CITI Technical Report 06-06

pNFS and Linux: Working Towards a Heterogeneous Future

Dean Hildebrand
dhildebz@umich.edu

Peter Honeyman
honey@umich.edu

ABSTRACT

Anticipating terascale and petascale HPC demands, NFSv4 architects are designing pNFS, a standard extension that provides direct storage access to high-performance file systems while preserving operating system and hardware platform independence. Researchers at the University of Michigan are collaborating with industry to develop pNFS for the Linux operating system. This paper discusses the progress and direction of Linux pNFS.

May 10, 2006

Center for Information Technology Integration
University of Michigan
535 W. William St., Suite 3100
Ann Arbor, MI 48103-4978

pNFS and Linux: Working Towards a Heterogeneous Future

Dean Hildebrand
Center for Information Technology Integration
University of Michigan
dhildebz@eecs.umich.edu

Peter Honeyman
Center for Information Technology Integration
University of Michigan
honey@citi.umich.edu

Abstract

Anticipating terascale and petascale HPC demands, NFSv4 architects are designing pNFS, a standard extension that provides direct storage access to high-performance file systems while preserving operating system and hardware platform independence. Researchers at the University of Michigan are collaborating with industry to develop pNFS for the Linux operating system. This paper discusses the progress and direction of Linux pNFS.

1. Introduction

Large research collaborations require global access to massive data stores. Parallel file systems feature impressive throughput, but sacrifice heterogeneous access, seamless integration, security, and cross-site performance. pNFS, an integral part of NFSv4.1, overcomes these enterprise and grand challenge-scale obstacles by enabling clients to access storage directly while preserving NFSv4 operating system and hardware platform independence. pNFS distributes I/O across the bisectional bandwidth of the storage network between clients and storage devices, removing the single server bottleneck so vexing to client/server-based systems. In combination, the elimination of the single server bottleneck and the ability for clients to access data directly from storage results in superior file access performance and scalability [1].

At the Center for Information Technology Integration at the University of Michigan, we are developing pNFS for the Linux operating system. A pluggable client architecture harnesses the potential of pNFS as a universal and scalable metadata protocol by enabling dynamic support for file layout format, storage protocol, and file system policies. In conjunction with several industry partners, a prototype is under development for support of file, block, object, and PVFS2 access methods. This paper discusses the progress and direction of Linux pNFS.

2. pNFS overview

pNFS is a heterogeneous metadata protocol. The NFSv4.1 client and server perform control and file management operations and delegate the responsibility for I/O to a storage-specific driver. By separating control and data flows, pNFS allows data to transfer in parallel from many clients to many storage endpoints. Distributing I/O across the bisectional bandwidth of the storage network between clients and storage devices removes the single server bottleneck.

Figure 1 displays the general pNFS architecture. The control path contains all NFSv4.1 operations and features.

The data path can support any storage protocol, but the IETF design effort focuses on file, object, and block storage protocols. Storage devices can be NFSv4.1 servers, other distributed file systems, object storage, even block-addressable disks. NFSv4.1 does not specify a management protocol, which may therefore be proprietary to the exported file system.

Clients perform direct and parallel I/O by first requesting data location (layout) information from the pNFS server. Clients then use the layout information in conjunction with the storage protocol to access data. For example, the NFSv4.1 file storage protocol stripes files across NFSv4.1 data servers (storage devices); only READ, WRITE, and COMMIT operations are used on the data path.

3. Pluggable storage protocol

Although parallel file systems separate control and data flows, there is tight integration of their control and data protocols. Users must adapt to different consistency and security semantics for each data repository. Using pNFS as a universal metadata protocol lets applications realize a consistent set of file system semantics across data repositories. Linux pNFS facilitates interoperability by providing a framework for the co-existence of the NFSv4.1 control protocol with all storage protocols. This is a major departure from current file systems, which can

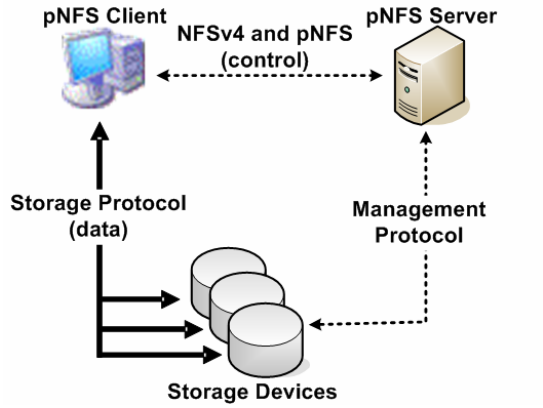


Figure 1. pNFS architecture
pNFS splits the NFSv4 protocol into a control path and a data path. The NFSv4.1 protocol exists along the control path. A storage protocol along the data path provides direct and parallel data access. A management protocol binds metadata servers with storage devices.

