Evaluating the Shared Root File System Approach for Diskless High-Performance Computing Systems*

Christian Engelmann, Hong Ong, and Stephen L. Scott

Computer Science and Mathematics Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
{engelmannc|hongong|scottsl}@ornl.gov

Abstract. Diskless high-performance computing (HPC) systems utilizing networked storage have become popular in the last several years. Removing disk drives significantly increases compute node reliability as they are known to be a major source of failures. Furthermore, networked storage solutions utilizing parallel I/O and replication are able to provide increased scalability and availability. Reducing a compute node to processor(s), memory and network interface(s) greatly reduces its physical size, which in turn allows for large-scale dense HPC solutions. However, one major obstacle is the requirement by certain operating systems (OSs), such as Linux, for a root file system. While one solution is to remove this requirement from the OS, another is to share the root file system over the networked storage. This paper evaluates three networked file system solutions, NFSv4, Lustre and PVFS2, with respect to their performance, scalability, and availability features for servicing a common root file system in a diskless HPC configuration. Our findings indicate that Lustre is a viable solution as it meets both, scaling and performance requirements. However, certain availability issues regarding single points of failure and control need to be considered.

1 Introduction

Scalability and availability are key issues that drastically affect system performance and efficiency in large-scale high-performance computing (HPC) systems. Many factors, such as shared networks, centralized servers, and compute node design, can potentially limit system scalability and eventually affect system performance. Other aspects, such as individual system component reliability as well as single points of failure and control, can potentially limit system availability and eventually affect system efficiency.

In view of recent HPC system architectures, a common design feature among them is the adoption of diskless compute nodes to exploit high performance

* This research is sponsored by the Mathematical, Information, and Computational Sciences Division; Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725.

networked storage. Removing disk drives significantly increases compute node reliability as they are known to be a major source of failures. Furthermore, networked storage solutions utilizing parallel I/O and replication are able to offer increased scalability and availability.

The diskless HPC system approach in essence extends traditional cluster computing mechanisms to MPP systems by providing an illusion of a symmetric multiprocessing (SMP) system without actually enforcing strong SMP semantics at all parts of the operating system (OS). For example, the most prominent feature is a system-wide unified networked storage and file system with concurrent and shared access.

However, diskless HPC systems have limitations. One major obstacle is the requirement by certain OSs, such as Linux, for a common root file system. While one solution is to remove this requirement from the OS and provide access to a networked shared hierarchical storage and file system for applications only, another is to share the root file system over the networked storage. This paper focuses on the latter approach within diskless HPC systems. Three networked file systems, Network File System version 4 (NFSv4) [1], Lustre [2, 3] and Parallel Virtual Filesystem version 2 (PVFS2) [4], are evaluated for their performance, scalability and availability when used for serving a common root file system.

This paper is organized as follows. First, we provide a background discussion on networked file system technologies in HPC. Second, we detail the shared root file system approach for diskless HPC systems. Third, we describe our testbed hardware and software environment. Fourth, we present performance and scalability test results for the three networked file system solutions. Fifth, we discuss certain availability deficiencies regarding single points of failure and control, and show how they can be addressed. Finally, this paper concludes with a summary of the presented research.

2 Background and Motivation

The concept of a diskless system configuration utilizing a common root file system in moderately scaled HPC systems has existed for a number of years [5]. Generally, diskless HPC systems improve system manageability by providing a central facility for administration tasks, such as configuration and system backup and restore. For instance, system upgrades and configuration rollbacks can be easily achieved by maintaining multiple versions of the OS kernel and configuration stored in a common location. Furthermore, software packages can be easily deployed and removed by simply maintaining the shared root file system.

A diskless HPC system typically employs a read-only NFS-based root file system. While this solution provides some benefits, it is severely limited by the scalability of NFS servers [6]. Although the new NFSv4 [1] has shown the ability to combine multiple file system operations into one request, it is still based on a high-overhead protocol.

Besides NFS, diskless HPC systems today also utilize high performance parallel storage and file system solutions, such as PVFS2, GPFS [7], TeraGrid [8, 9],

or Lustre, to provide a scalable I/O subsystem. These solutions provide tremendous performance gains over NFS. Nonetheless, many diskless HPC distributions continue to offer an NFS-based root file system within system partitions or across the entire system. Parallel file systems are solely used for application data and checkpointing.

High performance parallel file systems have not been used to provide root file systems in the past due to perceived poor software stability and reliability. For example, a fallacious perception is that these parallel file systems rely on a complex stack of OS kernel modules and system utilities that is not part of a Linux distribution. On the contrary, these parallel file systems are sufficiently mature as to provide reliable and highly available storage without requiring cumbersome OS kernel installations. As such, this makes their use for root file systems feasible. Our work leverages high performance parallel file systems to provide a common root file system in a diskless system configuration.

In the following, we provide a brief overview of two popular approaches for implementing parallel file systems: object-based and block-based.

