pNFS, POSIX, and MPI-IO: A Tale of Three Semantics

Dean Hildebrand
IBM Almaden
dhildeb@us.ibm.com

Arifa Nisar
Northwestern University
a-nisar@u.northwestern.edu

Roger Haskin
IBM Almaden
roger@almaden.ibm.com

ABSTRACT

MPI-IO is emerging as the standard mechanism for file I/O within HPC applications. While pNFS demonstrates high-performance I/O for bulk data transfers, its performance and scalability with MPI-IO is unproven. To attain success, the consistency semantics and interfaces of pNFS, POSIX, and MPI-IO must all be reconciled and efficiently translated. This paper investigates and discusses the challenges of using pNFS to support the consistency semantics of HPC applications.

1. INTRODUCTION

Advanced research collaborations push the bounds of modern technology, but continue to be constrained by rigid computing and storage infrastructures. Many large compute clusters are tightly coupled with a single file system, requiring large data sets to be moved multiple times across a computational grid [1]. In addition, while parallel file systems provide high I/O throughput to large data stores, they are highly specialized, have limited OS and hardware support, lack seamless integration and modern security features, and suffer from slow remote access [2-4].

Many HPC applications use the MPI-IO interface to separate themselves from the strict confines of a particular compute cluster [5]. Beyond an abstraction layer, MPI-IO further allows nodes to coordinate and optimize access to the file system, and is the foundation of modern data access libraries such as HDF5 and Parallel-netCDF [6, 7]. ROMIO, a popular MPI-IO implementation developed at Argonne National Laboratories, improves I/O performance through a variety of techniques including data sieving and collective I/O [8].

Unfortunately, MPI-IO does not completely shield applications, as some file systems continue to use specialized tuning parameters. In addition, MPI-IO offers this abstraction only to applications, leaving other producers and consumers of data, e.g., archival and backup systems, local and remote desktops, visualization clusters, to continue suffering from portability headaches.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Supercomputing PDSW'09, Nov. 15, 2009. Portland, OR, USA.
Copyright © 2009 ACM 978-1-60558-883-4/09/11... $10.00

Distributed file systems such as NFS and CIFS provide heterogeneous data access and are widely available, but their "single server" design, which binds one network endpoint to a given collection of files, limits opportunities to scale with network, CPU, memory, and disk I/O resources. NFSv3 is a poor match for HPC applications due to its poorly defined locking protocol and caching semantics. An existing ROMIO module for NFSv3 exists, but the unpredictable NFSv3 caching behavior forces it to lock/unlock and open/close files excessively, severely reducing performance. NFSv4 [9] improves functionality by including well-defined security and locking frameworks as well as migration and replication features, but retains the single server bottleneck.

pNFS, an integral part of NFSv4.1 [10], promises to bridge the gap between the performance requirements of large, parallel applications and the interoperability and security requirements of modern Grid workflows. pNFS provides high-performance data access to large-scale storage systems in both LAN and WAN environments [11, 12]. In addition, pNFS decouples the tight bond between storage systems and their clients, enabling pNFS clients to directly access parallel file systems. Direct access reduces latency, allows full use of the available network bandwidth, and reduces the management overhead and storage space required to maintain copies of large data sets in multiple data centers.

An original promise of pNFS was, and still is, to replace native parallel file system clients within a cluster. Unfortunately, while pNFS can achieve high-performance for bulk data transfers, its scalability and close-to-open caching semantics are unproven with HPC applications using MPI-IO.

This paper argues that two major developments indicate that it is now time to take a second look at NFS with MPI-IO. First, the HPC community is arguing that applications do not require strict POSIX I/O semantics as supported by most parallel file systems, but instead require a relaxed version [13]. Second, pNFS introduces a well-defined locking protocol, stricter caching semantics, caching and lock performance improvements, and parallel data access. In combination, these events suggest that HPC applications and NFS have a prosperous future together.

