

Deploying pNFS across the WAN: First Steps in HPC Grid Computing

Dean Hildebrand, Marc Eshel, Roger Haskin
IBM Almaden Research Center
{dhildeb, eshel, fku}@us.ibm.com

Patricia Kovatch, Phil Andrews
National Institute for Computational Sciences
{pkovatch, andrewspl}@utk.edu

John White
Revision3
jwhite@revision3.com

Abstract. Global file systems promise great advantages in convenience, performance, and ubiquitous access over more traditional data movement mechanisms for HPC grids. Unfortunately, their widespread acceptance is limited due to licensing issues and lack of interoperability. Using the pNFS extension to the popular NFS standard, we performed the first large-scale test of non-proprietary clients accessing GPFS servers within a real HPC Grid environment. Extremely high transfer rates across the TeraGrid Wide Area Network were achieved and we see this as a possible harbinger of a much wider applicability for global file systems.

Introduction

High performance grid computing can have very different requirements than the more common and computationally less demanding grid work. Although the size of the computational resources requested by a grid job can present its own difficulties, an onerous difficulty is satisfying the associated data requests. In addition to providing access to very large input datasets, the output dataset, which has unpredictable characteristics, must be handled successfully or the whole process may be for naught.

Traditionally, the input dataset moves to the chosen computational resource *in toto*, e.g., via GridFTP [1], which presupposes that sufficient space is available. A similar assumption is necessary for the output data, which also requires a move to some permanent home. An error at any point can lead to job termination, with possible loss of completed work. These concerns have led to great interest by both users and system administrators in global file systems. A global file system greatly simplifies the process by allowing all I/O operations to occur as if to a local resource. High performance grid computing is particularly amenable to this approach, which generally transfers sufficiently large datasets such that latency concerns are not paramount.

Given this background, an experimental global file system based on IBM's GPFS [2] software was first implemented [3] in 2004. Initial technical results were good enough that the decision to continue with a production facility was made, with implementation [4] in 2005. The NSF TeraGrid [5] is committed to providing a global file system and this work continues our efforts.

Although transfer rates and latencies [3] were definitely favorable for widespread adoption within the TeraGrid, other obstacles fettered progress. IBM's GPFS software is proprietary, requiring users to obtain a license. Although some TeraGrid sites use GPFS with no licensing issues, this is not true for all, or even most sites. In addition, restricting clients to a single back end file system limits the global file system's reach within the high performance grid community.

With this work, we are prototyping a new approach, which we hope will lead to a much wider implementation of global file systems, both within TeraGrid and within other grids. The paradigm has sites use pNFS to communicate with remote servers, allowing only the organization providing the back end file system servers to deal with proprietary issues. pNFS clients will be either open-source, or provided by the client site's OS vendor, much as with NFS clients today.

The work described in this paper deals with the first large-scale experiment using this approach, serving to provide both technical input and experience for the grid community.

pNFS overview

This section summarizes the pNFS architecture. A full description can be found elsewhere [6, 7].

NFSv4 provides transparent access to files and directories; replaces NFSv3's troublesome lock and mount protocols; mandates strong and diverse security mechanisms; and supports scalable and consistent client caching and internationalization.

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 1a displays the pNFS architecture. The control path contains all NFSv4.1 operations and features, continuing to use RPCSEC_GSS for authentication and NFSv4 access control lists (a super set of POSIX access control lists) for authorization. The data path can support any storage protocol, but the IETF design effort focuses on file-, object-, and block-based storage protocols. Storage nodes can be NFSv4.1 servers, object storage, or even block-addressable storage area networks. NFSv4.1 does not specify a management protocol, which may therefore be proprietary to the exported file system.

pNFS encapsulates the client I/O subsystem in a *layout driver*. This facilitates interoperability by providing a framework for the co-existence of the NFSv4.1 control

