# Design, Implementation, and Evaluation of Transparent pNFS on Lustre *

Weikuan Yu† Oleg Drokin‡ Jeffrey S. Vetter †

Computer Science & Mathematics†
Oak Ridge National Laboratory
{wyu,vetter}@ornl.gov

Lustre Group ‡
Sun Microsystems, Inc.
{Oleg.Drokin}@Sun.com

## Abstract

*Parallel NFS (pNFS) is an emergent open standard for parallelizing data transfer over a variety of I/O protocols. Prototypes of pNFS are actively being developed by industry and academia to examine its viability and possible enhancements. In this paper, we present the design, implementation, and evaluation of lpNFS, a Lustre-based parallel NFS. We achieve our primary objective in designing lpNFS as an enabling technology for transparent pNFS accesses to an opaque Lustre file system. We optimize the data flow paths in lpNFS by using two techniques: (a) fast memory coping for small messages, and (b) page sharing for zero-copy bulk data transfer. Our initial performance evaluation shows that the performance of lpNFS is comparable to that of original Lustre. Given these results, we assert that lpNFS is a promising approach to combining the benefits of pNFS and Lustre, and it exposes the underlying capabilities of Lustre file systems while transparently supporting pNFS clients.*

Keywords: *pNFS, Lustre, NFSv4*

## 1. Introduction

Since its inception in the mid-80's, the Network File System (NFS) [22] protocol has been ubiquitously accepted as a means for sharing files across a network of compute nodes and across many operating systems simultaneously. Despite stiff competition from recent file systems [31, 10, 3], incarnations of NFS endure due to its reliability, reasonable efficiency, and ease of deployment. Arguably, it has been most widely deployed on a great number of architectures and platforms, ranging from workstations to high-end server farms.

On the other hand, data-intensive applications, such as those in genomics, climate modeling, molecular dynamics, and high performance computing, are demanding ever increasing I/O bandwidth to scale effectively to an extreme number of the compute processors. A typical bandwidth requirement at the level of 10 GB/s or much higher can easily outstrip the capacity of a single server on any file system.

As a result, Parallel NFS (pNFS) has been introduced as an extension of NFS to meet the bandwidth demands of these applications. It achieves this improvement by decoupling the data path of NFS from its control path, thereby enabling concurrent I/O streams from many client processes to storage servers.

Many past and existing file systems provide parallel and distributed I/O, such as GPFS [23], Panasas [10], Lustre [31], PVFS [3] to name a few. But pNFS stands out as the only open standard that is currently being developed and endorsed by many commercial and non-profit organizations.

pNFS offers a number of attractive features. First, its versatile layout drivers make it feasible to be built on any existing I/O and storage protocols as has been exemplified for PVFS [16], Panasas, and others. Second, it provides a smooth, effortless path to migrate and upgrade existing NFS deployments in order to improve scalability and performance. Third, recent efforts have shown that pNFS is a compelling technology for wide-area storage access [15]. As a result, pNFS offers a convenient approach to datacenter-wide file system, or a geographically-distributed file system.

While a number of prototypes have been demonstrated for pNFS, its portability and effectiveness are yet to be demonstrated on many other file systems, such as

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

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

Lustre. Lustre is a popular file system that has been deployed on supercomputers, such as the 1.3 Petaflop Cray XT at Oak Ridge National Laboratory, Tera-10 at CEA in Europe, and TSUBAME at Tokyo Tech in Japan. Some of these sites are actively engaged in developing a centralized, Lustre-based file system to avoid moving data across clusters for different purposes, such as simulation and visualization. Such a strategy requires that the Lustre code base be installed at all clients, and that powerful Lustre servers be available for request forwarding.

In contrast, a pNFS implementation on Lustre can quickly enable file access from any client to Lustre storage through the standard NFS protocol. Thus, it provides a competitive alternative for an institution-wide central file system.

In this paper, we present the design and implementation of lpNFS: pNFS over Lustre-based storage. Our design is motivated by several key goals. First, to maintain transparency in lpNFS, pNFS clients shall have no internal knowledge of the underlying Lustre file system. Second, pNFS must achieve an I/O performance that is close to the original Lustre performance (without pNFS). To achieve these goals, we investigate the design alternatives of control and storage protocols in the overall architecture of lpNFS, and explore what to include in a file layout for the interactions between pNFS clients and servers. We then optimize lpNFS by providing fast data transfer paths including (a) memory copying for small messages and (b) page sharing for zero-copy transfer of large messages. An initial performance evaluation of lpNFS is provided, using sequential I/O, parallel I/O, and scientific benchmarks. The performance of lpNFS is compared to both the original Lustre file system and another implementation of pNFS, simple parallel NFS (spNFS) from Network Appliance Inc. Experimental results showed that lpNFS is able to achieve the performance comparable to Lustre, and better than spNFS.

