - PVFS

For both comparisons, we used the local disks on each node. The theoretical performance for these disks was 103 MB/s for write and 344 MB/s for read. Serving this data to the other nodes with over the Gigabit Ethernet limited the overall per-

formance to 125 MB/s. Including the TCP/IP overhead, the maximum theoretical performance over the Gigabit Ethernet was limited to 115 MB/s.

For the two tests in the first comparison, the aggregate maximum theoretical performance for PVFS was 115 MB/s * 12 nodes = 1.38 GB/s. We tried to run with 20 servers and 20 clients, but were not able to get the I/Os to successfully finish. We were able to run successfully with 20 servers and 12 clients so we calculated the maximum theoretical performance based on 12 clients.

For both tests in the second comparison, the aggregate maximum theoretical performance was 115 MB/s * 2 nodes = 230 MB/s because each of the two servers had one Gigabit Ethernet connection and all the data transferred from both of these servers since the data was striped.

Maximum Theoretical Performance - GPFS

In the first comparison, we used the SAN mode. In this mode, although data is stored across all the disks, each node can transfer the entire data directly over its local fibre channel adapter. The aggregate maximum theoretical performance for GPFS was 250 MB/s (speed of the fibre channel link) * 20 nodes = 5 GB/s.

In the second comparison, we used the NSD mode. In this mode, the clients can retrieve data from two NSD servers. Since these servers were connected via one Gigabit Ethernet connection each, the aggregate optimal performance for GPFS is 115 MB/s * 2 nodes = 230 MB/s.

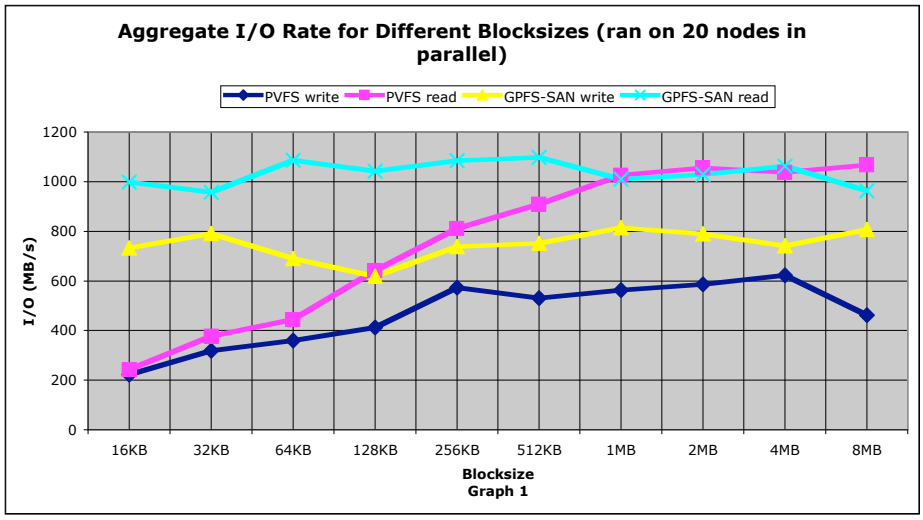
Maximum Theoretical Performance - Lustre

We didn't include Lustre in the first comparison because Lustre cannot have an OST server running on the same node as a client. To accomplish a test with 20 OST servers and 20 clients, we needed 40 nodes total, which our resources did not permit.

For both tests in the second comparison, the aggregate maximum theoretical performance was 115 MB/s. This was because we set the stripe count to be one in the XML configuration file. 10 LUNs were setup to one OST server and 10 additional LUNs were setup to the other OST server. Only one OST server served data to the requesting clients.

