Direct-pNFS: Scalable, transparent, and versatile access to parallel file systems

Dean Hildebrand
University of Michigan
dhildebz@eecs.umich.edu

Peter Honeyman
University of Michigan
honey@citi.umich.edu

ABSTRACT
Grid computations require global access to massive data stores. To meet this need, the GridNFS project aims to provide scalable, high-performance, transparent, and secure wide-area data management as well as a scalable and agile name space.

While parallel file systems give high I/O throughput, they are highly specialized, have limited operating system and hardware platform support, and often lack strong security mechanisms. Remote data access tools such as NFS and GridFTP overcome some of these limitations, but fail to provide universal, transparent, and scalable remote data access.

As part of GridNFS, this paper introduces Direct-pNFS, which builds on the NFSv4.1 protocol to meet a key challenge in accessing remote parallel file systems: high-performance and scalable data access without sacrificing transparency, security, or portability. Experiments with Direct-pNFS demonstrate I/O throughput that equals or outperforms the exported parallel file system across a range of workloads.

Categories and Subject Descriptors
D.4.8 [**Operating Systems**]: Performance – *measurements.*

General Terms
Performance, Design, Experimentation

Keywords
Parallel I/O, NFSv4, pNFS, Distributed File System

1. INTRODUCTION
The GridNFS project aims to facilitate large data sets in the Grid by providing scalable name space management and by developing the means for scalable, transparent, and secure data access. To meet performance requirements, GridNFS may need to use all available bandwidth provided by a parallel file system's storage nodes. In addition, GridNFS must be able to provide simultaneous, parallel access to a single file from many clients, a common requirement of high-energy physics applications.

HPDC'07, June 25–29, 2007, Monterey, California, USA.

Parallel file systems feature impressive throughput, but they are highly specialized, have limited operating system and hardware platform support, and often lack strong security mechanisms. In addition, while parallel file systems excel at large data transfers, many do so at the expense of small I/O performance. While large data transfers dominate many scientific applications, many workload characterization studies highlight the prevalence of small, sequential I/O requests in modern scientific applications [1-3].

Many application domains demonstrate the need for high bandwidth, concurrent, and secure access to large datasets across a variety of platforms and file systems. Scientific computing that connects large computational and data facilities across the globe can generate petabytes of data. Digital movie studios that generate terabytes of data every day require access from compute clusters and Sun, Windows, SGI, and Linux workstations [4]. This need for heterogeneous data access produces a tension between parallel file systems and application platforms. Distributed file access protocols such as NFS [5] and CIFS [6] bridge the interoperability gap, but they are unable to deliver the superior performance of a parallel file system.

pNFS, an integral part of NFSv4.1 [7], overcomes these grand challenge-scale obstacles by enabling direct NFSv4.1 client access to storage while preserving operating system, hardware platform, and parallel file system independence. pNFS provides file access scalability by using the storage protocol of the underlying parallel file system to distribute 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 direct storage access by clients yields superior remote file access performance and scalability [8].

Regrettably, pNFS does not retain NFSv4 file system access transparency and can therefore not shield applications from different parallel file system security protocols and metadata and data consistency semantics. In addition, implementing pNFS support for every storage protocol on every operating system and hardware platform is a colossal undertaking. File systems that support standard storage protocols may be able to share development costs, but full support for a particular protocol is often unrealized, hampering interoperability. The pNFS file-based storage protocol bridges this transparency gap with middle-tier data servers, but eliminates direct data access, which can hurt performance.

1.1. Contributions
This paper introduces *Direct-pNFS*, a novel augmentation to pNFS that increases portability and regains parallel file system access transparency while continuing to match the performance of

native parallel file system clients. Architecturally, Direct-pNFS uses the NFSv4 storage protocol for direct access to a parallel file system's storage nodes. In addition, Direct-pNFS leverages the strengths of NFSv4.1 to improve I/O performance over the entire range of I/O workloads. We know of no other file access protocol that offers this level of performance, scalability, file system access transparency, and file system independence.

Direct-pNFS makes the following contributions:

Heterogeneous and ubiquitous remote parallel file system access. Direct-pNFS benefits are available with an unmodified NFSv4.1 client and does not require file system specific layout drivers, e.g., object [9] or PVFS2 [8].

Remote parallel file system access transparency and independence. pNFS uses parallel file system-specific storage protocols, which can expose gaps in the underlying file system semantics (such as security support). Direct-pNFS, on the other hand, retains NFSv4 file system access transparency by using the NFSv4 storage protocol for data access. In addition, Direct-pNFS can provide remote access to any parallel file system since it does not interpret file system-specific information.

I/O workload versatility. While distributed file systems and file access protocols are usually engineered to perform well on small data accesses [10], parallel file systems target large data transfers. Direct-pNFS combines the strengths of both, providing high-performance data access to the small and large data requests in scientific workloads.

Scalability and throughput. To realize the performance of exported parallel file systems, conventional pNFS prototypes are forced to support the exported storage protocol [8]. Direct-pNFS, on the other hand, can match the performance of the exported parallel file system without requiring support for any protocol other than NFSv4.1. This paper uses numerous benchmarks to demonstrate that Direct-pNFS matches the I/O throughput of a parallel file system, and has superior performance in workloads that contain many small I/O requests.

A case for commodity high-performance remote data access. Direct-pNFS complies with emerging IETF standards and uses an unmodified NFSv4.1 client. This paper makes a case for open systems in the design of high-performance clients, demonstrating that standards-compliant commodity software can deliver the performance of a non-standards based parallel file system client.

The remainder of this paper is organized as follows. Section 2 makes the case for open systems in distributed data access. Section 3 reviews pNFS and its departure from traditional client/server distributed file access protocols. Sections 4 and 5 describe the Direct-pNFS architecture and our Linux prototype. Section 6 reports the results of experiments with micro-benchmarks and four different I/O workloads. Section 7 discusses related work. We summarize and conclude in Section 8.

2. COMMODITY HIGH-PERFORMANCE REMOTE DATA 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, remote data access requires all these properties for success in Grid, cluster, enterprise, and personal computing.