In summary, in this paper, we argue that pNFS is appealing as an open standard for parallel file systems in computer centers, and also, it provides benefits that merit its consideration for other diverse environments, such as scalable data access to clients on the wide-area network [15]. Meanwhile, Lustre has been deployed in various supercomputing sites and has proven scalability. By combining the benefits of pNFS and Lustre, we demonstrate that lpNFS is a promising strategy to bring scalable bandwidth transparently to diverse set of clients at scale.

The rest of the paper is organized as follows. An overview is provided on Lustre and pNFS software architectures in the next section. Then, the design and implementation of lpNFS is presented in Section 3. Section 4 gives the results of performance evaluation. Section 5 provides a review of related work. Finally, we conclude the paper in Section 6.

## 2. Overview of Lustre and pNFS

### 2.1. Lustre

Lustre [31] is a POSIX-compliant, stateful, object-based parallel file system. It provides fine-grained, scalable I/O accesses to many storage targets. As shown in Figure 1, Lustre separates essential file system activities into three components: clients, meta-data servers, and object storage servers. Three internal software components referred to as Object Storage Client (OSC, a part of client not shown), Meta-Data Server (MDS) and Object Storage Server (OSS) are responsible for undertaking the interactions between Lustre clients and servers. Lustre meta-data operations are decoupled from file IO operations in Lustre.

To access a file, a client first obtains from the primary MDS its meta-data, including file attributes, file permission, and the layout of file objects. Subsequent file I/O operations are performed directly from the client to the corresponding OSSes. By decoupling meta-data operations from data operations, I/O is performed concurrently, which allows greater aggregate bandwidth. Lustre also provides a fail-over MDS, which is quiescent normally but can provide complete meta-data services in case the primary MDS fails.

### 2.2. pNFS

pNFS is an extension introduced in NFS version 4.1 [24]. Its main objective is to remove the single server bottleneck of traditional NFS, and to meet the requirements of extreme I/O bandwidth in various environments, such as high-performance computing. pNFS is also positioned to be an industry standard client interface for parallel data accesses. Like other parallel file systems, such as Lustre, pNFS provides separate paths for metadata and storage protocols.

The pNFS metadata server (MDS) is a central component in pNFS. It is responsible for control and file management operations. As shown in Figure 2, a pNFS client contacts MDS through the standard protocol defined in NFSv4.1. MDS, then, performs the intended operation, such as creation, update, deletion, or query attributes, through a control protocol to the data servers.

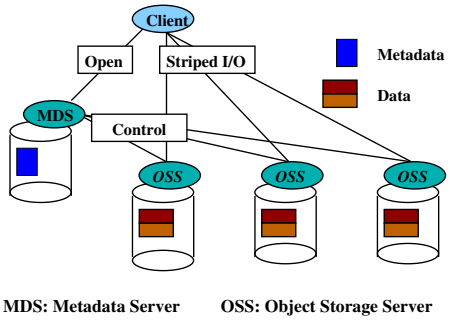



**Figure 1. Lustre Architecture**

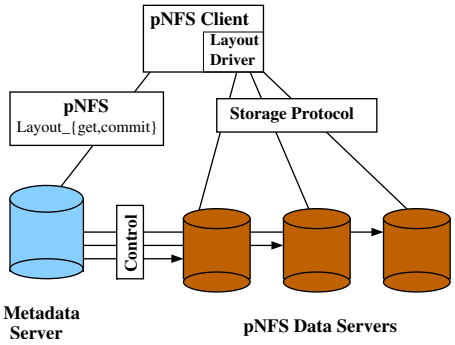



**Figure 2. pNFS Architecture**

NFSv4.1 layout operations, such as *layoutget* and *layoutcommit*, enable clients to exchange file layouts with the metadata server. For parallel data accesses, a client obtains from the MDS a layout descriptor, which describes how to access a file. Using the layout driver, the pNFS client can then translate the I/O requests, discern the layout, and obtain data from the data servers using a file-, object-, or block-based storage protocol. With the separated control and data paths, pNFS clients can obtain scalable bandwidth from a collection of data servers.

To allow a versatile implementation of pNFS, the NFSv4.1 specification does not dictate the specifics of control protocol so that an implementation of pNFS server can freely choose to communicate either directly with the data servers, or indirectly through the parallel file system. As an ongoing standardization effort, NFS version 4.1 contains only the description of an NFSv4-based file layout driver. Object- or storage-based layout drivers are left for future standardization.

## 3. Designing pNFS on Lustre

We describe the design of lpNFS in terms of control and storage protocols, the layout driver, and our optimizations of data transfer paths.