Comparison 1 **(All nodes clients and servers)**

First IOR test



GPFS	`mpirun -np 20 /testgpfs/IOR -b 1G -t $_BLOCK -e -a POSIX -g –u`
PVFS	`mpirun -np 12 /mnt/pvfs/IOR -b 1G -t $_BLOCK -e -a POSIX -g –u`
Legend	$_BLOCK = we vary this block from 16k, 32k, 64k, until 8M

Table 2. Graph 1 IOR command

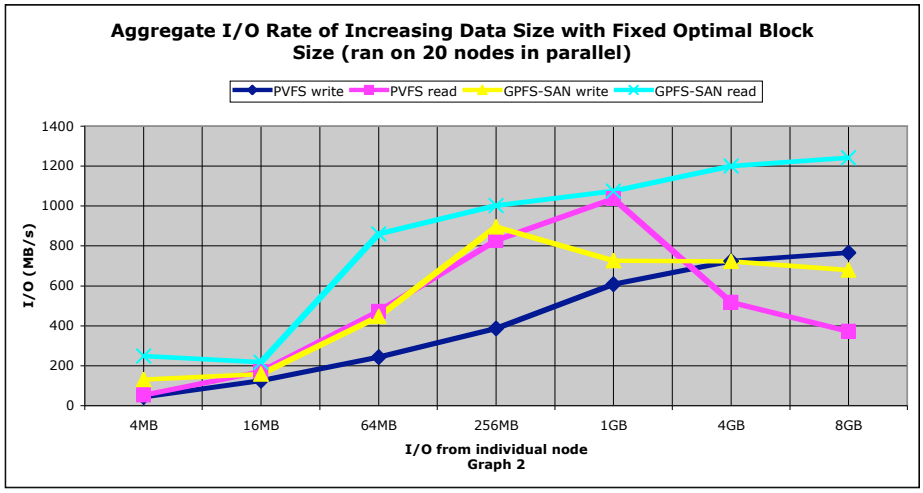
PVFS Performance – 20 IODs, 12 clients

In the best case, we achieved 1.1 GB/s read performance in the 8 MB block size and 600 MB/s write performance for the 4 MB block size.

GPFS Performance – 20 servers, 20 clients

In the best case, we achieved 1.1 GB/s read performance for several different block sizes and 800 MB/s write performance for various block sizes. The performance is consistent across all tested block sizes.

Second IOR test



GPFS	`mpirun -np 20 /testgpfs/IOR -b $_TRANSFER -t $_OPTIMAL -e -a POSIX -g –u`
PVFS	`mpirun -np 12 /mnt/pvfs/IOR -b $_TRANSFER -t $_OPTIMAL -e -a POSIX -g –u`
Legend	$_OPTIMAL = optimal block size for the particular file system $_TRANSFER = we vary this value from 4M, 16M, 64M, ... 8G

Table 3. Graph 2 IOR command

PVFS Performance – 20 IOD servers, 12 clients

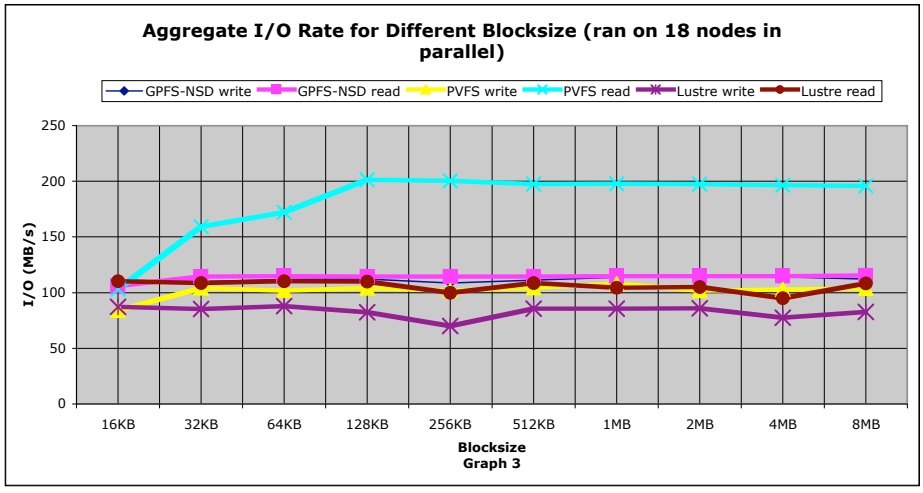
As the data size increased, PVFS scaled almost linearly for write operations. For reads, however, performance increased almost linearly until the 1 GB data size mark per node. This particular behavior is also noted in UCLA's Tajendra Singh paper on performance analysis with 17 nodes and much smaller I/O[8].