Object-based: In an object-based file system implementation, each file or directory is treated as an object with some predefined attributes. Each attribute can be assigned a value such as file type, file location, data stripes, ownership, and permissions. An object storage device in turn allows users to specify where to store the data blocks allocated to a file through a meta-data server (MDS) and object storage targets (OSTs). A rich attribute set also allows users to specify how many targets to stripe onto as well as the level of redundancy.

Panasas [10] and Lustre are two examples of object-based file system implementations. Panasas is a proprietary solution and has implemented the concept of object-based file systems entirely in hardware. Using a lightweight client on Linux, Panasas is able to provide highly scalable multi-protocol file servers, and have implemented per-file level RAID. Lustre on the other hand is an open source software solution. It runs on commodity hardware and uses MDSs for file system metadata, *i.e.*, `inodes`. Lustre's design provides efficient division of labor between computing and storage resources. Replicated, failover MDSs maintain a transactional record of high-level file and file system changes. One or many OSTs are responsible for actual file system I/O and for interfacing with physical storage devices. Consequently, Lustre achieves high I/O throughput through enabling file operations to bypass the MDS completely and fully utilize the data paths to all OSTs within a cluster. Additionally, Lustre supports strong file and metadata locking semantics to maintain total coherency of the file systems even under a high volume of concurrent accesses.

Block-based: In a block-based file system implementation, data blocks are striped, in parallel, across multiple storage devices on multiple storage servers. This is similar to network link aggregation in which the I/O is spread across several network connections in parallel. Each packet traverses a different link path. File systems using this approach place data blocks of files on more than one server and more than one physical storage device. As the number of available servers and storage devices increases, throughput can potentially increase as well

if sufficient network capacity is available to clients. When a client requests I/O operations, each sequential data block request can potentially go to (or come from) an entirely different server or storage device. Consequently, there is a linear increase in performance up to the total capacity of the network.

PVFS2 is an example of a block-based file system implementation. It uses a lightweight server daemon process to provide simultaneous access to storage devices from hundreds to thousands of clients. Each node in the cluster can be a server, a client, or both. Since storage servers can also be clients, PVFS2 supports striping data across all available storage devices in a cluster.

In a nutshell, an object-based file system achieves scalability and performance by striping data across dedicated storage devices, while a block-based file system achieves the same benefit by aggregating data across available storage devices.

3 Architecture

The shared root file system approach for large-scale diskless HPC systems, such as with 100,000 or more compute nodes, can only be achieved in a hierarchical fashion to avoid exceeding the resources of meta data and storage servers. There are generally three approaches: (1) partition-wide sharing across compute nodes, (2) system-wide sharing across I/O service nodes, and (3) a combination of both prior approaches. In all three, the same root file system is mounted over the network by each node, while configuration specific directories, such as `/etc`, are mounted over the network separately per node.

The first one does not offer a system-wide shared root file system. Instead, it partitions the system into small enough parts, such that each partition has its own shared root file system offered by a dedicated service node within the partition. This approach requires a larger system management overhead as the compute node root file system needs to be maintained within each partition. However, today's HPC systems often employ I/O service nodes for providing a hierarchical approach for application data and checkpointing storage. Each partition has its own I/O node connected to a system-wide shared file system. The functionality of I/O nodes may be extended to serve a partition-wide shared root file system.

The second approach is similar to providing application data and checkpointing storage. The root file system for the compute notes is shared across the entire system using I/O nodes within system partitions for I/O forwarding. The system management overhead is minimized as the compute node root file system is maintained system-wide. However, the I/O forwarding is only alleviating certain resource constraints, while the system-wide shared file system handles all read and write requests to the root file system.

The third approach combines both prior ones by using the I/O nodes as a write through cache. Since OS files are more read than written by compute nodes, this solution may offer a scalable approach by significantly reducing the read request load from a system-wide shared root file system. The system management

overhead is minimized as well, as the compute node root file system is maintained system-wide.

This paper focuses on evaluating the performance, scalability and availability of networked file systems serving a common root file system for (1) compute nodes within a partition and (2) I/O nodes across the entire system. In both cases, the number of client nodes for the common root file system is relatively small compared to the overall compute node count.

4 Testbed

This section describes the hardware and software environment used in our performance and scalability tests of shared root file system solutions. Since setup and administration are a crucial part of system management and maintenance, we are also considering documentation, configuration, and installation of each tested file system. Particularly, we investigate the ease-of-use regarding installing and upgrading as well as at monitoring runtime activity.

4.1 Hardware

The testbed system is a cluster of 30 nodes. Each node is equipped with a 2.6GHz Intel Xeon processor, 1 Gbyte RAM, and a 80 Gbytes Western Digital IDE hard disk operating at 7200 RPM. The nodes are interconnected through a 100 Mbits/s Fast Ethernet switch.