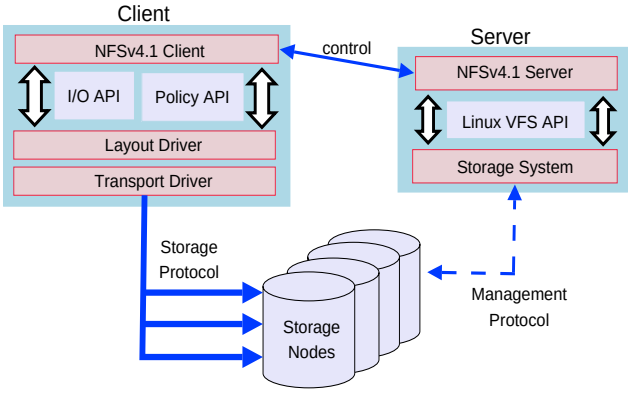


Figure 2. Linux pNFS architecture
The NFSv4.1 (pNFS) client uses I/O and policy interfaces to access storage nodes and follow underlying file system polices. The NFSv4.1 server uses VFS export operations to exchange pNFS information with the underlying file system.

only support a single storage protocol such as FCP or OSD [2, 3].

Figure 2 depicts the architecture of pNFS on Linux, which adds a layout and transport driver to the standard NFSv4 architecture. The *layout driver* understands the file layout of the storage system. A layout consists of all information required to access any byte range of a file. The layout driver uses the layout to translate read and write requests from the pNFS client into I/O requests understood by the storage devices. The *transport driver* performs I/O—e.g., iSCSI [4], Portals [5], SunRPC [6]—to the storage nodes.

Layout drivers are pluggable, using a standard set of interfaces for all storage protocols. An I/O interface facilitates the management of layout information and performing I/O with storage. A policy interface informs the pNFS client of file system and storage system specific policies. Example policies include the file system stripe and block size and when to retrieve layout information. An additional policy sets an I/O request size threshold that improves performance under certain workloads [7].

The policy interface also enables layout drivers to specify if it will use NFSv4.1 data management services or use customized implementations. The following is a list of services available to layout drivers:

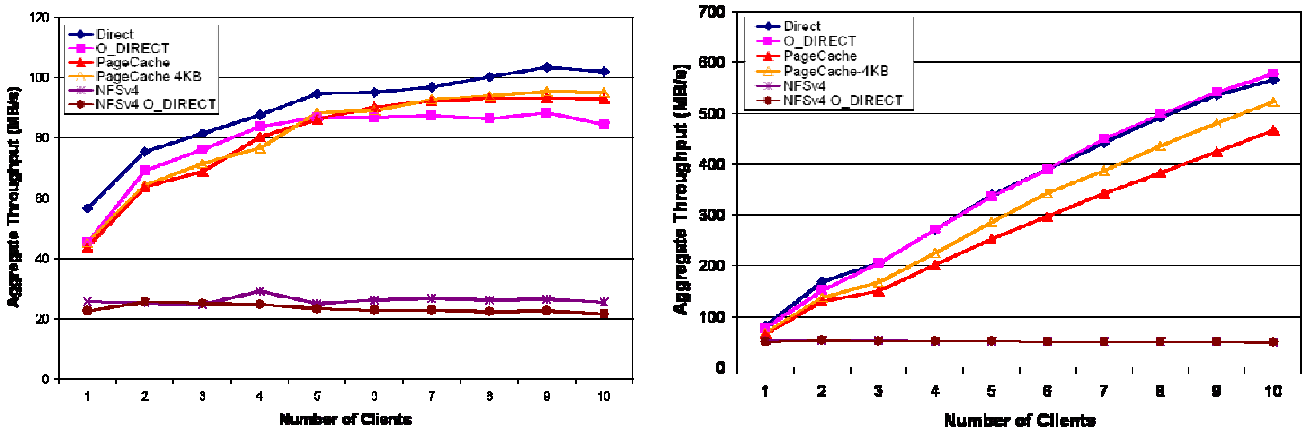
- Data cache
- Writeback cache with write gathering
- Readahead and read gathering algorithms

These policies can be set for each layout driver or acquired via the NFS server.