In the remainder of this paper, we give an overview of pNFS and describe our implementation with GPFS. We then argue for the benefits of a commodity high-

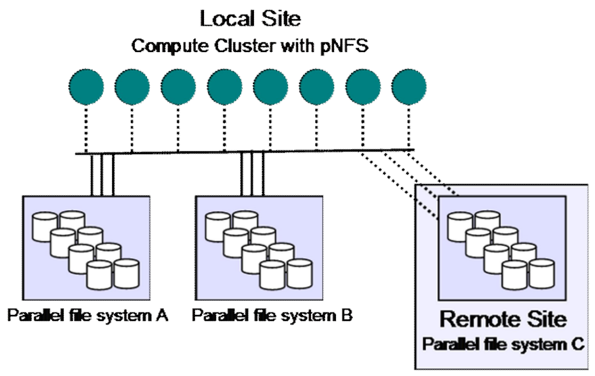


Figure 1. Using pNFS in a compute cluster. Applications can use a single pNFS client to mount and access their preferred parallel file system. The mounted file systems can be within the same site or across the WAN. Each compute node runs a stock Linux kernel.

performance file system client such as pNFS and discuss the challenges and benefits of using pNFS to support parallel MPI-IO applications.

2. COMMODITY FILE ACCESS

NFS owes its success to an open protocol, platform ubiquity, and transparent access to file systems—independent of the underlying storage technology. Beyond performance and scalability, the benefits offered by pNFS to the HPC community are numerous. One benefit is that clusters can decouple themselves from any particular parallel file system. Figure 1 demonstrates how applications can access the parallel file system that contains the data, regardless of whether it is within a LAN or across a WAN, thus reducing the cost of development, administration, and support.

Another benefit is that system administrators can select a storage solution with confidence that users are able to access their data. In addition, storage vendors are free to focus on advanced data management features such as fault tolerance, archiving, manageability, and scalability without having to custom tailor their products across a broad spectrum of client platforms.

3. BACKGROUND: pNFS WITH GPFS

Figure 2 displays the pNFS architecture with GPFS. The nodes in the GPFS cluster chosen for pNFS access are divided into (possibly overlapping) groups of state and data servers. pNFS clients are distributed across all state servers in a round-robin fashion. Clients send metadata requests to their associated state server while I/O is distributed across all of the data servers. Each state server functions as fully functional NFSv4.1 metadata server, with GPFS maintaining correctness using an internal management protocol.

To perform direct and parallel I/O, a pNFS client first requests layout information from a state server. A layout contains the information required to access any byte range of a

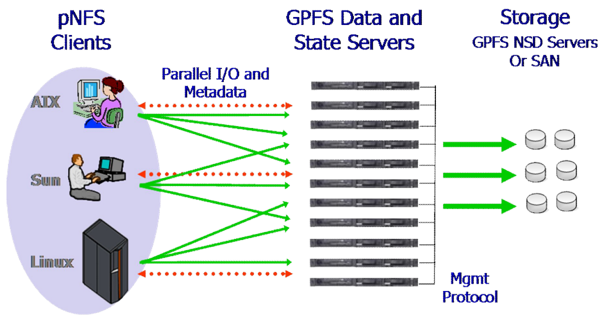


Figure 2. pNFS-GPFS Architecture. GPFS servers are divided into overlapping groups of state servers and data servers. pNFS clients access state servers for metadata operations and perform parallel I/O to the data servers.

file. The pNFS client uses the layout information to translate data access requests into READ and WRITE operations to the correct data servers. For writes, once the I/O is complete, the client sends an NFSv4 COMMIT operation to its state server. This single COMMIT operation has GPFS acquire exclusive access to the file, flushing data to stable storage on every data server. The GPFS management protocol maintains the freshness of NFSv4 state information among servers.

Since each GPFS server exports the entire file system, the layout does not indicate the actual location of the data. Instead, the layout provides a mechanism to balance client load among the data servers. This allows GPFS a great deal of flexibility in how it generates the layout information. For example, GPFS can rebalance data across the disks without needing to recall and generate new layout information for pNFS clients. In addition, layouts can be used to ensure all I/O for a byte-range of a file are sent to a single GPFS server, reducing lock contention and the number of read-modify-write sequences.

4. IT'S ALL SEMANTICS

The majority of parallel file systems support POSIX, the IEEE Portable Operating System Interface for Computing Environments. POSIX defines a standard interface and a strict set of semantics for applications to access a file system. Implementing POSIX semantics in multi-node file systems is extremely difficult and may incur severe performance penalties. For example, the POSIX requirement that writes to a file must be immediately visible to all other processes can cause extensive inter-node communication and create locking overhead if not carefully implemented.

Many file access protocols, including NFS, chose not to implement POSIX semantics to avoid severe performance penalties. Instead, NFS implements close-to-open semantics, which states that all file data and metadata changes must be on the server when a client closes a file. Once a client closes a file, changes are visible to other clients as soon as they open the file. NFSv4 now clearly defines these semantics within the protocol, but this laidback nature of close-to-open semantics prevents many parallel applications from using NFS.