4.2 Software

The system is configured with the Debian GNU/Linux version 3.1 OS distribution using the Linux kernel version 2.6.15.6. The kernel is configured with NFSv4 file system support. In addition to NFSv4, the Lustre 1.4.6.4 and PVFS2 version 1.4.0 are installed. The NFSv4 file system uses a single storage server, but is configured as a multi-threaded service. In contrast, the Lustre and PVFS2 file systems are configured with 1 metadata server and 3 storage servers. All three file systems are evaluated with up to 25 clients.

NFSv4 comes with the Linux OS and can be easily configured. Software upgrades are fairly simple as NFSv4 is maintained via the OS distribution.

The installation of PVFS2 is straightforward. It requires compiling the `pvfs-kernel` package against the currently running kernel source. The PVFS2 architecture consists of clients, I/O servers, and a metadata server (MDS). The PVFS2 users guide specifies the necessary installation and configuration steps for each component. When upgrading the OS kernel, the `pvfs-kernel` package needs to be recompiled.

Lustre utilizes a highly optimized and customized OS kernel as its entire architecture resides in kernel-space. This can easily result in great challenges, particularly with the required kernel patches. The availability of pre-patched kernels is limited to systems with certain OS distributions (Red Hat Enterprise

Linux 4/5 or SUSE Linux Enterprise Server 9/10). The Lustre architecture consists of clients, object storage target (OST) servers and a MDS. The OST servers are configured in our system as one single Logical Object Volume (LOV). Lustre is administered using only one command, `lconf`, which configures, starts and stops each Lustre component using command line switches and a common configuration file. However, some configuration details of Lustre components, such as IP ports, cannot be changed despite offered configuration options. When upgrading the OS kernel, the Lustre kernel patches need to be re-applied.

5 Evaluation

A series of tests were performed to determine the effect of using a NFSv4-, PVFS2- or Lustre-based shared root file system. We used the Interleaved-Or-Random (IOR) benchmark [11], which is a parallel program that performs concurrent writes and reads to/from a file using the POSIX and MPI-IO API, and reports the achieved I/O bandwidth. The tests were performed using out-of-the-box parameters, *i.e.*, we did not tune each file system specifically to produce the optimum performance. Our intent was to obtain baseline performance results for each file system.

5.1 Performance

Figures 1-6 show the collected IOR performance results. For each test, we performed the IOR read and write operations, decomposing a 128 Mbytes file to the file system. Additionally, we varied the transfer block size from 8 Kbytes to 256 Kbytes. The reason for using a small transfer block size is to synthesize the read/write of OS files, which often are small.

Using 8 Kbytes transfer block size, the aggregate write bandwidth for single client (indicated as 1 PE in Figures) is approximately 90 Mbytes/s, 12 Mbytes/s, and 51 MB/s for NFSv4, PVFS2, and Lustre, respectively. On the other hand, the aggregate read bandwidth is 114 Mbytes/s, 14 Mbytes/s and 112 Mbytes/s for NFSv4, PVFS2, and Lustre. Varying the transfer block size does not significantly affect the overall read/write bandwidth for all file systems. It is interesting to note that the Lustre read bandwidth for one node increases to 802 Mbytes/s as the transfer block size reaches 128 Kbytes. This indicates that Lustre is able to utilize the network more efficiently than NFSv4 and PVFS2 when the transfer buffer is large.

Serving the test data to 25 clients over the Fast Ethernet with 8 Kbytes transfer block attains a maximum write bandwidth of approximately 28 Mbytes/s, 34 Mbytes/s, and 30 Mbytes/s for NFSv4, PVFS2, and Lustre, respectively. The overall attainable read bandwidth is 19 Mbytes/s, 4 Mbytes/s, and 246 Mbytes/s for NFSv4, PVFS2, and Lustre, respectively.

Read operations are more crucial and frequent than write operations as OS files are often write-protected. As such, the performance results show that Lustre is suitable for serving a shared root file system especially when the number of