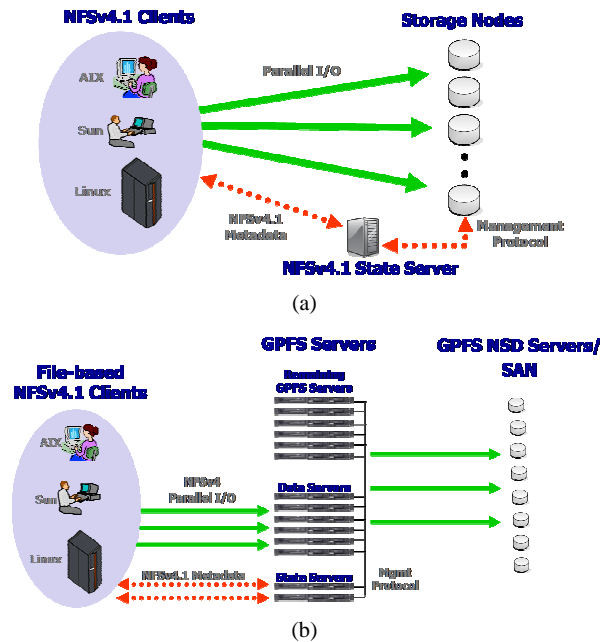


Fig. 1. (a) pNFS Architecture – pNFS splits the NFSv4.1 protocol into a control path and a data path. The NFSv4.1 protocol exists along the control path. A storage protocol along the data path provides direct and parallel data access. A management protocol binds metadata servers with storage devices. **(b) pNFS with GPFS** – GPFS servers are divided into (possibly overlapping) groups of one or more state servers, several data servers, and any number of non-pNFS related servers. NFSv4.1 clients use the state servers for metadata operations and use the file-based layout to perform parallel I/O to the data servers. Note that, in isolation, the remaining GPFS servers, GPFS NSD servers, and the SAN represent the standard GPFS shared-disk architecture.

protocol with all storage protocols. This is a major departure from current file systems, which can support only a single storage protocol such as OSD [8, 9].

To perform direct and parallel I/O, a pNFS client first requests layout information from the pNFS server. A layout contains the information required to access any byte range of a file. The layout driver uses the information to translate read and write requests from the pNFS client into I/O requests directed to storage nodes. For example, the NFSv4.1 file-based storage protocol stripes files across NFSv4.1 data servers with only READ, WRITE, COMMIT, and session operations sent on the data path. The pNFS state server can generate layout information itself or request assistance from the underlying file system.

pNFS with GPFS

The General Parallel File System (GPFS) is a symmetric parallel file system that scales to extremely large clusters with petabytes of disk space. GPFS successfully satisfies the needs for throughput, storage capacity, and reliability of the largest and most demanding problems.

The symmetric architecture of GPFS allows highly available and flexible distributed data access. IBM Clustered-NFS [10] and Scale Out File Services [11] uses NFSv3 and CIFS to provide a highly scalable, global, clustered NAS system on top of GPFS. Both systems also include automatic failover support to transfer client load from the failing node to another node in the GPFS cluster.

pNFS with GPFS goes beyond these scalable NAS solutions by also targeting the HPC community. Enabling NFS clients for parallel I/O to GPFS servers increases individual and aggregate I/O performance. The use of multiple symmetric servers allows pNFS to balance small I/O requests across the entire GPFS cluster. In addition, pNFS with GPFS eliminates the NFSv4 single metadata server restriction. A single GPFS file system can support any number of NFSv4.1 metadata servers, increasing the scalability of metadata operations such as file create and open.

Figure 1b displays the pNFS-GPFS architecture. The nodes in the GPFS cluster chosen for pNFS access are divided into (possibly overlapping) groups of state and data servers. pNFS client metadata requests are partitioned among the available state servers while I/O is distributed across all of the data servers. The GPFS management protocol maintains the freshness of NFSv4 state information among servers.

GPFS supports the file-based storage protocol, although the IETF is also specifying an object and block storage protocol. The NFSv4.1 protocol includes only the file-based storage protocol, with object and block to follow in separate specifications [12, 13]. As such, all NFSv4.1 implementations will support the file-based storage protocol, while support for object and block storage protocols will be optional. By supporting the file-based storage protocol, GPFS allows all NFSv4.1 implementations to experience increased performance.