MPI-IO recognizes that parallel applications are coordinated and should be granted greater control over their interaction with the file system. MPI-IO relaxes POSIX semantics and defines an interface that allows applications to manage cache coherency themselves. The MPI_FILE_SYNC command gives applications a guarantee about the freshness of data by controlling when data and metadata are flushed and revalidated.[1]

MPI-IO semantics are therefore looser than POSIX semantics, but are similar to NFSv4 semantics in how it flushes and revalidates data at specific points in time. Unfortunately, since POSIX is the only user-accessible interface to NFS and many parallel file systems, any potential performance gains can be lost [13]. Our project intends to demonstrate that pNFS can maintain its raw performance while leveraging its looser semantics to meet the needs of HPC applications with MPI-IO.

5. ACHIEVING MPI-IO SEMANTICS

This section discusses MPI-IO semantics, the performance obstacles of using NFS for their implementation, and several techniques to potentially regain performance.

5.1. The SYNC/BARRIER/SYNC Construct

This section describes MPI consistency semantics. To ensure data written on one node is visible to other nodes, all nodes perform a SYNC/BARRIER/SYNC [14] between write and read requests:

1. SYNC: Place written data on filing servers.

2. BARRIER: Clients wait for other clients to flush dirty data to the servers, ensuring that no client issues read requests until all clients have the same view of file contents.

3. SYNC: Perform file revalidation by ensuring written data is visible to all nodes.

In short, SYNC/BARRIER/SYNC is a way to enforce an ordering and separation of I/O operations in time. It is the primary method used by MPI applications to order I/O operations between nodes and must be correctly implemented by any file system that supports MPI-IO semantics.

5.2. NFS ADIO Driver Performance Problems

ROMIO [8], a popular MPI-IO implementation from Argonne National Laboratory, interfaces with an underlying file system through an Abstract Device I/O (ADIO) layer [15]. The ADIO layer allows any file system to implement and optimize MPI-IO requests through standard or private interfaces and/or customized hints. As a result, users can run applications portably and efficiently on any supported file system.

For POSIX-compliant file systems, a “Unix File System” (UFS) ADIO driver exists. With POSIX semantics being stricter that MPI-IO semantics, the UFS driver performs little, if any, extra work to implement MPI-IO semantics.

The NFSv3 ADIO driver is a completely different matter, requiring an enormous amount of effort for its development [16]. The driver is extremely limited in its ability to implement MPI-IO semantics as portability requirements demand that all interaction with NFSv3 occur through the POSIX interface. Therefore, the NFS ROMIO driver relies heavily on NFS locking and NFS close-to-open semantics to achieve correctness. The NFS client revalidates file data on a LOCK or OPEN request, and flushes all dirty data to disk on an UNLOCK or CLOSE request.

Applications using the NFSv3 driver continue to face many correctness and performance challenges:

- Flaky and inconsistent NFSv3 implementations on different operating systems. For example, clients reading a single byte may cache an entire page, forcing the driver to acquire and release locks after every write to flush cache.
- Well known protocol and implementation problems with the NFSv3 `lockd` daemon.
- NFSv3 client attribute caching must be disabled in order for a MPI client to view up-to-date file information. This greatly increases the chattiness of NFSv3 and henceforth the load on the NFSv3 server, severely reducing performance.

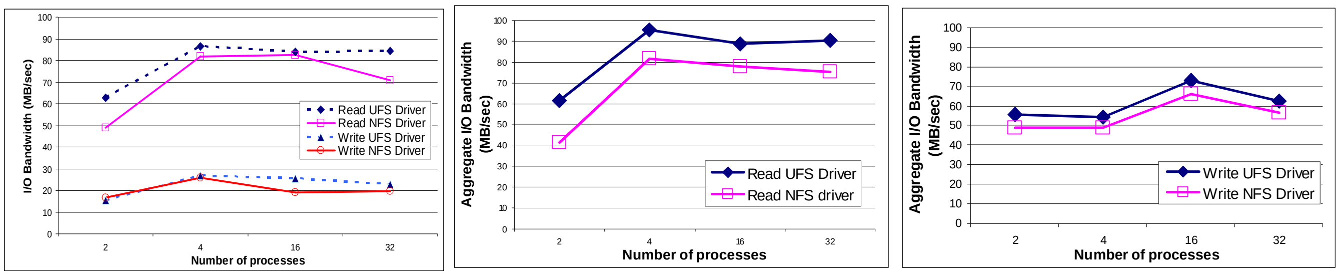
5.2.1. Example Application: POPIO Benchmark

The POPIO benchmark [17] is one of NCAR’s most scalable I/O codes and simulates the I/O requirements of a high resolution ocean model. POPIO uses MPI collective I/O operations to write and then read four files using a strided access pattern. The POPIO benchmark offloads all checkpoint data and then ingests it.