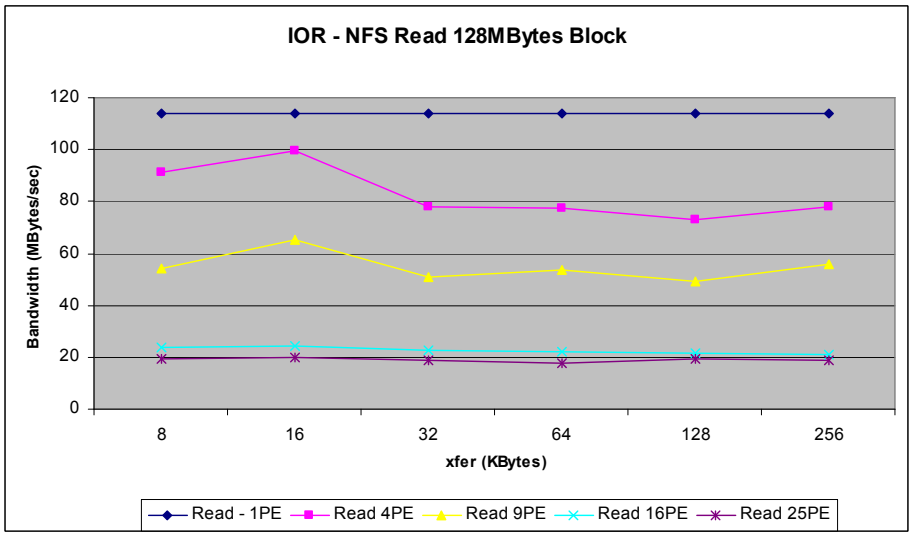

Fig. 1. NFSv4 Performance Tests: Read

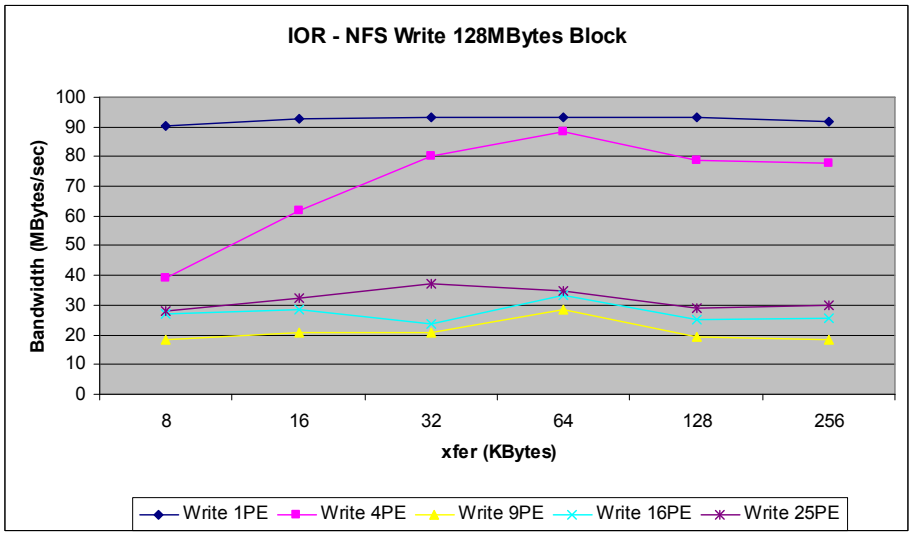

Fig. 2. NFSv4 Performance Tests: Write

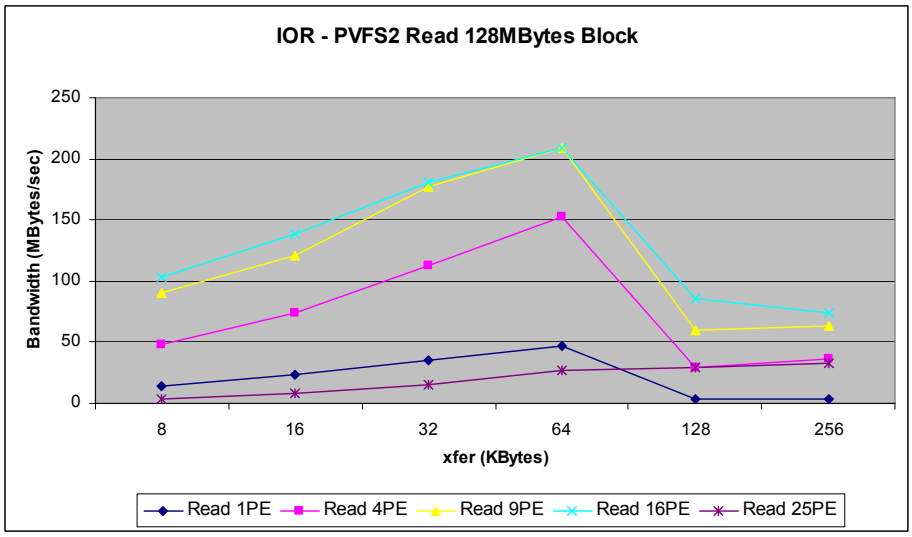

Fig. 3. PVFS2 Performance Tests: Read

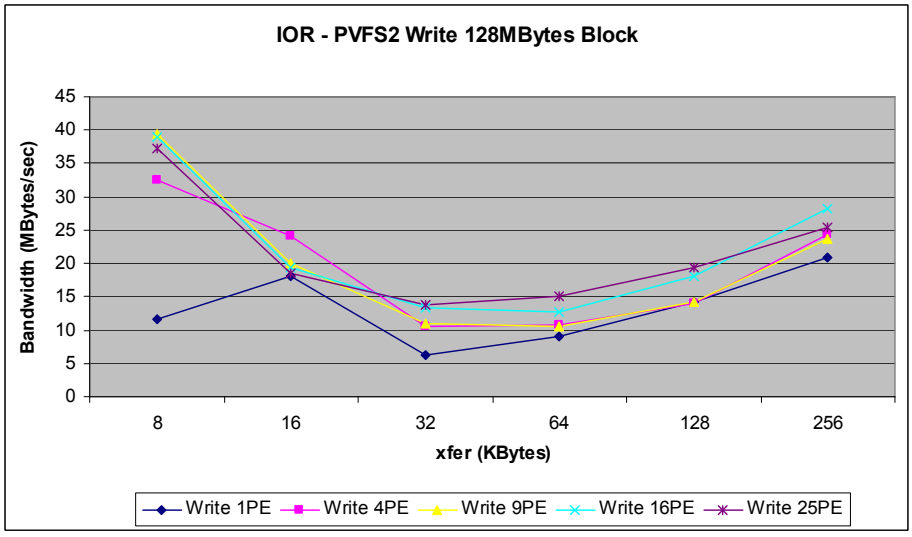

Fig. 4. PVFS2 Performance Tests: Write

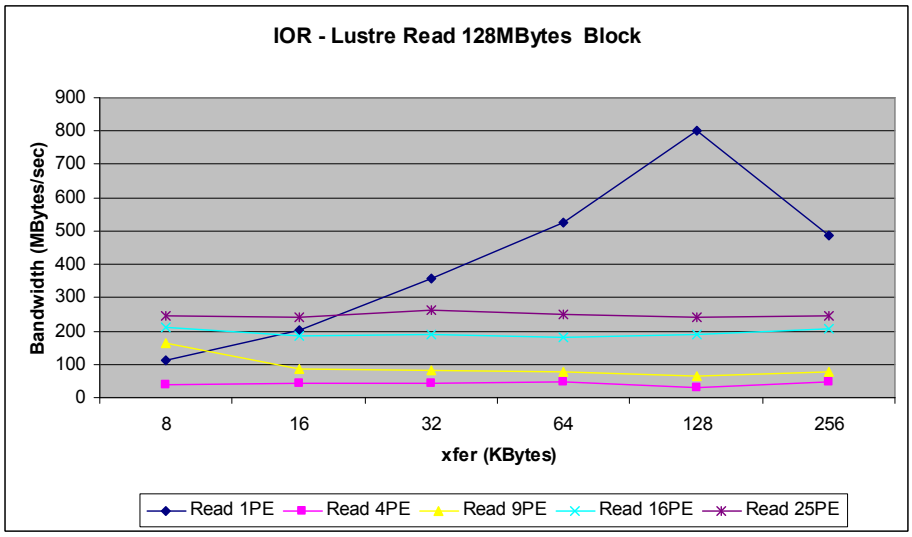

Fig. 5. Lustre Performance Tests: Read

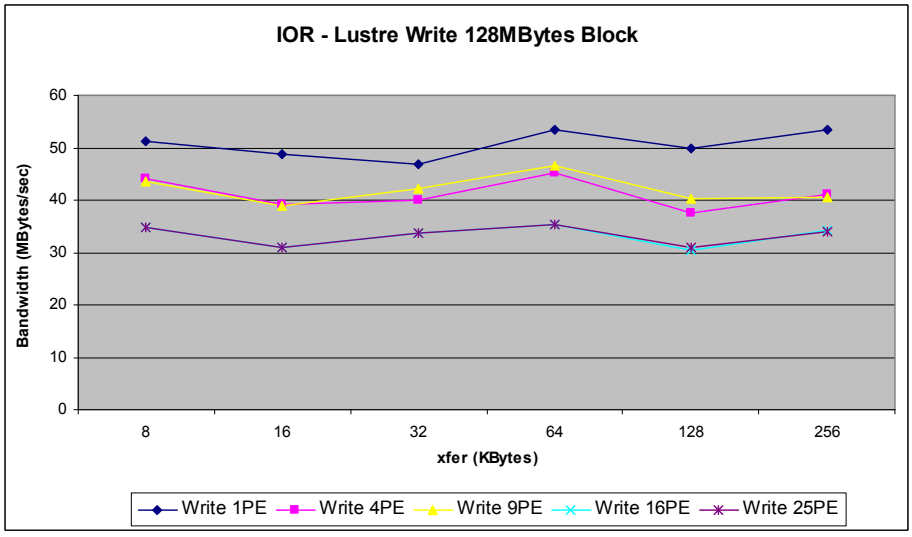

Fig. 6. Lustre Performance Tests: Write

nodes is large. NFSv4 is easier to set up than the other file systems, but it is limited by performance.

5.2 Scalability

Each file system performs differently under various types of workloads. As the number of clients reading or writing to a particular file system changes, the performance of the file system can increase, remain the same, or decrease. In this test, we evaluated how each networked file system would scale for various numbers of clients. We used the IOR benchmark and changed the number of clients (PEs) to perform scalability tests.