A file-based layout governs an entire file and is valid until recalled by the pNFS server. 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 in flexibility in how it generates the layout information. In addition, GPFS can rebalance data across the disks without needing to recall and generate new layout information for pNFS clients. The current GPFS prototype uses round-robin striping and continually alternates the first data server of the stripe.

To perform I/O, clients first obtain the file layout map from a metadata server. Clients access GPFS by using the layout information to send NFSv4 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 the metadata server. This single COMMIT operation flushes data to stable storage on every data server.

Although not the focus of this paper, several issues arise when using one storage protocol (NFSv4) in conjunction with another (GPFS). One major issue is handling failures of clients, servers, and their associated daemons. NFS and GPFS need to communicate failure information so each one can recover in a correct and graceful

manner. Another issue is handling I/O requests for a single file on multiple servers. Parallel file systems may assume I/O requests on separate servers are from separate pNFS clients, causing management overhead to ensure file system consistency. Even multiple NFS server threads can also reduce I/O throughput by processing read and write requests out of order, hampering the parallel file system's ability to improve its interaction with the physical disk.

Saturating Long-Fat Pipes

This section discusses using NFSv4 and pNFS in 10 Gigabit networks and over long distances. The challenge of filling long-fat networks is particularly difficult for NFS, a protocol designed for small I/O over local networks.

10-Gigabit Networks

Realizing the potential of 10 Gigabit networks is a major challenge [14-16]. The existence of the Bandwidth Challenge exemplifies this complexity. Most operating systems are struggling to identify and overcome all of the newly created bottlenecks, e.g., bus bandwidth, number of interrupts, TCP congestion control, and cache management.

Beyond operating system network scalability issues, NFS implementations optimize processing large number of small I/O requests and are not designed to have a large amount of outstanding data requests. One major problem is a relatively small maximum I/O request size (`rsize` and `wsizes`) combined with a limited number of concurrent I/O requests. A NFS client is limited to $(\text{MAX REQUEST SIZE}) \times (\# \text{ OUSTANDING REQUESTS})$ amount of outstanding data at any point in time, which in many cases does not fill 10 Gigabit pipes. Another major problem is the NFS client-server architecture, which limits each client to a single server. Achieving 10 Gb/s with a single endpoint is difficult due to congestion control algorithms, networking-related issues, network adapter drivers, and switch buffer space [15]. pNFS helps to alleviate both of these issues by empowering clients with parallel I/O. With multiple servers (or endpoints), the maximum number of outstanding bytes is $(\text{MAX REQUEST SIZE}) \times (\# \text{ OUSTANDING REQUESTS}) \times (\# \text{ SERVERS})$. Simply increasing the number of servers can allow clients to fill their 10 Gigabit pipe.

Wide Area Networks

Petascale grid computing requires inter-site data transfers involving clusters that may have different operating systems and hardware platforms and incompatible or proprietary file systems. The network bandwidth linking these sites is continuing to increase, with 10 Gigabit links now common [5, 15, 16]. While the chatty nature of NFSv3 can hamper WAN performance, NFSv4 includes many WAN friendly features

such as delegations and compound operations, making it a natural candidate for heterogeneous data transfer between sites.

Unfortunately, the long round-trip time and large bandwidth-delay product in the WAN can reduce NFSv4 I/O throughput. Beyond the known difficulties of using TCP with a long round-trip time [15, 17, 18], the Linux NFSv4 server implementation further complicates the issue by optimizing TCP buffer sizes for LAN environments. This is a consequence of using the same value for buffer space and the TCP congestion window. While the server requires $(\text{MAX_REQUEST_SIZE}) \times (\# \text{ NFS_SERVER_DAEMONS})$ amount of buffer space to use for request processing, the TCP congestion window in long fat pipes can be much larger and should be based on the bandwidth-delay product. In addition, the large bandwidth-delay product exacerbates the single-network endpoint issues previously mentioned. This is because in order to fill the long fat pipe, we must increase either the maximum request size or the maximum number of outstanding requests.

pNFS improves the situation by using multiple endpoints to fill the long-fat pipe, but this can have unintended consequences. For example, due to the large bandwidth-delay product in long fat pipes, large amounts of data must be transferred before the TCP congestion window will fully open. Until the window opens all the way, the application will experience only a fraction of the available bandwidth. In pNFS, which has a separate TCP congestion window for each data server, clients must transfer an even larger amount of data to fully open each TCP congestion window.