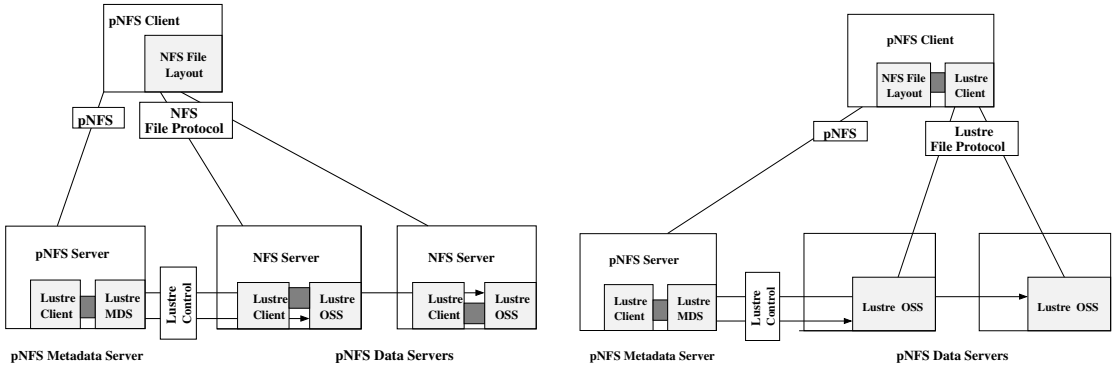
### 3.1. Control and Storage Protocols

The primary issue for lpNFS is to choose appropriate control and storage protocols. The pNFS metadata server relies on a control protocol to manage the data servers, while the pNFS clients perform I/O to data servers according to the storage protocol. Any control protocol would have to provide the functionalities of Lustre MDS for interactions with other components of Lustre, and meet the requirement of maintaining consistency with the original Lustre-MDS. Therefore, leveraging the existing Lustre control protocol for storage management is the natural choice. As shown in Figure 3, pNFS Metadata server (MDS) includes an encapsulated Lustre client and MDS. When receiving a request from pNFS client, pNFS MDS translates the intended control and/or management operation to Lustre client and MDS for execution.

However, for the storage protocol, there are several drawbacks to enable a co-located Lustre client and translate I/O requests from a pNFS client to the Lustre client. First, it requires that all the clients and servers to work within a Lustre environment, which is typically limited within a cluster or an institution. Second, the entire Lustre client code must be installed and running on all the pNFS clients, which defeats the purpose of transparent pNFS accesses, and limits the availability to only Lustre-capable clients. Third and equally important, it cannot leverage the existing NFS machinery for security and authentication management across multiple security domains. Figure 3 compares two different choices: lpNFS with NFS file protocol on the left and lpNFS with Lustre file protocol on the right. Due to the aforementioned reasons, we designed lpNFS using the ubiquitous NFS file protocol. A pNFS data server is co-located with a Lustre client and OSS, and works together with them to complete the I/O operations. It fulfills our main objective to support transparent NFS clients from any environment, within or out of a single administration domain, resident locally or geographically-distributed.

### 3.2. Layout Driver

For data accesses, pNFS clients need to obtain the information about the data servers, the file attributes, and the underlying layout of a file. These are accomplished through the pNFS operations, namely *GETDEVICEINFO*, *LAYOUTGET*, and *LAYOUTCOMMIT*. *GETDE-*




(a) lpNFS with NFS file protocol (b) lpNFS with Lustre file protocol

**Figure 3. Alternative Architectures for pNFS on Lustre**

*VICEINFO* obtains information on how to access the data servers. *LAYOUTGET* obtains information on how to access a file; *LAYOUTCOMMIT* commits back any changes in an altered file layout. To maintain transparency, our implementation does not alter the pNFS layout data structure, nor the pNFS layout driver on the clients. We support these operations by introducing additional calls on the Lustre metadata servers to do the following.

*GETDEVICEINFO*: When a pNFS client mounts lpNFS file system, a list of IP address and port number (default NFS port 2049) for all data servers is provided in the reply from the MDS to the client.

*LAYOUTGET*: The purpose of this operation is to retrieve the file attributes, striping parameters, and a file identifier (a.k.a file handle) for storage accesses on the NFS data servers. Lustre manages its storage as object units Ideally, it would be convenient to encode Lustre object IDs in the file layout. However, an NFS server does not recognize such Lustre object IDs, and is unable to locate the objects accordingly. For our initial implementation, we encode only the NFS file handles, one per NFS data server, in a file layout. When receiving a file handle, a NFS server opens a Lustre file accordingly. The co-located Lustre client and OSS locate the Lustre objects.

*LAYOUTCOMMIT*: When a pNFS client needs to commit altered metadata back to the metadata server, it uses this operation to inform the server of changes in the file layout such as an extension of the file size. Currently, in lpNFS, file accesses and updates on data servers are kept in sync with Lustre MDS through the concurrent Lustre file system, so this operation is a mere validation of file layout and attributes.