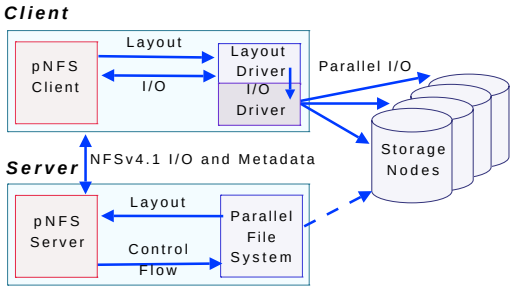


Figure 1. pNFS architecture. pNFS splits the NFSv4 protocol into control and data paths and adds a layout and I/O driver. The NFSv4.1 protocol governs the control path. To use the data path, a pNFS client obtains an opaque layout from a pNFS server and hands it off to the layout driver, which uses a storage-specific protocol to provide direct and parallel data access.

The benefits offered by Direct-pNFS or any other standards-based remote data access protocol are numerous. A single client can access data within a LAN and across a WAN, reducing the cost of development, administration, and support. System administrators can select a storage solution with confidence that regardless of the operating system and hardware platform, users are able to access the 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. SCALABLE I/O WITH pNFS

This section summarizes and evaluates the pNFS architecture. A full description can be found elsewhere [7, 8].

3.1. pNFS Overview

pNFS, an integral part of NFSv4.1, transforms NFSv4 into a heterogeneous metadata protocol. pNFS clients and servers are responsible for control and file management operations, but delegate I/O functionality to a storage-specific layout driver on the client. By separating control and data flows, pNFS clients can fully saturate the available bandwidth of the parallel file system.

Figure 1 displays the pNFS architecture. The control path contains all NFSv4.1 operations and features, continuing to use RPCSEC_GSS for authentication and NFSv4 ACLs (a super set of POSIX ACLs) for authorization. The data path can support any storage protocol, but the IETF design effort focuses on file-, object-, and block-based storage protocols.

pNFS adds a *layout driver* and an *I/O driver* to the standard NFSv4 architecture. The *layout driver* interprets and utilizes the opaque layout information returned from the pNFS server. A layout contains the information required to access any byte range of a file. In addition, a layout may contain file system specific access information. For example, the object-based layout driver requires the use of OSD access control capabilities [11]. To perform direct and parallel I/O, a pNFS client first requests layout information from the pNFS server. The layout driver uses the information to translate read and write requests from the pNFS client into I/O requests directed to storage devices. For example, the NFSv4.1 file-based storage protocol stripes files across NFSv4.1 data servers; only READ, WRITE, COMMIT, and ses-

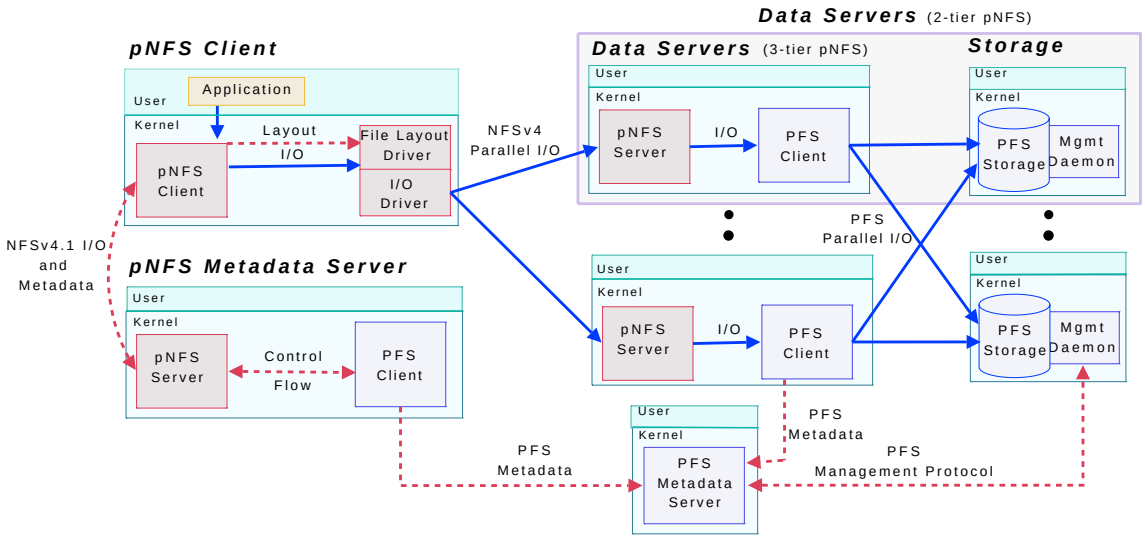


Figure 2. pNFS file-based architecture. A pNFS file-based system consists of pNFS data servers, clients and a metadata server, plus parallel file system (PFS) storage nodes, clients, and metadata servers. The three-tier design prevents direct storage access and creates overlapping and redundant storage and metadata protocols. The two-tier design, which places pNFS servers and the exported parallel file system clients on storage nodes, suffers from these problems plus diminished single client bandwidth.

sion operations are used on the data path. The pNFS server can generate layout information itself or, for non-NFSv4.1 file-based layouts, request assistance from the underlying file system. The *I/O driver* performs I/O, e.g., iSCSI [12], to the storage nodes.

3.2. Hybrid File System Semantics

Although parallel file systems separate control and data flows, there is tight integration of the control and data protocols. Users must adapt to different semantics for each data repository. pNFS, on the other hand, allows applications to realize common file system semantics across data repositories. As users access heterogeneous data repositories with pNFS, the NFSv4.1 metadata protocol provides a degree of consistency with respect to the file system semantics within each repository.

Unfortunately, certain semantics are layout driver and storage protocol dependent, and they can drastically change application behavior. For example, Panasas Activescale [13] supports the OSD security protocol [11], while Lustre [14] supports a specialized security protocol. This forces clients that need to access both parallel file systems to support multiple authentication, integrity, and privacy mechanisms. Additional examples of these semantics include client caching, and fault tolerance.

3.3. Burden of Layout Driver Development

Layout and I/O drivers are the workhorses of pNFS high-performance data access. These specialized components understand the parallel file system's storage protocol, security protocol, file system semantics, device identification, and layout description and management. For pNFS to achieve broad heterogeneous data access, layout and I/O drivers must be developed and supported on a multiplicity of operating system and hardware platforms—an effort comparable in magnitude to the development of a parallel file system client.

3.4. The pNFS File-Based Storage Protocol

Currently, the IETF is developing three storage protocols: file, object, and block. The NFSv4.1 protocol includes only the file-based storage protocol, with object and block to follow in separate specifications [9, 15]. As such, all NFSv4.1 implementations will support the file-based storage protocol, while support for object and block storage protocols will be optional.