Supercomputing 2007 Bandwidth Challenge

This section describes the results of the joint IBM and San Diego Supercomputing Center entry into the Bandwidth Challenge at Supercomputing 2007 in Reno, Nevada. Participants attempt to fill a 10 Gigabit SCinet link between the Supercomputing 2007 showroom floor and a remote site for fifteen minutes.

Our pNFS entry achieved the highest read performance. In addition, our use of pNFS is unique among bandwidth challenge participants, which all used an established product for data transfer.

Bandwidth Challenge Participants

The Bandwidth Challenge attracts a wide-range of participants from a variety of research institutions. The common goal of all entries is to demonstrate the ability of an application to saturate high-bandwidth, long-distance links. Most of the entries are application centric, and used commodity middleware tools for data transfer. The key difference of our entry is the focus on the file system. Our goal is to provide the middleware that Bandwidth Challenge participants will rely upon in the future.

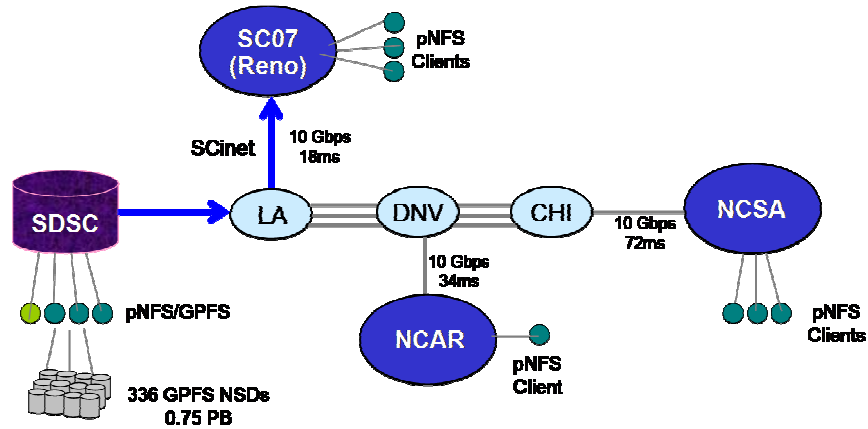


Fig. 2. pNFS in the Teragrid – In preparation for Supercomputing 2007, several 10 Gigabit pNFS clients were setup across the Teragrid. At Supercomputing 2007, three 10 Gigabit pNFS clients were setup to use SCinet. At SDSC, one state server and three data servers exported a production GPFS file system. Times indicate round-trip time to SDSC. Arrows indicate data flow during bandwidth challenge.

Setup

As shown in Figure 2, our bandwidth challenge entry used three pNFS clients in Reno and three data servers and a single state server at SDSC. The exported GPFS file system is a production file system at SDSC and consists of 336 network storage devices with a capacity of 0.75 petabytes. All nodes are equipped with two 2.6 GHz dual-core Opteron processors, 8 GB memory, and one Sun 10 Gigabit card. The round-trip time from Reno to San Diego is 18 ms. All nodes run the Linux 2.6.18-3 pNFS prototype and use the default Linux TCP congestion control algorithm, *bic*.