Figures 7 and 8 compare the scaling of the tested file systems. PVFS2 scaled almost linearly for write operations as more nodes were added up to 25 nodes. However, the performance of PVFS2 in general is not as good as Lustre or NFSv4. Lustre performed slightly better than PVFS2 with bandwidth decreasing from 53 Mbytes/s to 35 Mbytes/s as the number of clients increases. Although NFSv4 was able to attain maximum bandwidth of 93 Mbytes/s for one client, its performance deteriorates as the number of clients increases. In particular, the NFSv4 bandwidth decreases to 28 Mbytes/s for nine nodes, but slightly improves later on for 16 and 25 nodes reaching up to approximately 34 Mbytes/s for 25 nodes.

The read performance for NFSv4 on the other hand does not scale at all. PVFS2 and Lustre read performance increases as the number of nodes increases reaching 208 Mbytes/s and 249 Mbytes/s, respectively.

Three conclusions can be drawn from the performed scalability tests: First, Lustre and PVFS2 scale reasonably well as the number of nodes increase. Second, Lustre and PVFS2 do not perform well for small reads and writes as compared to NFSv4. This is primary due to the fact that Lustre and PVFS2 are designed for large data transfer. Third, NFSv4 write and read performance and scalability is severely limited by its single server architecture design.

5.3 Availability

Overall HPC system availability has become more critical in the last few years with the recent trend toward capability computing. Running scientific applications on the largest machines available for days, weeks, or even months at a time, requires high-level reliability, availability, and serviceability (RAS). Capability HPC machines are inherently large and complex interdependent systems. Due to the parallel nature of scientific applications, individual compute nodes directly depend on each other. Additionally, they also directly depend on service nodes for system resource management, networked data storage, I/O, etc..

Simple networked file systems, such as NFSv4, are a single point of failure and control for the entire HPC system or partition they belong to as a service outage severely disrupts system operation until repair. Parallel file system solutions, such as PVFS and Lustre, typically consist of three major components: clients, storage servers and a metadata server (MDS). While storage servers are

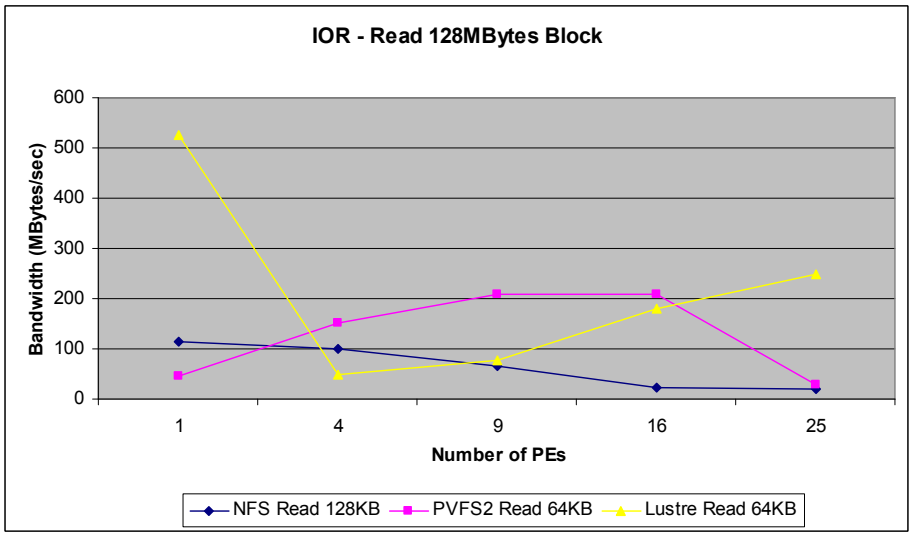

Fig. 7. Scalability Tests: Read

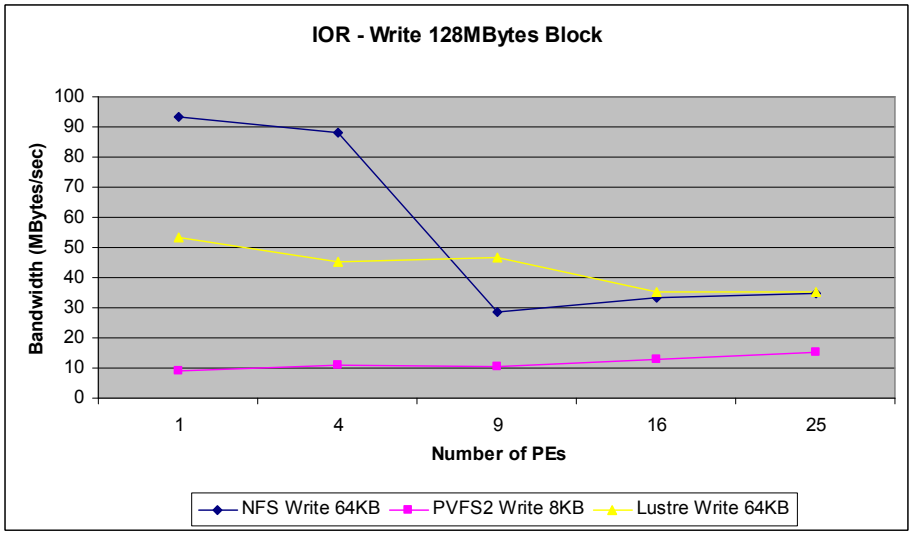

Fig. 8. Scalability Tests: Write

able to provide high availability of stored data utilizing common data replication techniques, such as striping across storage servers, the MDS is a single point of failure and control for an entire parallel file system, and therefore for the entire HPC system or partition it belongs to. In both cases, the vulnerable service may be equipped with a replication technique utilizing multiple redundant service nodes. A service-level replication mechanism is needed to assure consistent replication.