A file-based layout governs an entire file and is valid until recalled by the pNFS server. To perform data access, the file-based layout driver combines the layout information with a known list of data servers for the file system, and sends READ, WRITE, and COMMIT operations to the correct data servers. Once I/O is complete, the client sends updated file metadata, e.g., size or modification time, to the pNFS server.

pNFS file-based layout information consists of:

- Aggregation type and stripe size
- Data server identifiers
- File handles (one for each data server)
- Policy parameters

Figure 2 illustrates how the pNFS file-based storage protocol provides access to a parallel file system (parallel FS). pNFS clients access pNFS data servers that export parallel FS clients, which in turn access data from parallel FS storage nodes and metadata from parallel FS metadata servers. A parallel FS management protocol binds metadata servers and storage, providing a consistent view of the file system. pNFS clients use NFSv4 for I/O while parallel FS clients use the parallel FS storage protocol.

3.4.1. Performance Issues

Architecturally, the pNFS file-based storage protocol offers some latitude. As shown in Figure 2, while pNFS clients always access remote pNFS servers, we can configure the exported parallel FS to create two- and three-tier architectures. The three-tier architecture separates parallel FS clients and storage nodes, while the

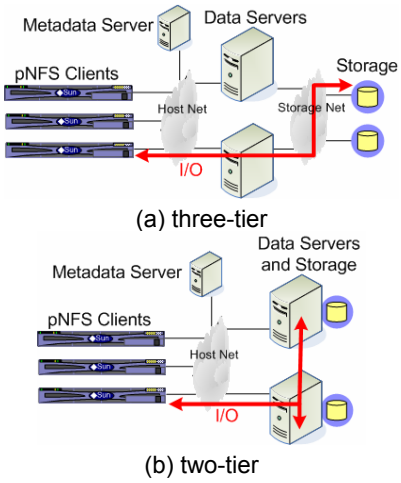


Figure 3. Indirect pNFS file-based data access. (a) Three-tier file-based pNFS uses intermediary data servers that block direct access to parallel file system storage nodes. (b) Two-tier file-based pNFS data servers must communicate to access both local and remote parallel file system storage nodes.

two-tier architecture couples parallel FS clients and storage on the same node. Figure 3 demonstrates how neither choice features direct data access: the three-tier architecture has intermediary data servers while with two-tiers, striped data is transferred between data servers, reducing the available bandwidth to the pNFS client. These architectures can improve NFS scalability, but the lack of direct data access—a primary benefit of pNFS—scuttles performance [16].

Block size mismatches and overlapping metadata protocols also diminish performance. If the pNFS block size is greater than the parallel FS block size, a large pNFS data request produces extra parallel FS data requests, each incurring a fixed amount of overhead [17]. Conversely, a small pNFS data request forces a large parallel FS data request, unnecessarily taxing storage resources and delaying the pNFS request. pNFS file system metadata requests to the pNFS server, e.g., file size, layout information, become parallel FS client metadata requests to the parallel FS metadata server. This ripple effect increases overhead and delay for pNFS metadata requests.

It is hard to address these remote access inefficiencies with fully connected block-based parallel file systems, e.g., GPFS [18], GFS [19, 20], and PolyServe Matrix Server [21], but for parallel file systems whose storage nodes admit NFSv4 servers, Direct-pNFS offers a solution.

4. DIRECT-pNFS

Direct-pNFS supports direct data access—without requiring a storage-specific layout driver on every operating system and hardware platform—by exploiting file-based layouts to describe the exact distribution of data on the storage nodes. Since a Direct-pNFS client knows the exact location of a file's contents, it can target I/O requests to the correct data servers. Direct-pNFS supports direct data access to any parallel file system that allows NFSv4 servers on its storage nodes—such as object based [13, 14], PVFS2 [22], and IBRIX Fusion [23]—and inherits the operational, fault tolerance, and security semantics of NFSv4.1.

4.1. Architecture

In the two- and three-tier pNFS file-based architectures shown in Figure 2, the underlying layout is unknown to pNFS clients. This forces them to distribute I/O requests among data servers without regard for the actual location of the data. To overcome this inefficient access, Direct-pNFS, shown in Figure 4, uses a *layout translator* to convert a parallel FS layout into a pNFS file-based layout. This provides Direct-pNFS clients with accurate knowledge of the underlying layout of data on storage. A pNFS server, which exists on every parallel FS data server, can now satisfy Direct-pNFS client data requests by simply accessing the local parallel FS storage component. Direct-pNFS and parallel FS metadata components also co-exist on the same node, which eliminates remote parallel FS metadata requests from the pNFS server.

In combination, the use of accurate layout information and the placement of pNFS servers on parallel FS storage and metadata nodes eliminates extra parallel FS data and metadata requests and obviates the need for data servers to support the parallel FS storage protocol altogether. The use of a single storage protocol also eliminates block size mismatches between storage protocols.

4.2. Layout Translator

To give Direct-pNFS clients exact knowledge of the underlying layout, a parallel FS uses the layout translator to specify a file's storage nodes, file handles, aggregation type, and policy parameters. The layout translator gathers this information and creates a pNFS file-based layout. The layout translator is independent of the underlying parallel FS and does not interpret parallel FS layout information.

The overhead in using the layout translator is small and confined to the metadata server. For example, our Linux prototype has the pNFS server specify the required file handles. The parallel FS needs to provide the layout translator with only the aggregation type and parameters, e.g., stripe size.

4.3. Optional Aggregation Drivers

It is impossible for the NFSv4.1 protocol (and hence NFSv4.1 clients) to support every method of distributing data among the storage nodes. At this writing, the NFSv4.1 protocol supports two aggregation schemes: round-robin striping and a second method that specifies a list of devices that form a cyclical pattern for all stripes in the file. To broaden support for unconventional aggregation schemes such as variable stripe size [24] and replicated or hierarchical striping [25, 26], Direct-pNFS also supports optional "pluggable" aggregation drivers. An *aggregation driver* provides a compact way for the Direct-pNFS client to understand how the underlying parallel FS maps file data onto the storage nodes.

Aggregation drivers are operating system and platform independent, and are based on the distribution drivers in PVFS2, which use a standard interface to adapt to most striping schemes. Although aggregation drivers are non-standard components, their development effort is minimal compared to the effort required to develop an entire layout driver.