GPFS Performance – 20 servers, 20 clients

Similar to the first IOR test, GPFS read and write performance increased almost linearly as the data size increased.

Comparison 2 (**17-18 clients and 2 servers)**

First IOR test



GPFS	`mpirun -np 20 /testgpfs/IOR -b 1G -t $_BLOCK -e -a POSIX -g –u`
PVFS	`mpirun -np 12 /mnt/pvfs/IOR -b 1G -t $_BLOCK -e -a POSIX -g –u`

Lustre	`mpirun -np 20 /mnt/lustre/IOR -b 1G -t $_BLOCK –e -a POSIX -g –u`
Legend	$_BLOCK = we vary this block from 16k, 32k, 64k, until 8M

Table 4. Graph 3 IOR command

PVFS Performance – 2 IOD servers, 12 clients

In the best case, we achieved 200 MB/s read performance. This performance was consistent for all block sizes over 128 KB. This is due to the two Gigabit Ethernet adapters in each IOD serving out the data. The write performance is consistent around 100 MB/s for all block sizes from 32 KB.

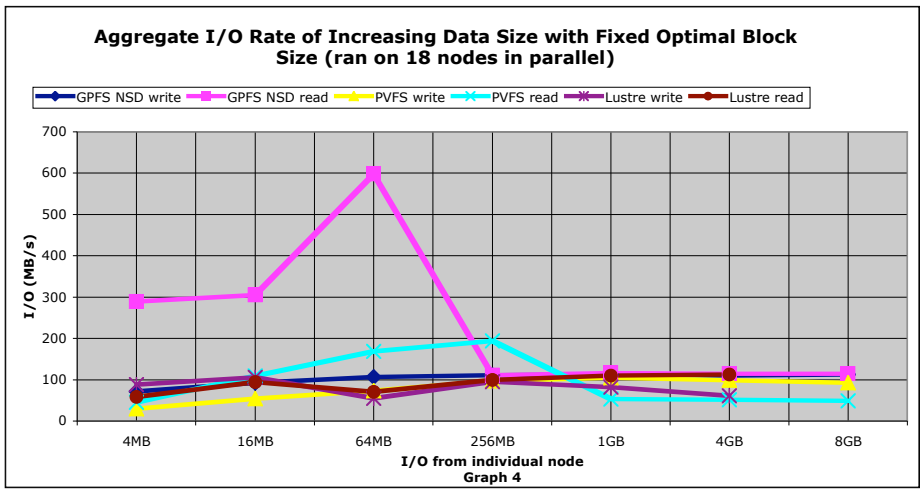
GPFS Performance – 2 NSD servers, 18 clients

In the best case, we achieved around 112 MB/s read and write performance for several different block sizes. The performance is consistent for all block sizes we tested.

Lustre Performance – 2 OST servers, 17 clients, 1 MDS

The maximum write performance for Lustre was 88 MB/s. The read performance was 110 MB/s. The read and write performances were fairly consistent for all block sizes.

Second IOR test



GPFS	`mpirun -np 20 /testgpfs/IOR -b $_TRANSFER -t $_OPTIMAL -e -a POSIX -g –u`
PVFS	mpirun –np 12 /mnt/pvfs/IOR -b $_TRANSFER -t $_OPTIMAL -e -a POSIX -g –u
Lustre	mpirun -np 20 /mnt/lustre/IOR -b $_TRANSFER -t $_OPTIMAL -e -a POSIX -g –u
Legend	$_OPTIMAL = optimal block size for the particular file system $_TRANSFER = we vary this value from 4M, 16M, 64M, ... 8G

Table 5. Graph 4 IOR command

PVFS Performance – 2 IOD servers, 12 clients

In the best case, we achieved 104 MB/s write performance. This performance was consistent for the data sizes 128 MB and over. The read performance increased to 200 MB/s up to the 256 MB data size per node (3 GB aggregate data size).

GPFS Performance – 2 NSD servers, 18 clients

The write performance is consistent for all data sizes around 112 MB/s aggregate. The read performance escalates sharply until 1 GB and then stabilizes around 115 MB/s. We believe the escalation is due to the 1 GB cache on a Minnow brick pair. Read requests for data sizes smaller than 1 GB reside in the cache. Requests for larger amounts of data are striped over multiple LUNs (and brick pairs).