The concept of using a shared storage device for saving service state is a common technique for providing high availability, but it has its pitfalls. Service state is saved on the shared storage device upon modification, while a standby service node takes over in case of a failure of the active service node. The standby monitors the health of the active service using a heartbeat mechanism and initiates the fail-over procedure. An extension of this technique uses a crosswise active/standby redundancy strategy. In this case, both are active services and additional standby services for each other. In both cases, the mean-time to recover (MTTR) depends on the heartbeat interval, which may vary between a few seconds and several minutes.

While the shared storage device is typically an expensive redundant array of independent drives (RAID) and therefore highly available, it remains a single point of failure and control. Furthermore, file system corruption on the shared storage device due to failures occurring during write operations are not masked unless a journaling file system is used and an incomplete backup state is discarded, *i.e.*, a commit protocol for saving backup state is used. Correctness and quality of service are not guaranteed if no commit protocol is used. The requirement for a journaling file system impacts the fail-over procedure by adding a file system check, which in-turn extends the MTTR.

The shared storage solution has become very popular with the `heartbeat` program [12, 13], which includes failure detection and automatic failover with optional network address cloning feature. Recent enhancements include support for file systems on a Distributed Replicated Block Device (DRBD) [14, 15], which is essentially a storage mirroring solution that eliminates the single shared device and replicates backup state to local storage of standby services. This measure is primarily a cost reduction, since an expensive RAID system is no longer required. However, the requirement for a journaling file system and commit protocol remain to guarantee correctness and quality of service, since the DRBD operates at the block device level for storage devices and not at the file system level.

NFSv4, PVFS and Lustre do not have built-in high availability support. However, high availability for the NFSv4 service and for the MDSs of PVFS and Lustre can be provided by involving a secondary node and a shared storage device in conjunction with the `heartbeat` program. In this case, the supported high availability configurations are active/standby and crosswise active/standby.

A recent accomplishment [16, 17] targeted the symmetric active/active replication model [18–20], which uses multiple redundant service nodes running in virtual synchrony via a state-machine replication mechanism. In this model, service node failures do not cause a fail-over to a backup and there is no disruption

of service or loss of service state. The size of the active service group is variable at runtime, *i.e.*, services may join, leave, or fail. Its membership is maintained by a group communication system in a fault tolerant, adaptive fashion. As long as one active service is alive, state is never lost, state changes can be performed, and output is produced according to state changes.

The developed symmetric active/active solution for the MDS of PVFS performed correctly and offered a remarkable performance with only 26ms latency overhead for MDS writes and 300% of PVFS MDS baseline throughput for MDS reads in a 4 service node system. While the incurred latency overhead is caused by the total message ordering of the group communication system, the throughput improvement is a result of a load balancing strategy for read requests across the replicated MDS nodes. Assuming a mean-time to failure (MTTF) of 5,000 hours for a service node, the prototype improved service availability from 99.285% to 99.995% in a two-node system, and to 99.99996% with three nodes.

6 Summary

In summary, multiple options are available for attaching storage to diskless HPC systems or system partitions. For systems with very light I/O loads, a NFS-based solution is probably sufficient. For heavier I/O loads, a parallel file system is preferable. Additionally, parallel file systems can potentially offer the highest performance and lowest overall cost for accesses to data storage. Considering the cost/performance factor, community-supported solutions, such as PVFS2 and Lustre, are preferred options that can be used in a wide variety of storage hardware.

Our work leverages parallel file systems for servicing a common root file system in a diskless HPC configuration. In particular, we have shown that Lustre is a viable solution for serving a root file system. However, we have also found certain drawbacks of the tested file system solutions, especially in the area of high availability.

The primary issue with NFS is the single point of access to a storage server resulting in a single point of failure and control. Additionally, NFS performance is unable to grow to the volume of data that is necessary to support I/O intensive applications on a large HPC system even with high performance network technology. On the other hand, Lustre and PVFS2 remove the dependency on a centralized monolithic solution. Furthermore, these parallel file systems are able to meet the need of performance and scalability of modern HPC systems. Like NFS, Lustre and PVFS2 lack of efficient out-of-the-box high availability support, *i.e.*, efficient redundancy for metadata servers. However, we have shown how recent advances in high availability technologies for HPC system services are able to address this issue.