4. Evaluation

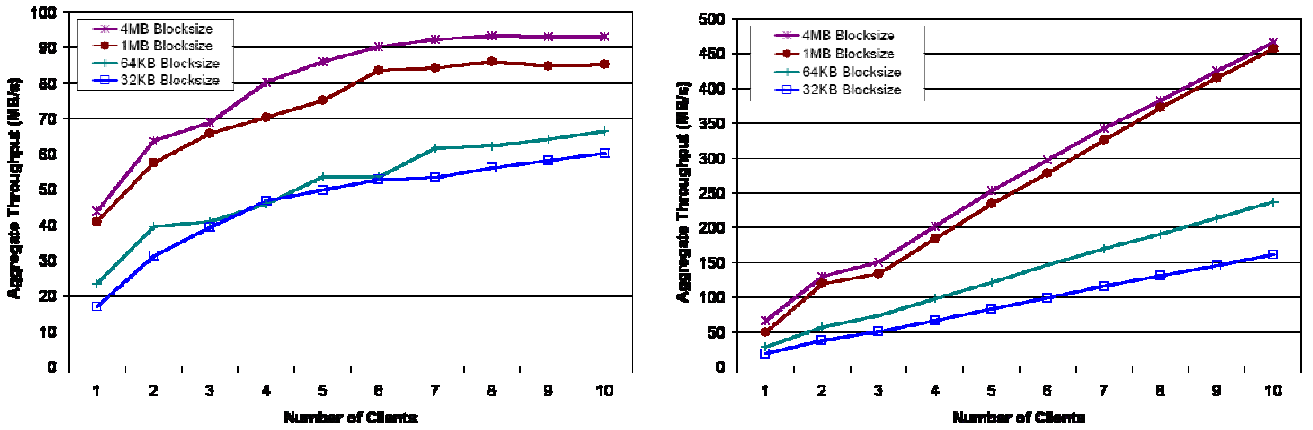
Our initial experiments evaluate different ways of accessing storage. We use our pNFS prototype with a PVFS2 file system and a PVFS2 layout driver. Native PVFS2 clients lack a data cache, providing high bandwidth data transfers with minimal overhead. With pNFS and the layout driver policy interface, the PVFS2 layout driver has the option of using the Linux (NFSv4.1) page cache. The flexibility of the policy interface facilitates fine-grained performance analysis of the data cache and NFSv4.1 I/O request processing. We plan further experiments that evaluate cross-file system transfer performance using different types of layout drivers on a single client.

The current Linux pNFS client prototype can access data through the Linux *page cache*, using *O_DIRECT*, or directly by bypassing the Linux page cache and all NFSv4 I/O request processing (*direct* access method). When using the Linux page cache, I/O requests are gathered or split into block sized requests before being sent to storage, with requested data cached on the client. Accessing data with O_DIRECT is similar, except data does not pass through the Linux page cache. When accessing data directly, I/O requests bypass the Linux page cache and NFS subsystem and are given directly to the layout driver. The first two data access methods are currently supported by the Linux NFS implementation. The direct method is the default behavior of some high-performance file systems, e.g., PVFS2.



a. Write b. Read

Figure 3. Aggregate I/O throughput to six PVFS2 storage nodes using the pNFS direct, O_DIRECT, and page cache access methods and the standard NFSv4 and NFSv4 with O_DIRECT access methods.



a. Write b. Read

Figure 4. pNFS/PVFS2 aggregate I/O throughput to six PVFS2 storage nodes using the pNFS page cache access method with four block sizes.

4.1. Main results

Our first set of experiments, shown in Figure 3, demonstrates the relative performance of each of the above pNFS access methods and standard NFSv4 with and without O_DIRECT. Clients write and read separate 200 MB files in 4 MB chunks. With six data servers, the available disk write bandwidth is 6 x 20 MB/s = 120 MB/s. The `wsize` and `rsize` is 4 MB for pNFS and 64 KB for NFSv4.

NFSv4 write performance is flat, obtaining an aggregate throughput of 26 MB/s. NFSv4 with O_DIRECT is also fat with a slight reduction in performance. The direct access method has the greatest aggregate write throughput, obtaining over 100 MB/s with eight clients. The aggregate write throughput of pNFS clients using the page cache is consistently 10 MB/s lower than pNFS clients using the direct method. The *PageCache 4KB* access method, which writes data in 4 KB chunks, obtains the same performance as *PageCache* demonstrating the Linux pNFS client's ability to gather small requests into larger and more efficient requests. O_DIRECT obtains an aggregate write throughput between the direct and page cache access methods, but flattens out as the number of clients increases. The relative performance of the O_DIRECT and the page cache access methods is consistent with the relative performance of NFSv4 and NFSv4 with O_DIRECT.