### 3.3. Optimizing Data Transfer in lpNFS

With the lpNFS architecture shown in Figure 3(a), the pNFS client does not contain any Lustre components, drivers, or code, whereas the pNFS metadata and data servers are co-located with Lustre MDS and OSSes, respectively. In this architecture, the main data paths for Lustre occur between Lustre client and the co-located Lustre MDS or OSS. Lustre is a highly componentized, layered architecture. Figure 4 shows a portion of its software stack to illustrate the data paths of request, reply, and bulk data (for large messages). All messages traverse through LOV (Lustre Object Volume), OSC, LNET (Lustre Networking Layer), the TCP/IP stack, finally to the OSS and the storage target. A small operation is executed as as a pair, consisting of the request and matching reply. For large messages with bulk data (e.g., read and write requests) , Lustre implements a separate data path. In this path, a client allocates memory pages, constructs a list of *niobuff*s (Network I/O Buffers) that describe data, and then make a request to the server. When the server detects a bulk data request, it acquires another set of memory pages, constructs a second list of niobuffs, and then invokes the bulk data transfer between the two lists of niobuffs. We introduce the following two optimizations in the data transfer paths.

**Fast memory copying for small messages** – As noted earlier, all data transfer goes through the TCP/IP stack. While the operating system implements a loopback network device that can detect and avoid going through the network, there is an additional data copy on the TCP/IP receive path using this loopback interface. We introduce an optimization that detects this co-located client and server, and, then, performs fast message transfers through memory copying. Figure 4 also shows a diagram of fast memory copying at the LNET layer. This opti-

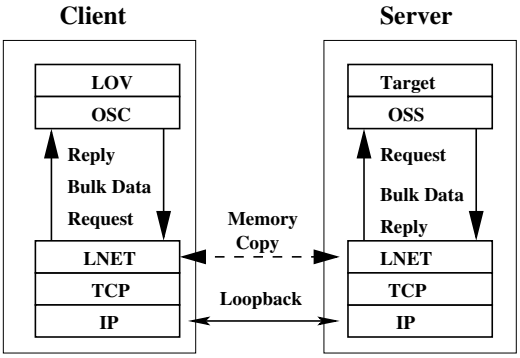



**Figure 4. Data Transfer with Single Memory Copy**

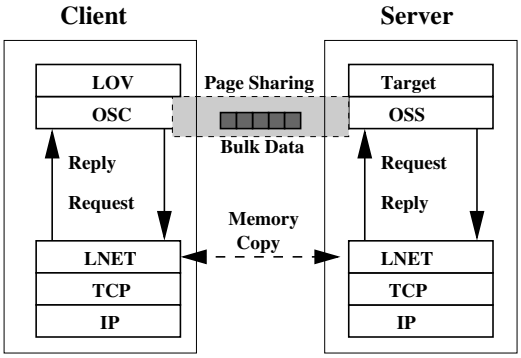



**Figure 5. Zero-copy Bulk Data Transfer**

mization is primarily targeted for small messages; large messages are further optimized as described below.

**Zero-copy transfer for large messages** – A single copy between two lists of niobuffs still incurs unnecessary overhead by increasing memory footprint and consuming more memory bandwidth. To address this issue, we introduced an optimization to enable page sharing between the client and the server. For large messages with bulk data, a client encodes a descriptor about the memory pages holding the data. When the server receives this request, it directly retrieves the list of memory pages as buffers for storage accesses. When the data is ready, the memory pages are returned to, and then freed by the client. Figure 5 shows the diagram of zero-copy bulk data transfer between the OSS and the OSC.