To generate read requests, we continuously executed a 30 GB checkpoint file ingest simulator on the three pNFS clients. The large file ensures all data is retrieved from disk at the back end, i.e., a disk-to-memory experiment. Once the simulator begins execution, challenge organizers measure I/O throughput for fifteen minutes.

pNFS Prototype Tuning

This section describes the tuning parameters that worked for our nodes on the SCinet. Tuning the network performance of an application is difficult in the best of times. Possibly the most difficult part of the Bandwidth Challenge is installing, configuring, and tuning a system in a few short days. Throw in a few network outages, severe network contention, and a production file system, and tuning becomes more of an art than a science.

Ensuring the correct TCP congestion window for the path from SDSC to SCinet was critical. The TCP congestion window essentially controls the upper limit that pNFS can attempt to reach. Although it is common to set the TCP buffer size to twice

```

net.ipv4.tcp_timestamps          = 0
net.ipv4.tcp_sack                 = 1
net.ipv4.tcp_no_metrics_save     = 1

# buffer value = min default max
net.ipv4.tcp_rmem = 4096 33554432 33554432
net.ipv4.tcp_wmem = 4096 33554432 33554432
net.ipv4.tcp_mem  = 4096 33554432 33554432

net.core.rmem_max      = 33554432
net.core.wmem_max      = 33554432
net.core.rmem_default  = 33554432
net.core.wmem_default  = 33554432
net.core.optmem_max    = 33554432

net.core.netdev_max_backlog = 30000

```

Fig. 3. Supercomputing 2007 Bandwidth Challenge Linux TCP settings. The NFS client uses the default TCP buffer setting (middle value). The NFS server sets its own value based on the number of NFSD daemons and the maximum request size.

the bandwidth-delay product [17] (18 MB in our case), we found better performance with a higher value on both the client and server of 30 MB. In addition, we found using SACK critical to achieving good read performance.

To set the pNFS client TCP congestion window, we used the TCP parameters in Figure 3. It is important to note that NFS uses the default, not the maximum, TCP buffer setting. Also, note that optimizing the global TCP settings for the WAN can affect network applications communicating within the LAN. For example, it is impossible to achieve good performance to NFS servers in both the LAN and the WAN. As such, it seems TCP settings should be set per mount point, not set for the entire machine.

As mentioned previously, the NFS server TCP congestion window is permanently set to $(\text{MAX REQUEST SIZE}) \times (\# \text{ NFS SERVER DAEMONS})$. With a 1 MB maximum request size and 32 NFSDs, we reached our 30 MB congestion window target. As an aside, it was necessary to modify the Linux NFS server to decouple the value of NFSD buffer space and the value of the TCP congestion window to achieve good performance where a larger bandwidth-delay product exists, e.g., NCSA to SDSC in Figure 2.

The data servers use 32 NFS daemons and the clients can have 48 outstanding requests. Increasing the number of daemons beyond 32 actually reduces I/O performance, most likely due to the increased randomness of the I/O requests. Similarly, allowing more than 48 outstanding requests reduces performance through increased packet loss.

Another Linux parameter we increased was `min_free_kbytes`, which forces the Linux VM to keep a minimum amount of available memory. The increase is vital since a 10 Gigabit card can quickly exhaust available memory. Subsequently we found that data transfers were blocked waiting for Linux to free memory in the page cache. We later found out that this was a known problem with the Linux 2.6.18