For future work, we would like to explore high-bandwidth/low-latency HPC interconnects, such as InfiniBand, where sustained data rates of over 800 Mbytes/s were reported. Additionally, the IETF NFSv4 working group has introduced a parallel NFS (pNFS) [21] protocol extension derived from earlier work by

Panasas. It allows for object storage access to parallel data sources using out of band metadata servers. pNFS has claimed to perform well under both light and heavy loads. This potentially could provide a good solution to both, system (single root file system) and application data files (scratch file system).

Additionally, we would like to explore other scalability factors such as booting. In particular, the system boot process is still dependent on a single server to provide TFTP services, which limits the number of nodes that can boot simultaneously. A possible solution is to replicate the TFTP service across multiple servers.

References

1. Internet Engineering Task Force (IETF): Request For Comments (RFC) 3530: Network File System (NFS) version 4 Protocol At http://www.ietf.org/rfc/rfc3530.txt.
2. Cluster File Systems, Inc.: Lustre Cluster File System documentation At http://www.lustre.org.
3. Sun Microsystems, Inc.: Lustre file system – High-performance storage architecture and scalable cluster file system White paper.
4. PVFS Development Team: Parallel Virtual File System (PVFS) documentation At http://www.pvfs.org.
5. Laros, J.H., Ward, L.H.: Implementing scalable disk-less clusters using the network file system. In: Proceedings of the Los Alamos Computer Science Institute (LACSI) Symposium 2003, Santa Fe, NM, USA (October 27-29, 2003)
6. Olivares, T., Orozco-Barbosa, L., Quiles, F., Garrido, A., Garcia, P.J.: Performance study of NFS over Myrinet-based clusters for parallel multimedia applications. Electrical and Computer Engineering **2**(5) (2001) 999–1004
7. Schmuck, F., Haskin, R.: GPFS: A shared-disk file system for large computing clusters. In: Proceedings of 1^{st} Conference on File and Storage Technologies (FAST) 2002, Monterey, CA, USA (January 28-30, 2002) 231–244
8. Reed, D.A.: Grids, the TeraGrid, and beyond. Computer **36**(1) (2003) 62–68
9. Catlett, C.: The philosophy of TeraGrid: Building an open, extensible, distributed terascale facility. Proceedings of 2^{nd} IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid) 2002 (May 21-24, 2002) 8
10. Panasas Inc.: Panasas ActiveScale file system datasheet At http://www.panasas.com.
11. Lawrence Berkeley National Laboratory: IOR Benchmark documentation At http://www.llnl.gov/asci/purple/benchmarks/limited/ior.
12. Robertson, A.L.: The evolution of the Linux-HA project. In: Proceedings of the UKUUG LISA/Winter Conference – High-Availability and Reliability – 2004, Bournemouth, UK (June 21, 2004)
13. Linux-HA (High-Availability Linux) project: Heartbeat program documentation At http://www.linux-ha.org/HeartbeatProgram.
14. Reisner, P., Ellenberg, L.: Drbd v8 – Replicated storage with shared disk semantics. In: Proceedings of the 12^{th} International Linux System Technology Conference (Linux-Kongress) 2005, Hamburg, Germany (October 11-14, 2005)
15. Reisner, P., Ellenberg, L.: Distributed Replicated Block Device (DRDB) documentation At http://www.drbd.org.

16. He, X., Ou, L., Engelmann, C., Chen, X., Scott, S.L.: Symmetric active/active metadata service for high availability parallel file systems. Journal of Parallel and Distributed Computing (JPDC) (2008) Submitted, under review.
17. Ou, L., Engelmann, C., He, X.B., Chen, X., Scott, S.L.: Symmetric active/active metadata service for highly available cluster storage systems. In: Proceedings of the 19^{th} IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS) 2007, Cambridge, MA, USA (November 19-21, 2007)
18. Engelmann, C.: Symmetric Active/Active High Availability for High-Performance Computing System Services. PhD thesis, Department of Computer Science, University of Reading, UK (2008)
19. Engelmann, C., Scott, S.L., Leangsuksun, C.B., He, X.B.: Symmetric active/active high availability for high-performance computing system services: Accomplishments and limitations. In: Proceedings of the 8^{th} IEEE International Symposium on Cluster Computing and the Grid (CCGrid) 2008: Workshop on Resiliency in High Performance Computing (Resilience) 2008, Lyon, France (May 19-22, 2008) 813–818
20. Engelmann, C., Scott, S.L., Leangsuksun, C.B., He, X.B.: Symmetric active/active high availability for high-performance computing system services. Journal of Computers (JCP) **1**(8) (2006) 43–54
21. Hildebrand, D., Ward, L., Honeyman, P.: Large files, small writes, and pNFS. In: Proceedings of 20^{th} ACM International Conference on Supercomputing (ICS) 2006, Cairns, Australia (June 28-30, 2006) 116–124