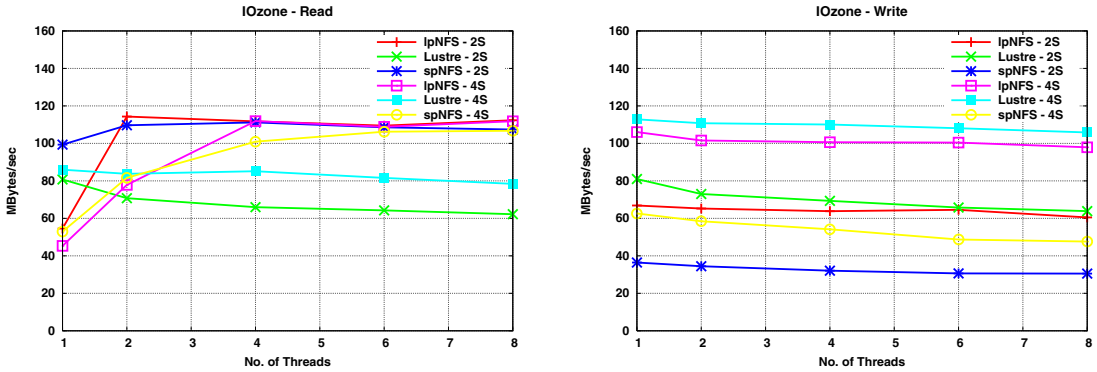
# 4. Performance Evaluation

We have designed and implemented a prototype of lpNFS on top of a developmental pNFS tree, Linux-2.6.25-pNFS from linux-nfs.org, and a base Lustre release, version 1.6.5. Our experiments were conducted on a cluster of Clara VXB-7520J blades: each with dual Intel Xeon 3.4 GHz processors, 2MB cache, 800MHz Front Side Bus (FSB), 4GB physical memory, and a Gigabit Ethernet card connected to PCI-X 100 Mhz bus. On each Lustre OST, a 80GB, 7200 RPM, ATA/100 Western Digital hard disk WD800JB is used for disk storage. These nodes are running CentOS 5.1 Linux operating system. We compared lpNFS to the original Lustre, and spNFS, a simple parallel NFS implementation from Network Appliance Inc. All tests were conducted with one MDS, up to 16 clients, and a varying number of storage servers (pNFS Data Servers or Lustre Object Storage Servers). All data were collected through 10 runs and the average of the best five was taken as the final number. Very small deviations were observed so we omitted the error bars in these figures.

## 4.1. Sequential I/O

IOzone [2] is a popular sequential I/O benchmark that measures a broad range of performance characteristics of a file system. We used IOzone to measure the performance of lpNFS. A varying number of threads were used for the sequential I/O tests. IOzone record size was 16 MB. Currently, the pNFS client implementation does not support client-side caching. The total size of data from all threads was chosen to be 5120 MB, bigger than the memory size to eliminate Lustre client-side caching effects.

Figure 6 shows the IOzone performance comparisons of lpNFS, spNFS and Lustre. With an increasing number of threads, lpNFS has the best read performance among the three. Both lpNFS and spNFS can reach near the maximum network performance of Gigabit Ethernet, with 2 storage servers (2S) or 4 storage servers. This is because the spNFS and lpNFS servers can keep all the data in their buffer caches. In contrast, the read performance of Lustre stays low, because of the elimination of Lustre client-side caching effects using the large data size, and also because Lustre does not have server-side file caching. Thus, Lustre read performance for IOzone is determined by the aggregate performance of server-side disks. Compared to 2 storage servers, four servers provide better performance for reads. On the other hand, Lustre achieved the best performance for IOzone writes. lpNFS achieves 88%-98% of Lustre write performance, indicating a low-overhead pNFS implementation on top




(a) IOzone Read (b) IOzone Write

**Figure 6. lpNFS Sequential I/O**

of Lustre. spNFS has the lowest IOzone performance among the three.

### 4.2. Parallel I/O

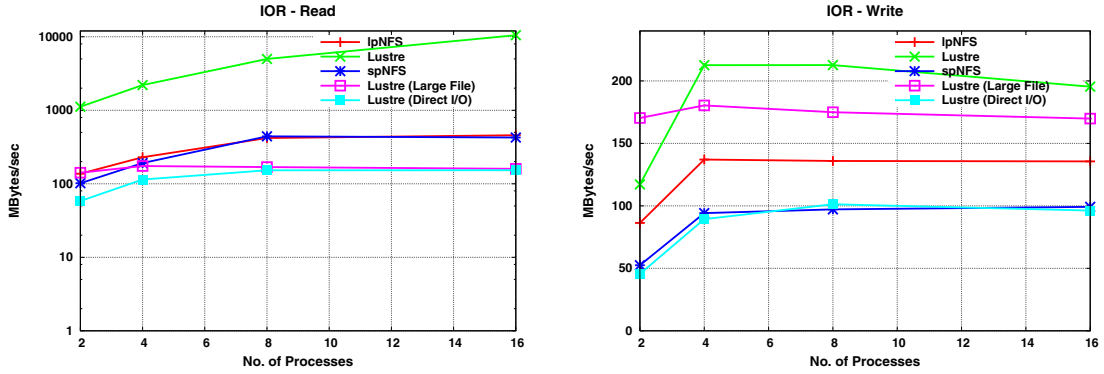
IOR [18] is a parallel I/O benchmark that measures a broad range of access patterns, including concurrent I/O to separate files or to a shared file. It supports a number of different I/O interfaces, including POSIX, MPI-IO, and HDF5 [12]. We used the POSIX interface of IOR to measure the aggregate bandwidth from concurrent reads or writes, using separated files. We measured the parallel I/O performance of lpNFS using four storage servers. The data size was chosen as 512 MB per process, and the transfer size as 16 MB. To show the performance of Lustre without client-side caching effects, we also run Lustre using direct I/O, and a large file size of 5120 MB.

Figure 7 shows the parallel I/O performance of lpNFS, compared to spNFS and Lustre. For parallel reads, the client-side caching effects is apparent for Lustre. When using a large file size or direct I/O, the performance of Lustre parallel reads stayed about the same, suggesting the performance is determined by the number of server-side disks. In contrast, the aggregate read bandwidth of lpNFS and spNFS reached the level of 450 MB/sec, close to the aggregate link bandwidth of four storage servers. Lustre achieves the best aggregate write bandwidth, except in the case of direct I/O. No dramatic client-side caching effects were observed for writes because Lustre client periodically flushes dirty data to the storage servers, with the maximum size for cached dirty data as a configurable parameter (40 MB per storage server in our experiments). As mentioned earlier, lpNFS does not have the support of client-side caching. But it still achieves 79% of the Lustre performance when the Lustre file is 5120 MB, and 70% when the file size is 512 MB.