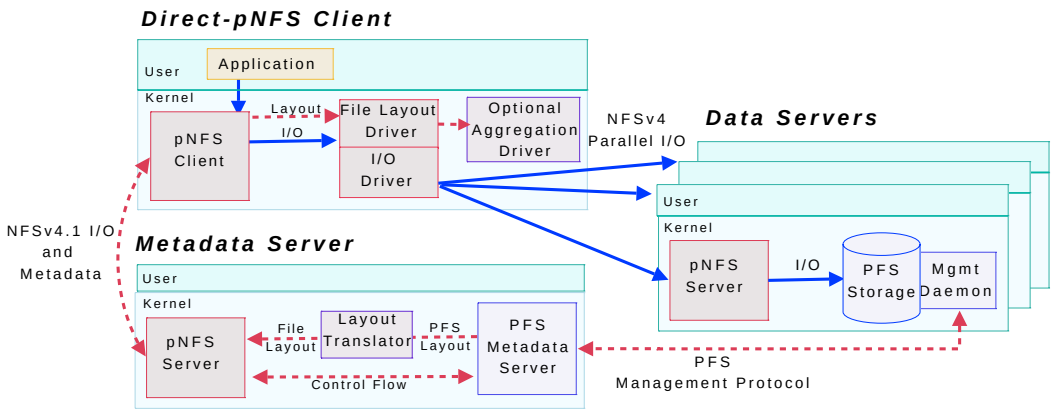


Figure 4. Direct-pNFS architecture. Direct-pNFS eliminates overlapping I/O and metadata protocols and uses the NFSv4 storage protocol to directly access parallel file system (PFS) storage nodes. The parallel file system uses a layout translator to converts its layout into a pNFS file-based layout. A Direct-pNFS client may use an aggregation driver to support specialized file striping methods.

5. DIRECT-pNFS PROTOTYPE

We implemented a Direct-pNFS prototype that maintains strict agnosticism of the underlying parallel file system and, as we shall see, matches the performance of the parallel file system that it exports. Figure 5 displays the architecture of our Direct-pNFS prototype, using PVFS2 for the exported file system.

PVFS2 is a user-level, open-source, scalable, parallel file system designed for the large data needs of scientific applications in research and production environments. PVFS2 uses large transfer buffers, supports limited request parallelization, incurs a substantial per-request overhead, and does not use a client data or write back cache. A kernel module allows integration into a user's environment and access by other file systems such as NFS.

Many scientific applications can re-create lost data, so PVFS2 buffers data on storage nodes and sends the data to stable storage only when necessary or at the application's request (`fsync`). To match this behavior, our Direct-pNFS departs from the NFSv4 protocol, committing data to stable storage only when an application issues an `fsync` or closes the file.

At this writing, the user-level PVFS2 storage daemon does not support direct VFS access. Instead, the Direct-pNFS data servers simulate direct storage access by way of the existing PVFS2 client and the loopback device. The PVFS2 client on the data servers functions solely as a conduit between the NFSv4 server and the PVFS2 storage node on the node.

Our Direct-pNFS prototype uses special NFSv4 StateIDs for access to the data servers, round-robin striping as its aggregation scheme, and the following NFSv4.1 operations:

GETDEVLIST: Issued at file system mount time. GETDEVLIST retrieves access information for the storage nodes in the underlying parallel file system.

LAYOUTGET: Issued after opening a file but before accessing file data. LAYOUTGET retrieves file access information for a byte-range of a file. Layouts apply to an entire file, are stored in a file's inode, and are valid for the lifetime of the inode.

LAYOUTCOMMIT: Issued after file I/O. LAYOUTCOMMIT informs the NFSv4.1 server of changes to file metadata such as a possible extension of the file size.

6. EVALUATION

In this section we asses the performance and I/O workload versatility of Direct-pNFS. We first use the IOR micro-benchmark [27] to demonstrate the scalability and performance of Direct-pNFS compared with PVFS2, the pNFS file-based storage protocol with two- and three-tiers, and NFSv4. To explore the versatility of Direct-pNFS, we use two scientific I/O benchmarks and two macro benchmarks to represent a variety of access patterns to large storage systems.

6.1. Experimental Setup

All experiments use a sixteen-node cluster connected via gigabit Ethernet with jumbo frames. One exception is the experiment in Figure 6c, which uses 100 Mbps Ethernet. To ensure a fair comparison between architectures, we keep the number of nodes and disks in the back end constant. The PVFS2 1.5.1 file system has six storage nodes, with one storage node doubling as a metadata manager, and a 2 MB stripe size. The pNFS three-tier architecture uses three NFSv4.1 servers and three PVFS2 storage nodes. For the three-tier architecture, we move the disks from the data servers to the storage nodes. All NFS experiments use eight server threads and 2 MB wsize and rsize. All nodes run Linux 2.6.17.

Parallel File System: Each PVFS2 storage node is equipped with dual 1.7 GHz P4 processors, 2 GB memory, one Seagate 80 GB 7200 RPM hard drive with Ultra ATA/100 interface and 2 MB cache, and one 3Com 3C996B-T gigabit card.

Client System: Client nodes one through seven are equipped with dual 1.3 GHz P3 processors, 2 GB memory, and an Intel Pro gigabit card. Client nodes eight and nine have the same configuration as the storage nodes.

6.2. Scalability and Performance

Our first set of experiments use the IOR benchmark to compare the scalability and performance of Direct-pNFS, PVFS2, two- and three-tier file-based pNFS, and NFSv4. Clients sequentially read and write separate 500 MB files as well as disjoint 500 MB portion of a single file. To view the effect of I/O request size on performance, the experiments use a large block size (2 to 4 MB) and a small block size (8 KB). Read experiments use a warm

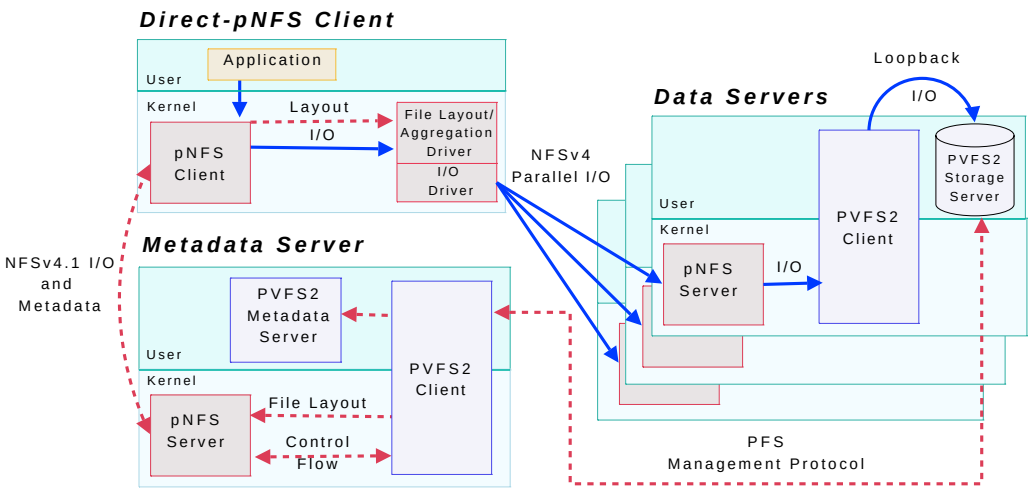


Figure 5. Direct-pNFS prototype architecture with PVFS2. The PVFS2 metadata server converts the PVFS2 layout into a pNFS file-based layout, which is passed to the pNFS server and then to the Direct-pNFS file-based layout driver. The pNFS data server uses the PVFS2 client as a conduit to retrieve data from the local PVFS2 storage daemon. Data servers do not communicate.