NFSv4 read performance is flat, obtaining an aggregate throughput of 52 MB/s. The aggregate read throughput of pNFS clients using the two methods that avoid the page cache is the same as we increase the

number of clients, nearly exhausting the available network bandwidth with ten clients. Working through the page cache reduces the aggregate read throughput by up to 110 MB/s.

Our second set of experiments sets out to verify the performance sensitivity to layout driver block size (I/O request size) when working through the Linux page cache. As shown in Figure 4, increasing the block size from 32 KB to 4 MB improves aggregate I/O throughput, although this boost eventually hits a performance ceiling.

5. Future Directions

Petascale computing requires inter-site data transfers involving clusters that may have different operating systems and hardware platforms, incompatible or proprietary file systems, or different storage and performance parameters that require differing data layouts. pNFS offers a solution.

Figure 5 shows two clusters separated by a long range, high-speed WAN. Each cluster has the architecture described in Figure 1 and can use a different storage protocol as long as the pNFS client implements the appropriate pluggable storage protocol. (The management protocol is not shown.)

The *application cluster* is running an MPI application that wants to read a large amount of data from the server cluster and perhaps write to its backend. The MPI head node obtains the data location from the server cluster and distributes portions of the data location information (via MPI) to other application cluster nodes, enabling direct access to server cluster storage devices. The MPI application then reads data in parallel from the server cluster across the WAN, processes the data, and directs output to the application cluster backend.

A natural use case for this architecture is a visualization application processing the results of a scientific MPI code run on the server cluster. Another use case is an MPI application making a local copy of data from the server cluster on the application cluster.

pNFS not only bridges the gap between proprietary cluster file systems, it also opens cluster file systems to data access from enterprise desktop distributed file systems using common security, file naming, and file ACLs as the basis for data management.

Acknowledgments

This work was partially supported by the NSF Middleware Initiative Grant No. SCI-0438298, by ASC under contract B523296, and by grants from Network Appliance and IBM.

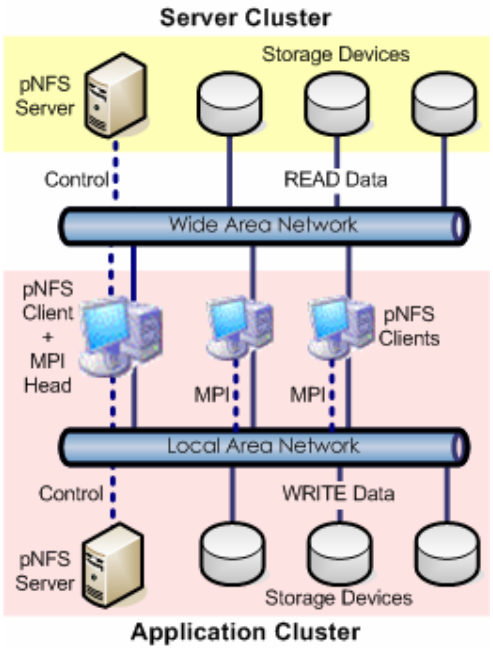


Figure 5. pNFS and inter-cluster data transfers across the WAN

A pNFS cluster retrieves data from a remote storage system, processes the data, and writes to its local storage system. The MPI head node distributes layout information to pNFS clients.

References

[1] 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.

[2] Panasas Inc., "Panasas ActiveScale File System," `www.panasas.com`.

[3] EMC Celerra HighRoad Whitepaper, `www.emc.com`, 2001.

[4] J. Satran, D. Smith, K. Meth, O. Biran, J. Hafner, C. Sapuntzakis, M. Bakke, M. Wakeley, L. Dalle Ore, P. Von Stamwitz, R. Haagens, M. Chadalapaka, E. Zeidner, and Y. Klein, "iSCSI," Internet Draft, `draft-ietf-ips-iscsi-08.txt`, 2001.

[5] R. Brightwell, A.B. Maccabe, R. Riesen, and T. Hudson, "The Portals 3.3 Message Passing Interface," 2003.

[6] R. Srinivasan, "RPC: Remote Procedure Call Protocol Specification Version 2," *RFC 1831*, 1995.

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