Figure 3 demonstrates the extra locking overhead in the current NFS ADIO driver by comparing it with the UFS ADIO driver, which does not perform `fcntl` locking. We can drop byte-range locking since checkpointing and ingest require revalidation only when a file is opened and closed.

With both NFSv4 accessing Ext3 and pNFS accessing GPFS, as the number of client processes increases, the current NFS ADIO driver sustains lower performance than the UFS ADIO driver due to its heavy use of `fcntl` locks to

[1] For completeness, MPI-IO also defines atomic operations, which provide the semantics similar to POSIX.



(a) NFSv4/Ext3 Read and Write (b) pNFS/GPFS Read (c) pNFS/GPFS Write

Figure 3. POP-IO benchmark read and write performance with UFS and NFS ADIO drivers. With both NFSv4 accessing Ext3 and pNFS accessing GPFS, the current NFS ADIO driver has lower performance than the UFS ADIO driver due to its heavy use of fcntl locks to protect and revalidate the data cache.

protect and revalidate the data cache. While maintaining correctness, for pNFS, read performance improves by 15% and write performance improves by up to 10%. The POPIO benchmark is executed with up to 32 processes running on 6 client machines. The NFSv4 server exporting Ext3 is on a single machine while pNFS accesses GPFS via two data servers. The POPIO benchmark is executed with up to 32 processes running on 6 client machines. The NFSv4 server exporting Ext3 is on a single machine while pNFS accesses GPFS via two data servers. The disk controller is the overall performance bottleneck in all experiments.

5.3. Building a Better Driver

The performance of pNFS combined with the integrated NFSv4 lock protocol opens up the possibility of building a correct and high-performance MPI-IO driver. Our goal is to build a new pNFS ROMIO driver that provides outstanding performance for the most common HPC workloads, i.e., using checkpoints to read and write large amounts of data.

Unfortunately, the POSIX interface continues to pose challenges. While the NFSv4 protocol has the tools to satisfy MPI-IO semantics, there is currently no revalidation interface in POSIX. As a result, the pNFS driver must rely on any number of other possible revalidation mechanisms to implement the SYNC/BARRIER/SYNC construct:

Direct I/O: The NFS client can avoid revalidation issues by simply avoiding data caching all together. This may be fine for some workloads, but the lack of readahead or writeback caching could severely hinder performance.

Locking: Now that NFSv4 has a well-defined locking protocol with clear semantics, a pNFS driver can use `fcntl` locks in a similar manner as the UFS driver. One side affect is that NFSv4.1 fault tolerance semantics mandate that all data must be written to disk on release of a write lock. This may create a performance penalty when the amount of written data is less than the size of the server cache.

Forced cache revalidation and invalidation: No portable revalidation mechanism exists, but a non-portable IOCTL function could force the NFS client to revalidate its data cache. In the longer term, if this mechanism proves useful, a standard interface can hopefully be adopted [13].

Force Data Eviction: Another non-portable technique is to evict cached data from memory. For example, Linux supports allows applications to clear all non-dirty pages from the page cache with '`sysctl -w vm.drop_caches=1`'. This achieves effectively the same behavior as using Direct I/O, but allows limited use of the read and writeback cache.

6. RELATED WORK

Previous work with pNFS focuses on micro-benchmarks and parallel applications that did not rely on the SYNC/BARRIER/SYNC construct [11, 12, 18].

Numerous high-performance file systems support MPI-IO [2, 19-21]. Some ADIO drivers continue to optimize I/O performance, e.g., Lustre and PanFS per-file striping parameters.

The HPC community has started an effort to extend the POSIX I/O API to improve clustering, parallelism, and high concurrency applications [13]. This effort may end up being critically important in providing platform-independent ADIO drivers.

GPFS implemented several optimizations for MPI-IO, including data shipping, improved sparse data access, and double buffering [20]. These optimizations are built directly into GPFS instead of ROMIO. While this can simplify the application interface, allowing additional enhancements at the ROMIO layer such as striping parameters would also be beneficial.