server cache. The presented value is the average over several executions of the benchmark

Figures 6a and 6b display the maximum aggregate write throughput with separate files and a single file. Direct-pNFS matches the performance of PVFS2, reaching a maximum aggregate write throughput of 119.2 MB/s and 110 MB/s for separate and single file experiments.

pNFS-3tier write performance levels off at 83 MB/s with four clients. pNFS-3tier splits the six available servers into data servers and storage nodes, which cuts the maximum network bandwidth in half relative to the other pNFS and PVFS2 architectures. In addition, using two disks in each storage node does not offer twice the disk bandwidth of a single disk due to the constant level of CPU, memory, and bus bandwidth.

Lacking direct data access, pNFS-2tier incurs a write delay and performs a little worse than Direct-pNFS and PVFS2. Transferring data between data servers restricts the maximum bandwidth between pNFS clients and data servers. This is not visible in Figures 6a and 6b because network bandwidth exceeds disk bandwidth, so Figure 6c repeats the multiple file write experiments with 100 Mbps Ethernet. Using a slower network, pNFS-2tier yields only half the performance of Direct-pNFS and PVFS2, clearly illustrating the network bottleneck of pNFS-2tier.

Figures 6d and 6e display the aggregate write throughput with separate files and a single file using an 8 KB block size. The performance for all NFSv4-based architectures is unaffected from the large block size experiments due to the NFSv4 client write back cache, which combines write requests until they reach the NFSv4 wsize (2 MB in our experiments). However, the performance of PVFS2, a parallel file system designed for large I/O, decreases dramatically with small block sizes, reaching a maximum aggregate write throughput of 39.4 MB/s.

Figures 7a and 7b display the maximum aggregate read throughput with separate files and a single file. With separate files, Direct-pNFS matches the performance of PVFS2, reaching a maximum aggregate read throughput of 509 MB/s. With a single file, PVFS2 has lower throughput than Direct-pNFS with only a few clients, but outperforms Direct-pNFS with eight clients, reaching a maximum aggregate read throughput of 530.7 MB/s. Direct-pNFS places the NFSv4 and PVFS2 server modules on the same node, placing higher demand on server resources. In addition, PVFS2 uses a fixed number of buffers to transfer data between the kernel and the user-level storage daemon, creating an additional bottleneck.

The division of the six available servers between data servers and storage nodes in pNFS-3tier limits its maximum performance again, achieving a maximum aggregate bandwidth of only 115 MB/s. NFSv4 aggregate performance is flat, limited to the bandwidth of a single server.

The pNFS-2tier bandwidth bottleneck is readily visible in Figures 7a and 7b, where disk bandwidth is no longer a factor. Each data server is not only responding to client read requests and but also transferring data to other data servers so they can satisfy their client read requests. Sending data to multiple targets limits each data server's maximum read bandwidth.

Figures 7c and 7d display the aggregate read throughput with separate files and a single file using an 8 KB block size. The performance for all NFSv4-based architectures is unaffected from the large block size experiments due to the use of the Linux page-cache and readahead algorithm. The performance of PVFS2 again decreases dramatically with small block sizes, reaching a maximum aggregate read throughput of 51 MB/s.

6.2.1. Discussion

In the write experiments, Direct-pNFS and PVFS2 fully utilize the available disk bandwidth. In the read experiments, data are read directly from the server cache, so the disks are not a bottleneck. Instead, client and server CPU performance becomes the limiting factor. The pNFS-2tier architecture offers comparable performance with fewer clients, but is limited by network bandwidth as we increase the number of clients. The pNFS-3tier architecture demonstrates that using intermediary data servers to access data is inefficient: those resources are better used as storage nodes.