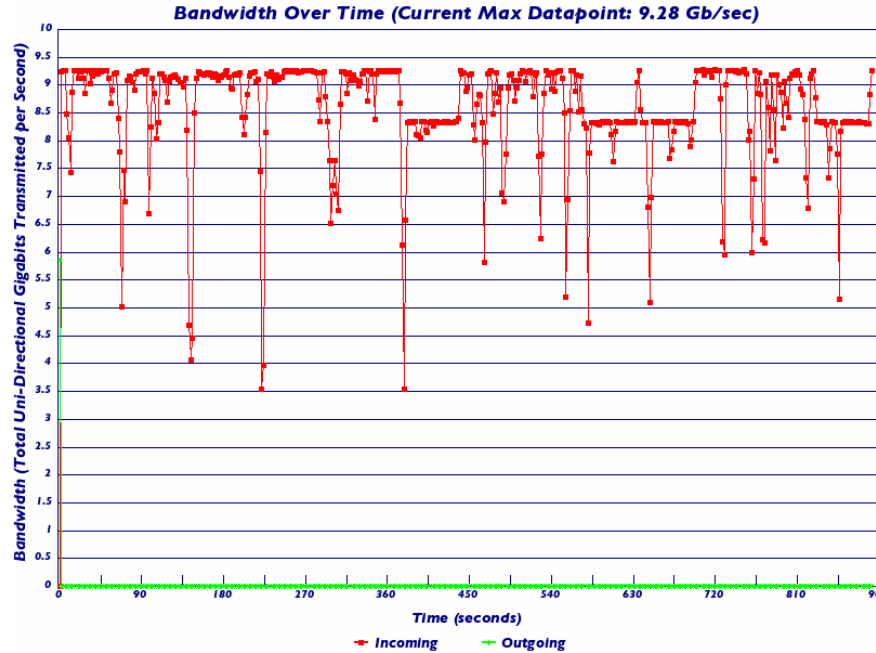


Fig. 4. Supercomputing 2007 Bandwidth Challenge Aggregate Read Performance – pNFS clients in Reno continuously read a 30 GB file from a production GPFS cluster at SDSC. pNFS prototype achieved highest read bandwidth among participants. TCP protocol overhead is not included in measured bandwidth. Setup included three 10 Gigabit clients in Reno and three 10 Gigabit data servers and one state server at SDSC. Write performance not measured.

kernel. In the end, to avoid Linux page cache eviction delays, and increase our performance by avoiding extra data copies, we used `O_DIRECT`.

Finally, although we found a slight performance increase when using two threads to ingest the data, the three pNFS clients could easily exhaust the SCinet 10 Gigabit network with a single thread.

Results

Figure 3 displays the aggregate pNFS read throughput during the bandwidth challenge. pNFS maintains its peak bandwidth of 9.28 Gb/s for 70% of the challenge, which is the highest of any bandwidth challenge participant. Considering that Figure 3 only measures transferred payload data, and does not include TCP protocol overhead, 9.28 Gb/s is a practical ceiling on the 10 Gigabit Ethernet SCinet network hardware. During the remaining time, bandwidth dropped to 8.4 Gb/s due to competing load on the SDSC production GPFS cluster. The frequent short drops in bandwidth are the result of the pNFS clients pushing the limits of the available hardware and possible contention within the non-dedicated TeraGrid network. As

switch and other hardware devices along the path drop packets, TCP reduces the congestion window, causing a drop in bandwidth.

Related work

A fair amount of experience exists with optimizing 10-Gigabit Ethernet over wide area networks [14-16]. Most work focuses solely on optimizing TCP performance and performs measurements using Iperf [19] or Netperf [20]. Cope et al. [16] performed additional file transfer experiments over the WAN using tools such as GridFTP [1]. This work provides the foundation to start down the path of tuning distributed file systems—with their complex data sharing protocols—for long-fat links.

Much work focuses on improving the performance and scalability of a *single* file system, e.g., GPFS [2], GPFS-WAN [4, 21], Lustre [22], PVFS2 [23], Google file system [24], Gfarm [25], and FARSITE [26]. Although this paper focuses on pNFS with GPFS, the goal of pNFS is to enhance a single file access protocol to scale I/O throughput to a *diversity* of parallel file systems.

The Storage Resource Broker (SRB) [27] aggregates storage resources, e.g., a file system, an archival system, or a database, into a single data catalogue but does not support parallel I/O to multiple storage endpoints and has difficulty integrating with the local file system.

GridFTP [28] is used extensively in the Grid to enable high throughput, operating system independent, and secure WAN access to high-performance 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.

It is interesting to note that pNFS can work alongside GridFTP. In tiered projects such as ATLAS, GridFTP could handle scheduled transfers among the upper tiers, while the file system semantics of pNFS offers advantages in the lower tiers. Domain scientists can then use pNFS to work directly with files using conventional names.