### 4.3. Scientific Application Benchmark

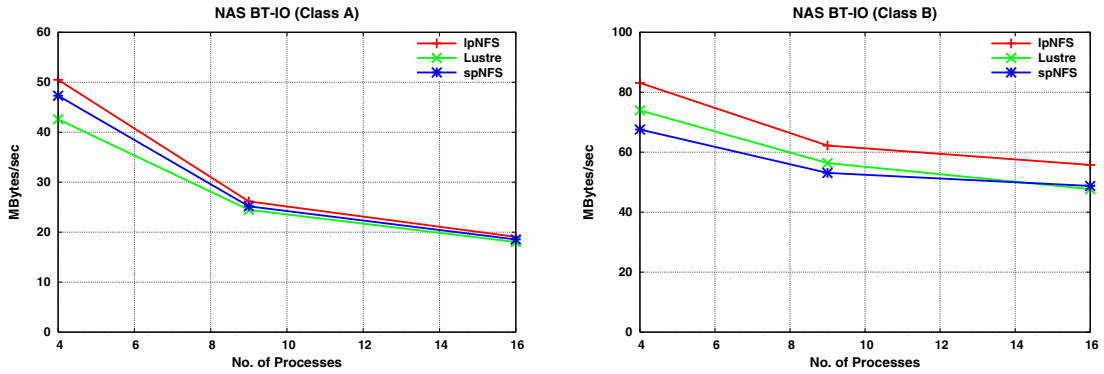
Scientific applications can have very diverse I/O access patterns. We used the NAS BT-IO benchmark [28] as a representative to evaluate the performance of lpNFS. NAS BT-IO was developed at NASA Ames Research Center. It tests the output capability of NAS BT (Block-Tridiagonal) parallel benchmark. Data sets in BT-IO undergo diagonal multi-partitioning and then distributed among MPI-processes. The data structures are represented as structured MPI datatypes and written to a file periodically. There are several different BT-IO implementations, which vary on the methods used to perform the file I/O. In our experiments, we used an implementation that performs I/O using MPI-IO collective I/O routines – the so called *full mode* BT-IO.

We measured the performance of NAS BT-IO on lpNFS, spNFS, and Lustre, all with four storage servers. Figure 8 shows the performance of BT-IO, classes A and B, using a varying number of processes. lpNFS delivered the best collective I/O performance for NAS BT-IO, while spNFS performed comparably to Lustre for Class A, but the lowest for Class B. With an increasing number of processes, the aggregate BT-IO bandwidth decreases. This is because the file size for the same class of NAS BT-IO program is the same, and more processes lead to finer-grained data patterns. Therefore, more time is spent at the MPI-IO layer to perform data aggregation in collective I/O. It is also interesting to note that lpNFS achieves better performance than the original Lustre. This is because lpNFS clients does not have to invoke the costly Lustre locking operations like the Lustre clients do. Instead, the servers in lpNFS request Lustre locks. They can also amortize the cost of locks on its cached data.




(a) Read (b) Write

**Figure 7. lpNFS Parallel I/O**




(a) Class A (b) Class B

**Figure 8. Scientific Benchmark using lpNFS**

## 5. Related Work

There is a long history and a large body of research on NFS. The latest version NFS version 4 [1] specification [25] proposes a number of salient features, such as client delegation, compound operations, and file replication and migration, among many others. A number of studies have shown the performance benefits of different caching strategies to NFS. Peng et al. [21] showed that a network-centric reorganization of the buffer cache can improve the NFS performance. Xu et al. [29] investigated the performance benefits of client caching to concurrent read sharing over NFS. Batsakis et al. [5] extends NFSv4 delegation model to enable clustered delegation among multiple clients, thereby improving the performance and recoverability for a cluster of computing clients. Gulati et al. [13] proposed a proxy NFSv4 caching strategy, Ncache, to enable a consistent cache of a remote NFS server that can be maintained and shared across multiple local NFS clients. It leveraged the features of NFSv4 to improve the performance of file accesses in a wide-area distributed setting by bringing the data closer to the client. Recently, the emergence of high speed networks with direct access protocols such as RDMA (Remote Direct Memory Access) led to a new version of RPC [26] that can enable fast data transfer over RDMA-capable networks. Callaghan et al. [7] provided an initial implementation NFS over RDMA (NFSoRDMA) on Solaris. A team at the Ohio State University further completed the design and development of NFSoRDMA on Open Solaris for performance enhancement, compliant to IETF specification [8]. Talpey et al. [27] recently announced the availability of initial Linux NFSoRDMA implementations.