Lustre Performance – 2 OST servers, 17 clients, 1 MDS

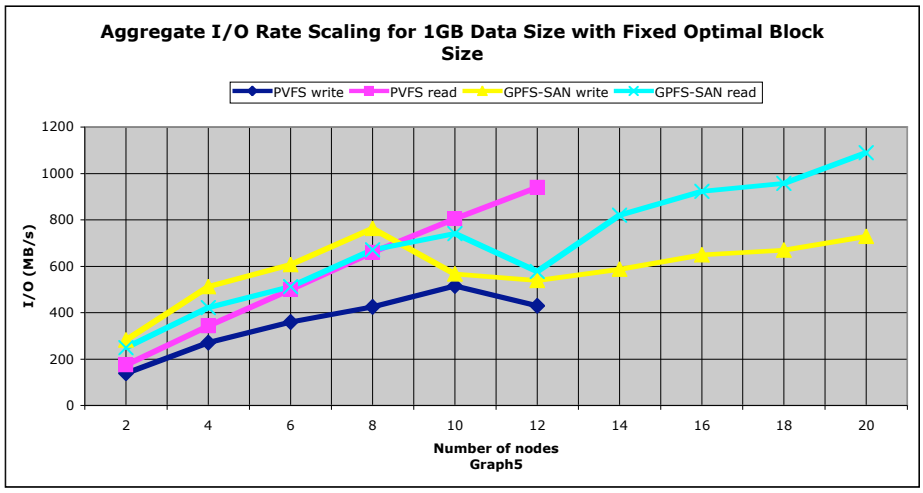
The Lustre performance in the second IOR test was remarkably consistent with the performance from the first IOR test. The maximum write performance for Lustre was 88 MB/s. The read performance was 110 MB/s. The read and write performances were fairly consistent for all data sizes.

Scalability

Different file systems support different numbers of clients. They also perform differently under different kinds of load. As the number of nodes reading or writing to a particular file system changes, the performance of the file system can increase, stay the same or decrease. We wanted to see how each parallel file system would perform for various numbers of nodes. Using the results from the Performance test, we selected the block size that performed the best for each file system. We used this block size and varied the number of nodes from two to 20 that read and wrote the 1 GB file. Each test calculated the total I/O operation time including opening the file, writing/reading from the file and closing the file.

We used IOR and changed the number of processors in the MPIRUN command to perform the scaling test.

Comparison 1 **(All nodes clients and servers)**



GPFS	`mpirun -np $_NUMNODE /testgpfs/IOR -b 1G -t $_OPTIMAL -e -a POSIX -g –u`
PVFS	`mpirun -np $_NUMNODE /mnt/pvfs/IOR -b 1G -t $_OPTIMAL -e -a POSIX -g -u`
Leg-end	$_OPTIMAL = optimal block size for the particular file system $_NUMNODE = we vary the number of node from 2, 4, 6, 8...20

Table 6. Graph 5 IOR command

PVFS Scalability – 20 IOD servers, 12 clients

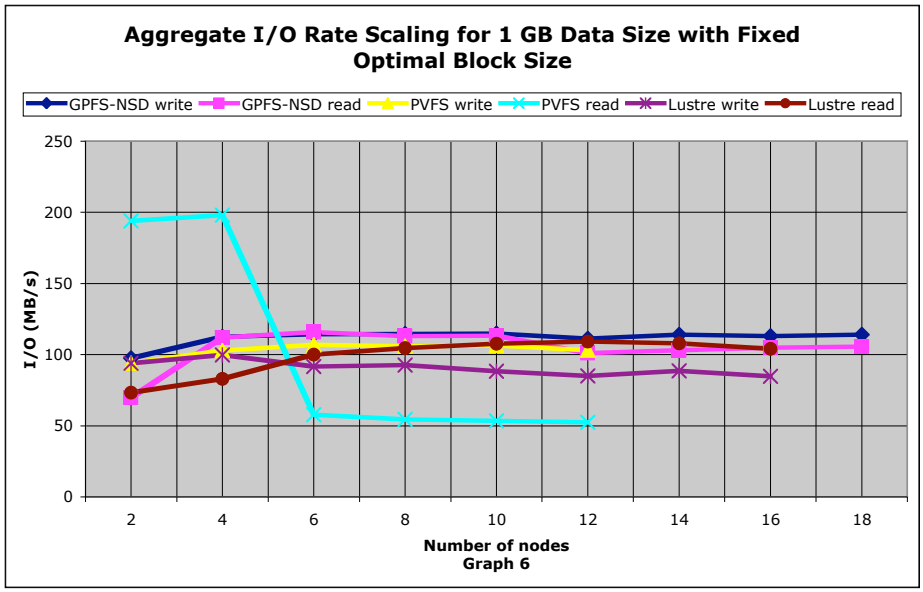
As more nodes were added up to 12 nodes, PVFS scaled almost linearly for reading operations. The best performance was 929 MB/s. The write performance scaled linearly up to 10 nodes (and 500 MB/s aggregate) and then decreased to 429 MB/s aggregate.