Scalable NAS solutions are emerging to provide scalable and heterogeneous access to large data stores [22-24]. These systems support NFSv3 and CIFS clients through multiple server endpoints. While these architectures improve scalability and can provide automatic failover, load balancing among the servers is uncoordinated and relies on out-of-band mechanisms such as DNS. In addition, forcing each client to perform all I/O through a single server creates hot spots, which limits available client bandwidth.

7. CONCLUDING REMARKS

HPC pushes the limits of modern computing, but continues to use rigid storage infrastructures. The emergence of pNFS promises to boost the capabilities of commodity file access. pNFS offers the promise of using a commodity client to access thousands of servers and petabytes of data. Ongoing work to align pNFS, MPI-IO, and POSIX consistency semantics directly addresses the needs of the high-performance community by seeking to overcome limitations that fetter portability, correctness, and performance.

8. REFERENCES

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus Striped GridFTP Framework and Server," in *Proceedings of Supercomputing '05*, Seattle, WA, 2005.

[2] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable Performance of the Panasas Parallel File System," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, San Jose, CA, 2008.

[3] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the USENIX Conference on File and Storage Technologies*, San Francisco, CA, 2002.

[4] Cluster File Systems Inc., "Lustre: A Scalable, High-Performance File System," www.lustre.org, 2002.

[5] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, *MPI: The Complete Reference, volume 2--The MPI-2 Extensions*. Cambridge, MA: MIT Press, 1998.

[6] NCSA, "HDF5 ", hdf.ncsa.uiuc.edu/HDF5.

[7] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A Scientific High-Performance I/O Interface," in *Proceedings of Supercomputing '03*, Phoenix, AZ, 2003.

[8] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," in *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, 1999.

[9] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "NFS Version 4 Protocol Specification," RFC 3530, 2003.

[10] S. Shepler, M. Eisler, and D. Noveck, "NFSv4 Minor Version 1," Internet Draft, 2008.

[11] D. Hildebrand, P. Andrews, M. Eshel, R. Haskin, P. Kovatch, and J. White, "Deploying pNFS across the WAN: First Steps in HPC Grid Computing," in *Proceedings of the 9th LCI International Conference on High-Performance Clustered Computing*, Urbana, IL, 2008.

[12] D. Hildebrand and P. Honeyman, "Exporting Storage Systems in a Scalable Manner with pNFS," in *Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*, Monterey, CA, 2005.

[13] G. Grider, L. Ward, R. Ross, and G. Gibson, "A Business Case for Extensions to the POSIX I/O API for High End, Clustered, and Highly Concurrent Computing," www.opengroup.org/platform/hecewg, 2006.

[14] "MPI-Forum," www.mpi-forum.org/docs/ mpi-20-html/node215.htm.

[15] R. Thakur, W. Gropp, and E. Lusk, "An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces," in *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation*, 1996.

[16] R. Thakur, E. Lusk, and W. Gropp, "Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation," Technical Memorandum ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory, Revised May 2004.

[17] M. Oberg, H.M. Tufo, and M. Woitaszek, "Exploration of Parallel Storage Architectures for a Blue Gene/L on the TeraGrid," in *Proceedings of the 9th LCI International Conference on High-Performance Clustered Computing*, Urbana, IL, 2008.

[18] D. Hildebrand, L. Ward, and P. Honeyman, "Large Files, Small Writes, and pNFS," in *Proceedings of the 20th ACM International Conference on Supercomputing*, Cairns, Australia, 2006.

[19] Sun Microsystems Inc., "Lustre File System," White-paper, 2007.

[20] J.P Prost, R. Treumann, R. Hedges, B. Jia, and A.E. Koniges, "MPI-IO/GPFS, an Optimized Implementation of MPI-IO on top of GPFS," in *Proceedings of Supercomputing '01*, Denver, CO, 2001.

[21] P.H. Carns, W.B. Ligon III, R.B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, 2000.

[22] I. Chavis, D. Coutts, J. Huie, S. Liu, S. Qualters, B. Demkowicz, and D.L Turkenkopf, "A Guide to the IBM Clustered Network File System," *IBM Redbooks*, 2008.

[23] IBM Corp., "IBM Storage Optimization and Integration Services-scale out file services," datasheet, 2007.

[24] M. Eisler, P. Corbett, M. Kazar, D. Nydick, and C. Wagner, "Data ONTAP GX: A Scalable Storage Cluster," in *Proceedings of the 5th USENIX conference on File and Storage Technologies*, San Jose, CA, 2007.