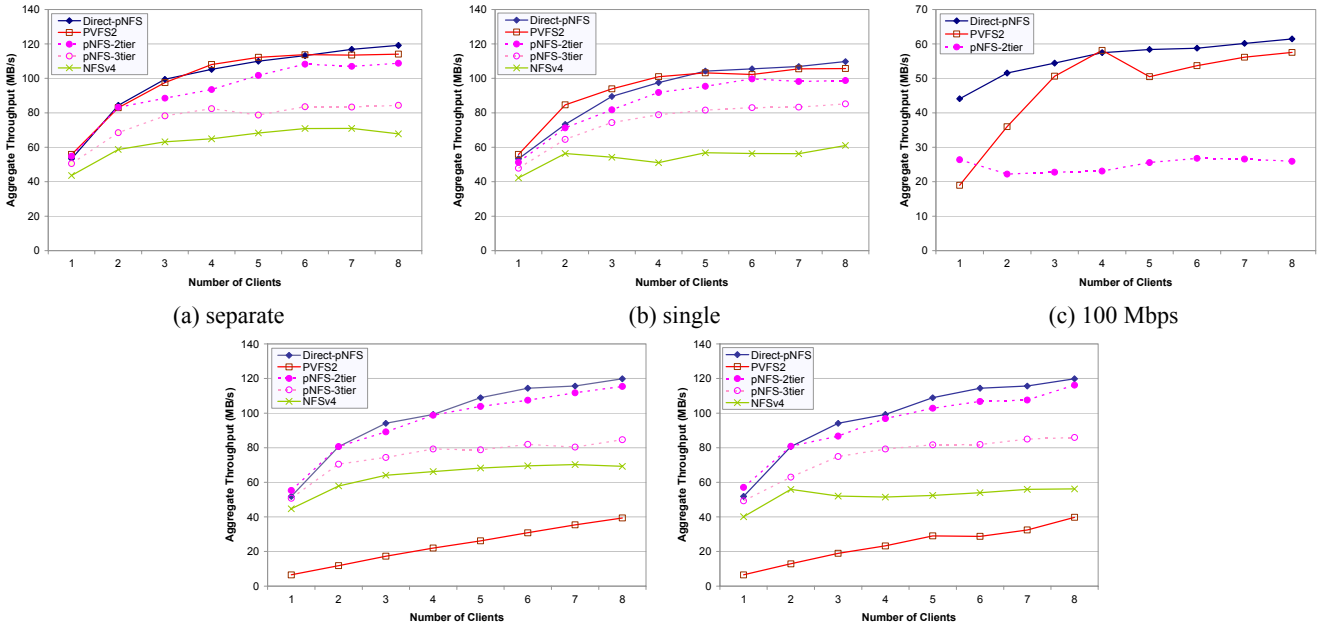


Figure 6. Aggregate write throughput. (a) and (b) With a separate or single file and a large block size, Direct-pNFS scales with PVFS2 while pNFS-2tier suffers from a lack of direct file access. pNFS-3tier and NFSv4 are CPU limited. (c) With separate files and 100 Mbps Ethernet, pNFS-2tier is bandwidth limited due to its need to transfer data between data servers. (d) and (e) With a separate or single file and an 8 KB block size, all NFSv4 architectures outperform PVFS2.

6.3. Scientific Application Benchmarks

This section uses two scientific benchmarks to analyze Direct-pNFS in the high-performance arena.

6.3.1. ATLAS

ATLAS [28] is a particle physics experiment under construction at CERN. The ATLAS detector can detect one billion events per second with a combined data volume of 40 TB — or a PB every 25 seconds! — so ATLAS scientists are performing large-scale simulations of the detector to develop real-time event filtering algorithms to reduce the volume of data. After filtering, data from fewer than one hundred events per second will be distributed for offline analysis.

The ATLAS simulation runs in four stages; the `Digitization` stage simulates detector data generation. With 500 events, `Digitization` spreads approximately 650 MB randomly over a single file. While 95 percent of the requests are less than 275 KB, 95 percent of the data are written in requests greater than or equal to 275 KB. Each client writes to a separate file.

To evaluate `Digitization` write throughput we used IOZone to replay the write trace data for 500 events. Each client writes to a separate file.

Figure 8a shows that Direct-pNFS can manage efficiently the mix of small and large write requests, achieving an aggregate write throughput of 102.5 MB/s with eight clients. While small write requests reduce the maximum write throughput achievable by Direct-pNFS by approximately 14 percent, they severely reduce the performance of PVFS2, which achieves only 41 percent of its maximum aggregate write throughput.

6.3.2. NAS Parallel Benchmark 2.4 – BTIO

The NAS Parallel Benchmarks (NPB) are used to evaluate the performance of parallel supercomputers. The BTIO benchmark is based on a CFD code. We use the class A problem set, which performs 200 time steps, checkpoints data every five time steps, and generates a 400 MB checkpoint file. The version of the benchmark used in this paper uses MPI-IO collective buffering, which increases the I/O request size to one MB and greater. The benchmark times also include the ingestion and verification of the result file.

BTIO performance experiments are shown in Figure 8b. BTIO running time is approximately the same for Direct-pNFS and PVFS2. With nine clients, the running time of Direct-pNFS is five percent longer due to the fixed number of buffers in PVFS2 (as discussed in Section 6.2).

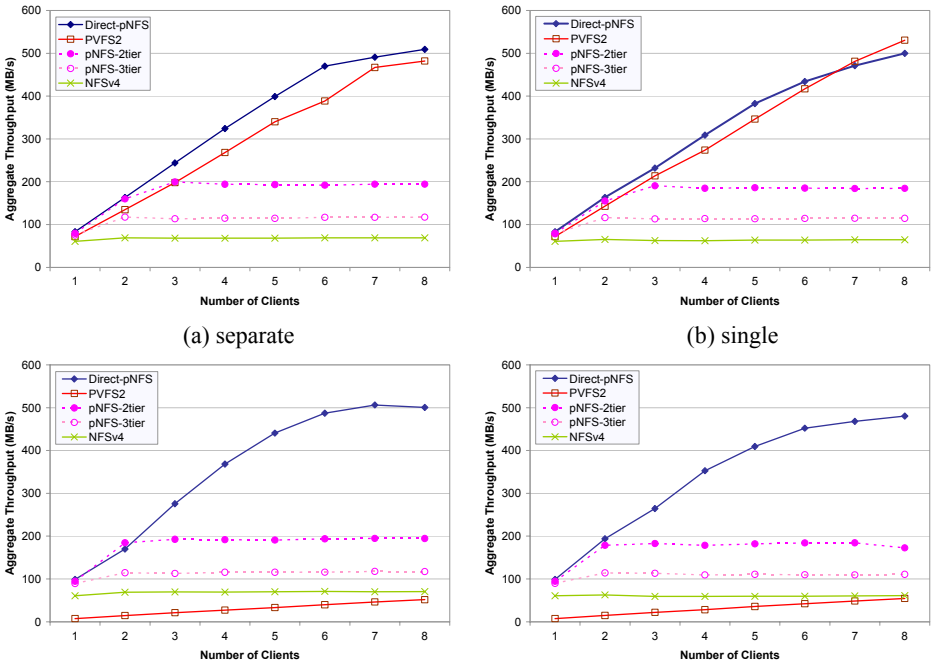
6.4. Synthetic Workloads

This section uses two macro-benchmarks to analyze Direct-pNFS in a more general setting.

6.4.1. OLTP

OLTP models a database workload as a series of transactions on a single large file. Each transaction consists of a random 8 KB read, modify, and write. Each client performs 20,000 transactions, with data sent to stable storage after each transaction.

Figure 8c displays OLTP experimental results. Direct-pNFS scales well, achieving 26 MB/s with eight clients. As expected, PVFS2 performs poorly with small I/O requests, achieving an aggregate I/O throughput of 6 MB/s.



(a) separate (b) single

(c) separate – 8 KB (d) single – 8 KB

Figure 7. Aggregate read throughput. (a) With separate files and a large block size, Direct-pNFS outperforms PVFS2 for some numbers of clients. pNFS-2tier and pNFS-3tier are bandwidth limited due to a lack of direct file access. NFSv4 is bandwidth and CPU limited. (b) With a single file and a large block size, PVFS2 eventually outperforms Direct-pNFS due to a prototype software limitation. pNFS-2tier and pNFS-3tier are bandwidth limited due to a lack of direct file access. NFSv4 is CPU limited. (c) and (d) With a separate or single file and an 8 KB block size, all NFSv4 architectures outperform PVFS2.