GPFS Scalability – 20 servers, 20 clients

The GPFS read performance scales linearly up to 20 nodes and 1.09 GB/s aggregate. There is one data point out of line with this scaling and we consider it an anomaly. The write performance scales as well up to 729 MB/s.

The version of GPFS that we used supported 512 compute nodes for both SAN and NSD modes.

Comparison 2 **(17-18 clients and 2 servers)**



GPFS	`mpirun -np $_NUMNODE /testgpfs/IOR -b 1G -t $_OPTIMAL -e -a POSIX -g –u`
PVFS	`mpirun -np $_NUMNODE /mnt/pvfs/IOR -b 1G -t $_OPTIMAL -e -a POSIX -g –u`
Lustre	`mpirun -np $_NUMNODE /mnt/lustre/IOR -b 1G -t $_OPTIMAL -e -a POSIX -g –u`
Leg-end	$_OPTIMAL = optimal block size for the particular file system $_NUMNODE = we vary the number of node from 2, 4, 6, 8, ...20

Table 7. Graph 6 IOR command

PVFS Scalability – 2 IOD servers, 12 clients

In the best case, we achieved 200 MB/s read performance. We achieved this for the two and four node tests. From 6 to 12 nodes, the performance was consistent at 50 MB/s. The write performance was consistent at 100 MB/s for all numbers of nodes.

GPFS Scalability – 2 NSD servers, 18 clients

The read and write performance is consistent for all number of nodes. The performance is around 114 MB/s for both.

Lustre Scalability – 2 OST servers, 17 clients, 1 MDS

The write performance of 100 MB/s is consistent for all number of nodes. The read performance is consistent around 110 MB/s for all number of nodes.

According to the Lustre documentation, Lustre can handle thousands and tens of thousands of clients. We did not have the opportunity to test this (though we would be happy to test this if a reader would donate equipment to us).

Special Features

Each parallel file system possesses unique features exclusive to that specific file systems. These additional functionalities can make a specific file system especially attractive depending on your requirements. There are many features special to each of the three parallel file systems. We include a few features of particular interest for each parallel file system.

PVFS Special Features

PVFS is an open source file system that is portable across many different platforms. PVFS is available for most Linux flavors including both IA-32 and IA-64.

Users can easily modify the file system to their needs. PVFS is compatible with ROMIO, an MPI-IO implementation that directly performs I/O bypassing the kernel. Programs can be written specifically for PVFS by including PVFS headers and libraries when compiling. Applications implemented this way outperform other applications that access PVFS through UNIX kernel because the UNIX kernel has sizable overhead when performing I/O. Moreover, PVFS-compiled applications have more control over stripe size, stride size, and offset, making the I/O more efficient. This gives these applications about 10% better performance.

In a demonstration with the California Institute of Technology, we mounted PVFS successfully to SDSC over the Wide Area Network (WAN).

GPFS Special Features

GPFS has native support for a SAN infrastructure. Metadata can travel over the Gigabit Ethernet but I/O transfers occur directly over the 2 Gb/s fibre channel link. This dedicated I/O path allows for better performance.