Multiple research efforts have investigated pNFS over the recent years. Hildebrand et al. [6] first demonstrated a prototype of pNFS over PVFS2 in 2005. The original prototype did not have pNFS data servers; instead, the I/O requests in pNFS clients were redirected to a co-located PVFS client daemon, and then, con-

verted to PVFS client requests for I/O accesses. Further efforts from the same team demonstrated the direct-pNFS architecture using PVFS [16]. In direct-pNFS, pNFS clients talk to pNFS metadata and data servers, which export the PVFS storage to pNFS clients. Direct-pNFS showed the benefits of direct I/O accesses from the pNFS data server to the attached storage, without going through a PVFS client daemon. Using this prototype, Hildebrand et al. [17] also investigated how to improve the overall performance of pNFS by enabling different I/O paths for small and large write requests.

There has been continuous interest in pNFS from a number of storage vendors: Panasas has undertaken the task of developing an object-based layout driver; EMC, the task of block-based layout driver; and, Network Appliance, the task of file-based layout driver. Some early prototypes are already available for adventurous users. In particular, the simple parallel NFS (spNFS) used in this paper is developed by Network Appliance, Inc. It provides a simple NFS-based file layout driver that can integrate multiple NFS data servers into a single file system as pNFS. Using spNFS, the Ohio State University team demonstrated NFSoRDMA as a fast RPC transport protocol for pNFS [19]. Our work adds to these file-based pNFS layout drivers by supporting transparent pNFS on top of Lustre. We examined the internal Lustre software architecture and provided an efficient lpNFS implementation while maintaining the file consistency of Lustre.

Lustre has also garnered significant research activity. He et al. [14] studied the benefits of a parallel data moving architecture to scalable data archiving and backup from Lustre file system to hierarchical storage devices. In order to provide parallel archiving paths, they have introduced a new Lustre module to move Lustre objects directly between individual Lustre object targets and hierarchical storage devices. Zhang et al. [32] proposed a coordinated scheme to address the availability of job input data by staging in and reconstructing on-demand transient data as jobs are being scheduled to the execution queue in a supercomputing environment. In order to reconstruct partial file segments, they have instrumented the striping attributes of Lustre files to identify Lustre targets from which data is staged or offloaded. Yu et al. [30] proposed a hierarchical striping mechanism to mitigate the overhead of wide striping in Lustre for better collective I/O. The hierarchical striping prototype is implemented inside MPI-IO by creating many files with small, balanced stripe widths for parallel I/O, and then joining them in place using the Lustre file joining feature. Our work differs from all these work to enable the versatile pNFS protocol for transparent client accesses to Lustre. [14] and [32] are close to our work in terms of enabling I/O access to individual Lustre object targets. However, [14] did not address the concern of maintaining the name space of Lustre objects after they were moved into hierarchical storage devices. Due to the different environments being targeted, neither of them provided optimizations between co-located Lustre client and servers. There are many other techniques in the literature to provide zero-copy data movement, such as Fbuf[11], IO-Lite [20], zero-copy TCP [9], and page remapping [4]. All these techniques were targeted to be generically applicable to other system components, requiring intrusive customization to the operating system. In contrast, our techniques are Lustre-specific optimizations. They enable fast data paths for lpNFS through optimizations in the Lustre code base, without any change to the operating system.

## 6. Conclusions and Future Work

We have presented the design and implementation of lpNFS: pNFS on the Lustre File System. We designed lpNFS using a NFS-based file protocol for storage access in order to allow transparent client accesses. In view of the co-located Lustre clients and servers in lpNFS, we provided fast data transfer paths through fast memory copying of small messages and zero-copy page sharing of large messages. We also conducted a performance evaluation of lpNFS using sequential I/O, parallel I/O, and scientific benchmarks, comparing it to Lustre and spNFS. Our results indicated that the performance of lpNFS is comparable to Lustre, and better than spNFS. With the emergence of pNFS as an open standard for parallel file systems and Lustre's proven scalability in many supercomputing sites, we believe that lpNFS is well positioned to combine the strengths of the two and bring the scalable bandwidth of Lustre to clients across many diverse environments.

In the future, we plan to study the metadata and bandwidth scalability of pNFS using lpNFS in a larger cluster. We also intend to study the performance of lpNFS for large-scale scientific applications within a computer center or across wide-area network. Furthermore, we intend to extend lpNFS further with more features that are defined for pNFS, such as client-side caching, session management, and file delegation.

## References

[1] General Information and References for the NFSv4 pro-

tocol. http://www.nfsv4.org/.

[2] IOzone Filesystem Benchmark. In *http://www.iozone.org*.

[3] The Parallel Virtual File System, version 2. http://www.pvfs.org/pvfs2.

[4] D. C. Anderson, J. S. Chase, S. Gadde, A. J. Gallatin, K. G. Yocum, and M. J. Feeley. Cheating the i/o bottleneck: Network storage with trapeze/myrinet. In *Proceedings of the 1998 USENIX Technical Conference*, pages 143–154, 1998.