6.4.2. Postmark

The Postmark benchmark simulates metadata and small I/O intensive applications such as electronic mail, NetNews, and Web-based services [29]. Postmark performs transactions on a large number of small randomly sized files (between 1 KB and 500 KB). Each transaction first deletes, creates, or opens a file, then reads or appends 512 bytes. Data are sent to stable storage before the file is closed. Postmark performs 2,000 transactions on 100 files in 10 directories. All other parameters are left as default. To ensure a more even distribution of requests among the storage nodes, we reduce the stripe size, wsize, and rsize to 64 KB.

The Postmark experiments are shown in Figure 8d, with results given in transactions per second. Direct-pNFS again leverages the asynchronous, multi-threaded Linux NFSv4 implementation, designed for small I/O intensive workloads like Postmark, to perform up to 36 times as many transactions per second as PVFS2.

6.4.3. Discussion

This set of experiments demonstrates that Direct-pNFS performance compares well to the exported parallel file system with the large I/O scientific application benchmark BTIO. Direct-pNFS performance for ATLAS, for which 95% of the I/O requests are smaller than 275 KB, far surpasses native file system performance. The Postmark and OLTP benchmarks, where small I/O requests also dominate, yield similar results.

A natural next step is to explore performance with routine tasks such as a build/development environment. Following the SSH build benchmark [30], we created a benchmark that uncompresses, configures, and builds OpenSSH [31]. Using the same systems as above, we find that Direct-pNFS reduces compilation time, a stage heavily dominated by small read and write requests, but increases uncompress and configure time, stages dominated by file creates and attribute updates.

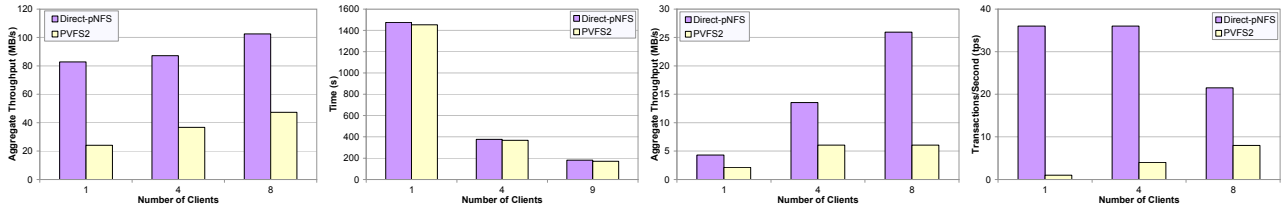
Tasks like file creation—relatively simple for standalone file systems—become complex on parallel file systems. Consequently, some parallel file systems distribute metadata across many nodes and have clients gather and reconstruct the information, relieving the overloaded metadata server. NFSv4 relies on a central metadata server, effectively recentralizing the decentralized parallel file system metadata protocol. This paper does not focus on improving metadata performance, but the sharp contrast in metadata management technique between NFSv4 and parallel file systems merits further study.

7. RELATED WORK

Unlike much recent work that focuses on improving the performance and scalability of a *single* file system, e.g., GPFS-WAN [32, 33], Google file system [34], Gfarm [35], and FARSITE [36], the goal of Direct-pNFS is to enhance a commodity protocol to scale I/O throughput to a *diversity* of parallel file systems.

Several file systems aggregate NFS servers into a single file system image [37-39]. Direct-pNFS generalizes these architectures, making them independent of the underlying parallel file system.

NFS-CD [40] uses NFSv4 delegations and cooperative caching to enable client data sharing without server involvement. NFS-CD and Direct-pNFS address different but related issues. NFS-CD



(a) ATLAS (b) BTIO (c) OLTP (d) Postmark

Figure 8. (a) ATLAS. Direct-pNFS outperforms PVFS2 with a small and large write request workload. **(b) BTIO.** Direct-pNFS and PVFS2 achieve comparable performance with a large read and write workload. Lower time values are better. **(c) OLTP.** Direct-pNFS outperforms PVFS2 with an 8 KB read-modify-write write request workload. **(d) Postmark.** Direct-pNFS outperforms PVFS2 in a small read and append workload.

can improve the effective runtime of an application, but clients must access physical storage at some point. Cooperative caching can be used together with the increased I/O throughput of Direct-pNFS, combining the performance benefits of both approaches.

GridFTP [41] is used extensively in the Grid to enable high throughput, operating system independent, and secure remote access to parallel file systems. Successful and popular, GridFTP nevertheless has some serious limitations: it copies data instead of providing shared access to a single copy, complicating its consistency model and decreasing storage capacity; lacks a global namespace; and is difficult to integrate with the local file system.

The Storage Resource Broker [42] aggregates storage resources into a single data catalogue, but does not support parallel I/O to multiple storage endpoints and uses a customized interface.

A PVFS2 layout driver has existed since 2004 [8] and file-based layout drivers have been demonstrated with GPFS, Lustre, and PVFS2. Panasas object and EMC block drivers are also under development. Earlier pNFS research improved overall write performance by using direct, parallel I/O for large write requests and a distributed file access protocol for small write requests [17]. This technique still benefits file systems whose storage nodes cannot interpret the NFSv4 storage protocol. Direct-pNFS uses a single storage protocol, allowing more efficient request gathering, and eliminates the single server bottleneck.

8. CONCLUDING REMARKS

Universal, transparent, and scalable remote data access is a critical enabling feature of widely distributed collaborations. Existing remote data access technologies such as NFS, pNFS, and GridFTP fail to satisfy *all three* of these requirements.

Direct-pNFS, on the other hand, *satisfies* these requirements by enhancing the portability and transparency of pNFS. Direct-pNFS enables a stock NFSv4.1 client to support high-performance remote data access to different parallel file systems. Experiments demonstrate that Direct-pNFS matches the I/O throughput of the specialized parallel file system that it exports. Furthermore, Direct-pNFS "scales down" to outperform the parallel file system client in diverse workloads.

ACKNOWLEDGEMENTS

This material is based upon work supported by the Department of Energy under Award Numbers DE-FG02-06ER25766 and B548853, Sandia National Labs under contract B523296, and by grants from Network Appliance and IBM. We thank Lee Ward, Gary Grider, James Nunez, Marc Eshel, Garth Goodson, Benny Halvey, and the PVFS2 development team for their valuable insights and system support.