At the Supercomputing 2003 conference in a technology demonstration, SDSC and NCSA mounted their respective GPFS's to each other's sites and to the show room floor cluster in Phoenix, AZ. The IA-64 cluster on the show floor ran the National Partnership for Advanced Computational Infrastructure (NPACI) Visualization Toolkit to render earthquake simulation data from the Southern California Earthquake Center. The NPACI Visualization Toolkit[9] read the remote data from the "locally-mounted" GPFS (in San Diego on an IA-64 cluster). This demonstration utilized over 92% of the available 10 Gb/s bandwidth.

Another feature of GPFS is support for the Linux and AIX operating systems to access the same GPFS simultaneously. With this feature, data can be shared between two heterogeneous clusters. This reduces the amount of source code and data that has to be copied between clusters by application scientists using both systems. For instance, scientists can compute on one cluster and render on another without copying the output files from the compute system to the rendering system.

Lustre Special Features

Lustre was used by a team of ASCI and other partner sites at Supercomputing 2003 that demonstrated Lustre over the WAN. Lustre was exported and mounted on IA-32 Linux clusters across the country.

Most parallel file systems lock specific bytes to prevent file corruption. Lustre uses an intent-based locking system to reduce contention. When a client requests a range of bytes from a file, it specifies to the MDS whether it plans to read or write to this range of bytes. If it plans to read from the file, then this intent is recorded and the MDS allows another client to write to these bytes

Comparison

To summarize, we listed the evaluation criteria for each of the three parallel file systems. We examined and noted our experiences with the ease of installation and administration, redundancy, performance, scalability and special features for each parallel file system. We found that each parallel file system had strengths.

	PVFS	GPFS	Lustre
Ease of installation, administration	Mostly well documented	IBM Redbook	Difficult
Redundancy	Limited	Replicate data, replicate metadata, failure groups	Replicate OST, failover MDS
Performance	GE speed	GE and 1 GB/s over fibre channel speeds	GE speed
Scalability	Limited to 12 clients	IBM guarantees and tests up to 512 nodes	Tens of thousands of clients
Special features	WAN, ROMIO	SAN, WAN, AIX-Linux interoperability	WAN, intent-based locking

Table 8. Summary of features.

Conclusions

PVFS, GPFS and Lustre are all feature-rich parallel file systems with different characteristics. PVFS was easy to install and administer and gave respectable performance. It has the most moderate hardware requirements and is open source. In both SAN and NSD modes, GPFS performed the best. It was also easy to install and had numerous redundancy and special features. Lustre performed well and had various redundancy and other special features. It is also open source. All three of these

parallel file systems are under continual development and will continue to evolve increasing functionality and performance.

Acknowledgements

For information and configuration of the parallel file systems, we wish to thank several individuals. From Argonne National Laboratories, we wish to thank Rob Ross. We are grateful for the help we received from Roger Haskin and Puneet Chaudhary from IBM. We acknowledge Andreas Dilger and Phil Schwan from CFS for their help with Lustre. We wish to thank Don Thorp, Haisong Cai and Christopher Jordan of SDSC for their assistance with this paper.

References

[1] Enzo –AMR Cosmological Simulation Code. (http://cosmos.ucsd.edu/enzo/)

[2]Ross, Robert B. et al. - Using the Parallel Virtual File System. (http://www.parl.clemson.edu/pvfs/user-guide.html)

[3] Hochstetler, Stephen et al. IBM redbook – Linux Clustering with CSM and GPFS. (http://publibb.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246601.html?Open).

[4]Braam, Peter J. – The Lustre Storage Architecture. (http://www.lustre.org/docs/lustre.pdf)

[5]Brightwell, Ron – Sandia Portals Project. (http://www.cs.sandia.gov/~ktpedre/portals/index.html)

[6] MPICH-GM – Message Passing Interface over GM drivers (http://www.myri.com/)

[7] Bonnie - (http://www.textuality.com/bonnie/)

[8]Singh, Tajendra – Experiment with PVFS on a Cluster of 17 Nodes. (http://www.ats.ucla.edu/at/beowulf/pvfs/atsdocument.htm)

[9] Cutchin, Steve – NPACI Visualization Toolkit (http://vistools.npaci.edu/)