[5] A. Batsakis and R. Burns. Cluster delegation: high-performance, fault-tolerant data sharing in nfs. In *Proceedings of the High Performance Distributed Computing (HPDC)*, pages 100–109, Washington, DC, USA, 2005.

[6] D. L. Black and S. Fridella. pNFS Block/Volume Layout. http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-pnfs-block-00.txt.

[7] B. Callaghan, T. Lingutla-Raj, A. Chiu, P. Staubach, and O. Asad. NFS over RDMA. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence*, pages 196–208. ACM Press, 2003.

[8] B. Callaghan and T. Talpey. RDMA Transport for ONC RPC. http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-rpcrdma-02.txt.

[9] H. K. J. Chu and S. Inc. Zero-copy tcp in solaris. In *In Proceedings of the USENIX 1996 Annual Technical Conference*, pages 253–264, 1996.

[10] A. M. David Nagle, Denis Serenyi. The Panasas ActiveScale Storage Cluster – Delivering Scalable High Bandwidth Storage. In *Proceedings of Supercomputing '04*, November 2004.

[11] P. Druschel and L. L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *Proceedings of the Fourteenth ACM symposium on Operating Systems Principles*, pages 189–202, 1993.

[12] T. N. C. for Supercomputing Applications. HDF Group - HDF5. http://www.hdfgroup.org/HDF5/.

[13] A. Gulati, M. Naik, and R. Tewari. Nache: Design and implementation of a caching proxy for nfsv4. In *Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST)*, San Francisco, CA, February 2007.

[14] D. He, X. Zhang, D. H. Du, and G. Grider. Coordinating Parallel Hierarchical Storage Management in Object-base Cluster File Systems. In *Proceedings of the 23rd IEEE Conference on Mass Storage Systems and Technologies (MSST)*, May 2006.

[15] D. Hildebrand, M. Eshel, R. Haskin, P. Andrews, P. Kovatch, and J. White. Deploying pnfs across the wan: First steps in hpc grid computing. In *Proceedings the 9th LCI International Conference on High-Performance Clustered Computing*, Urbana, IL, April 2008.

[16] D. Hildebrand and P. Honeyman. Direct-pNFS: Scalable, Transparent, and Versatile Access to Parallel File Systems. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing*, Monterey, CA, June 2007.

[17] D. Hildebrand, L. Ward, and P. Honeyman. Large files, small writes, and pnfs. In *Proceedings of the 20th ACM International Conference on Supercomputing (ICS06)*, Cairns, Australia, June 2006.

[18] LLNL. IOR Benchmark. http://www.llnl.gov/asci/purple/benchmarks/limited/ior.

[19] R. Noronha, X. Ouyang, and D. K. Panda. Designing a high-performance clustered NAS: A case study with pnfs over rdma on infiniband. In *Proceedings of International Conference on High Performance Computing (HiPC)*, 2008.

[20] V. S. Pai, P. Druschel, and W. Zwaenepoel. Io-lite: A unified i/o buffering and caching system. In *ACM Transactions on Computer Systems*, pages 15–28, 1999.

[21] G. Peng, S. Sharma, and T. Chiueh. A Case for Network-Centric Buffer Cache Organization. In *Hot Interconnect 11*, August 2003.

[22] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network Filesystem. In *Proc. Summer 1985 USENIX Conf.*, pages 119–130, Portland OR (USA), 1985.

[23] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *FAST '02*, pages 231–244. USENIX, Jan. 2002.

[24] M. E. S. Shepler and D. Noveck. NFS Version 4 Minor Version 1. http://tools.ietf.org/html/draft-ietfnfsv4-minorversion1-19.

[25] S. Shepler, B. Callaghan, D. Robinson, R. thurlow, C. Beame, M. Eisler, and D. Noveck. NFS version 4 Protocol. http://www.ietf.org/rfc/rfc3530.txt.

[26] R. Srinivasan. RPC: Remote Procedure Call Protocol Specification Version 2. http://www.ietf.org/rfc/rfc1831.txt.

[27] T. Talpey et. al. NFS/RDMA ONC Transport. http://sourceforge.net/projects/nfs-rdma.

[28] P. Wong and R. F. Van der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. Technical Report NAS-03-002, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division.

[29] Y. Xu and B. D. Fleisch. NFS-cc: Tuning NFS for Concurrent Read Sharing. *The International Journal of High Performance Computing and Networking (IJHPCN)*, 3, 2004.

[30] W. Yu, J. Vetter, R. S. Canon, and S. Jiang. Exploiting lustre file joining for effective collective io. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, May 2007.

[31] R. Zahir. Lustre Storage Networking Transport Layer. http://www.lustre.org/docs.html.

[32] Z. Zhang, C. Wang, S. S. Vazhkudai, X. Ma, G. G. Pike, J. W. Cobb, and F. Mueller. Optimizing center performance through coordinated data staging, scheduling and recovery. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC)*, 2007.