DISCLAIMER

REFERENCES

[1] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Schlatter Ellis, and M. Best, "File-Access Characteristics of Parallel Scientific Workloads," *IEEE Trans. on Parallel and Distributed Systems*, 7(10):1075-1089, 1996.

[2] P.E. Crandall, R.A. Aydt, A.A. Chien, and D.A. Reed, "Input/Output Characteristics of Scalable Parallel Applications," in *Proc. of Supercomputing '95*, San Diego, CA, 1995.

[3] F. Wang, Q. Xin, B. Hong, S.A. Brandt, E.L. Miller, D.D.E Long, and T.T. McLarty, "File System Workload Analysis For Large Scale Scientific Computing Applications," in *Proc. of the 21st IEEE/12th NASA Goddard Conf. on Mass Storage Systems and Technologies*, College Park, MD, 2004.

[4] D. Strauss, "Linux Helps Bring Titanic to Life," *Linux J.*, 46, 1998.

[5] B. Callaghan, B. Pawlowski, and P. Staubach, NFS Version 3 Protocol Specification. RFC 1813, 1995.

[6] Common Internet File System File Access Protocol (CIFS), `msdn.microsoft.com/library/en-us/cifs/protocol/cifs.asp`.

[7] S. Shepler, M. Eisler, and D. Noveck, NFSv4 Minor Version 1. Internet Draft, 2006.

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

[9] B. Halevy, B. Welch, and J. Zelenka, Object-based pNFS Operations. Internet Draft, 2007.

[10] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout, "Measurements of a Distributed File System," in *Proc. of the 13th ACM Symp. on Operating Systems Principles*, Pacific Grove, CA, 1991.

[11] R.O. Weber, SCSI Object-Based Storage Device Commands (OSD). Storage Networking Industry Association. ANSI/INCITS 400-2004, `www.t10.org`, 2004.

[12] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, Internet Small Computer Systems Interface (iSCSI). RFC 3720, 2001.

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

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

[15] D.L. Black, S. Fridella, and J. Glasgow, pNFS Block/Volume Layout. Internet Draft, 2007.

[16] D. Hildebrand and P. Honeyman, "Scaling NFSv4 with Parallel File Systems," in *Proc. of Cluster Computing and Grid*, Cardiff, UK, 2005.

[17] D. Hildebrand, L. Ward, and P. Honeyman, "Large Files, Small Writes, and pNFS," in *Proc. of the 20th ACM Intl. Conf. on Supercomputing*, Cairns, Australia, 2006.

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

[19] Red Hat Software Inc., "Red Hat Global File System," `www.redhat.com/software/rha/gfs`.

[20] S.R. Soltis, T.M. Ruwart, and M.T. O'Keefe, "The Global File System," in *Proc. of the 5th NASA Goddard Conf. on Mass Storage Systems*, College Park, MD, 1996.

[21] Polyserve Inc., "Matrix Server Architecture," `www.polyserve.com`.

[22] Parallel Virtual File System - Version 2, `www.pvfs.org`.

[23] IBRIX Fusion, `www.ibrix.com`.

[24] S.V. Anastasiadis, K. C. Sevcik, and M. Stumm, "Disk Striping Scalability in the Exedra Media Server," in *Proc. of the ACM/SPIE Multimedia Computing and Networking*, San Jose, CA, 2001.

[25] D.A. Patterson, G.A. Gibson, and R.H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *Proc. of the ACM SIGMOD Conf. on Management of Data*, Chicago, IL, 1988.

[26] F. Isaila and W.F. Tichy, "Clusterfile: A Flexible Physical Layout Parallel File System," in *Proc. of the IEEE Intl. Conf. on Cluster Computing*, Newport Beach, CA, 2001.

[27] IOR Benchmark, `www.llnl.gov/asci/purple/benchmarks/limited/ior`.

[28] ATLAS, `atlasinfo.cern.ch`.

[29] J. Katcher, "PostMark: A New File System Benchmark," Network Appliance, Technical Report TR3022, 1997.

[30] M. Seltzer, G. Ganger, M.K. McKusick, K. Smith, C. Soules, and C. Stein., "Journaling versus Soft Updates: Asynchronous Meta-data Protection in File Systems.," in *Proc. of the USENIX Annual Technical Conf.*, San Diego, CA, 2000.

[31] OpenSSH, `www.openssh.org`.

[32] P. Andrews, C. Jordan, and W. Pfeiffer, "Marching Towards Nirvana: Configurations for Very High Performance Parallel File Systems," in *Proc. of the HiperIO Workshop*, Barcelona, Spain, 2006.

[33] P. Andrews, C. Jordan, and H. Lederer, "Design, Implementation, and Production Experiences of a Global Storage Grid," in *Proc. of the 23rd IEEE/14th NASA Goddard Conf. on Mass Storage Systems and Technologies*, College Park, MD, 2006.

[34] S. Ghemawat, H. Gobioff, and S.T. Leung, "The Google File System," in *Proc. of the 19th ACM Symp. on Operating Systems Principles*, Bolton Landing, NY, 2003.

[35] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid Datafarm Architecture for Petascale Data Intensive Computing," in *Proc. of the 2nd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid*, Berlin, Germany, 2002.

[36] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R.P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," in *Proc. of the 5th Symp. on Operating Systems Design and Implementation*, Boston, MA, 2002.

[37] G.H. Kim, R.G. Minnich, and L. McVoy, "Bigfoot-NFS: A Parallel File-Striping NFS Server (Extended Abstract)," 1994, `www.bitmover.com/lm`.

[38] F. Garcia-Carballeira, A. Calderon, J. Carretero, J. Fernandez, and J.M. Perez, "The Design of the Expand File System," *Intl. J. of High Performance Computing Applications*, 17(1):21-37, 2003.

[39] P. Lombard and Y. Denneulin, "nfsp: A Distributed NFS Server for Clusters of Workstations," in *Proc. of the 16th Intl. Parallel and Distributed Processing Symp.*, Fort Lauderdale, FL, 2002.

[40] A. Batsakis and R. Burns, "Cluster Delegation: High-Performance Fault-Tolerant Data Sharing in NFS," in *Proc. of the 14th IEEE Intl. Symp. on High Performance Distributed Computing*, 2005.

[41] B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke., "Data Management and Transfer in High-Performance Computational Grid Environments," *Parallel Computing*, 28(5):749-771, 2002.

[42] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," in *Proc. of the Conf. of the Centre for Advanced Studies on Collaborative Research*, Toronto, Canada, 1998.