Conclusions

Within this paper, we have shown the results of the first large-scale experimental implementation of pNFS clients operating with GPFS servers in a real High Performance Computing grid environment. While the write performance has yet to be demonstrated (and initial results are very positive), the transfer rates achieved are extremely impressive and bode well for pNFS as a usable paradigm. We intend to proceed with more TeraGrid experiments with the goal of exploring this mixed-mode Global File System as a possible ubiquitous data access mechanism.

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] 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.
- [3] P. Andrews, B. Banister, P. Kovatch, and R. Haskin, "Scaling a Global File System to the Greatest Possible Extent, Performance, Capacity, and Number of Users," in *Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*, Monterey, CA, 2005.
- [4] P. Andrews, C. Jordan, and H. Lederer, "Design, Implementation, and Production Experiences of a Global Storage Grid," in *Proceedings of the 23rd IEEE/14th NASA Goddard Conference on Mass Storage Systems and Technologies*, College Park, MD, 2006.
- [5] C. Catlett, "The TeraGrid: A Primer," www.teragrid.org, 2002.
- [6] 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.
- [7] S. Shepler, M. Eisler, and D. Noveck, "NFSv4 Minor Version 1," Internet Draft, 2006.
- [8] Panasas Inc., "Panasas ActiveScale File System," www.panasas.com.
- [9] EMC Celerra HighRoad Whitepaper, www.emc.com, 2001.
- [10] 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.
- [11] IBM Corp., "IBM Storage Optimization and Integration Services—scale out file services," Datasheet, 2007.
- [12] B. Halevy, B. Welch, and J. Zelenka, "Object-based pNFS Operations," Internet Draft, 2007.
- [13] D.L. Black, S. Fridella, and J. Glasgow, "pNFS Block/Volume Layout," Internet Draft, 2007.
- [14] W. Feng, J. Hurwitz, H. Newman, S. Ravot, R. Les Cottrell, O. Martin, F. Coccetti, C. Jin, X. Wei, and S. Low, "Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters, and Grids: A Case Study," *Proceedings of Supercomputing '03*. Phoenix, AZ, 2003.
- [15] C. Meirosu, P. Golonka, A. Hirstius, S. Stancu, B. Dobinson, E. Radius, A. Antony, F. Dijkstra, J. Blom, and C. Laat, "Native 10 Gigabit Ethernet Experiments Over Long Distances," *Future Generation Computer Systems*, (21)4, pp. 457-468, 2005.
- [16] J. Cope, T. Voran, M. Woitaszek, A. Boggs, S. McCreary, M. Oberg, and H.M. Tufo, "Experiences Deploying a 10 Gigabit Ethernet Computing Environment to Support Regional Computational Science," in *Proceedings of the 8th LCI International Conference on High-Performance Clustered Computing*, Lake Tahoe, CA, 2007.
- [17] B. Tierney, "TCP Tuning Guide for Distributed Application on Wide Area Networks," *Usenix ;login*, (26)1, 2001.
- [18] D.X. Wei, C. Jin, S.H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, (14)6, pp. 1246-1259, 2006.
- [19] Iperf - The TCP/UDP Bandwidth Measurement Tool, dast.nlanr.net/Projects/Iperf, 2008.
- [20] Netperf, www.netperf.org, 2008.

- [21] P. Andrews, C. Jordan, and W. Pfeiffer, "Marching Towards Nirvana: Configurations for Very High Performance Parallel File Systems," in *Proceedings of the HiperIO Workshop*, Barcelona, Spain, 2006.
- [22] Cluster File Systems Inc., "Lustre: A Scalable, High-Performance File System," www.lustre.org, 2002.
- [23] Parallel Virtual File System - Version 2, www.pvfs.org, 2008.
- [24] S. Ghemawat, H. Gobioff, and S.T. Leung, "The Google File System," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, 2003.
- [25] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid Datafarm Architecture for Petascale Data Intensive Computing," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, Germany, 2002.
- [26] 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 *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, 2002.
- [27] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Canada, 1998.
- [28] 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, pp. 749-771, 2002.