NFS/RDMA over InfiniBand: Is It Beneficial? *

Ranjit Noronha, Weikuan Yu and D.K. Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{*noronha,yuw,panda*}*@cse.ohio-state.edu*

Abstract

Network File System (NFS) is a ubiquitous component of most modern clusters. It allows users to transparently share file and IO services on a variety of different platforms. Traditionally NFS has used TCP or UDP as the underlying transport for communication. However, these protocols add considerable overhead to communication and may not allow applications to extract performance from high-speed networks. The emergence of modern high-performance networking technologies like InfiniBand in conjunction with technologies like PCI-Express have dramatically enhanced the achievable network bandwidth and reduced message latency. These networks also allow the user to use operations like Remote Direct Memory Access (RDMA) to achieve zero-copy communication. Recent NFS protocol extensions can communicate directly over RDMA-capable networks. This may potentially help improve performance of these protocols. However, given the dependency of NFS performance on the application and underlying system characteristics, it is not clear whether there can be an improvement in performance at the application level. In this paper, we evaluate the performance of NFS over RDMA in OpenSolaris on InfiniBand using a variety of benchmarks, applications and system utilities. These experiments show that NFS/RDMA can increase MPI Tile I/O write bandwidth by 126% compared to NFS/IPoIB, as well as reduce kernel compile time by 73%. In addition, PCI-Express can help reduce the bottlenecks imposed by PCI-X and help improve IOzone aggregate Write bandwidth in NFS/RDMA up to 24%. Finally, we show that NFS critically depends on the back end filesystem being used. With design changes to the back end memory based filesystem, the performance of NFS over the same transport show an order of magnitude improvement of up to 11 times.

Keywords: *InfiniBand, NFS, Distributed File Systems, System Area Networks, Clusters*

1 Introduction

Modern computer systems have evolved substantially over the past decade with impressive improvements in computer and memory speeds, with decreasing costs. This has led to the evolution of clusters of servers to allow users to cheaply expand the computing capacity available to meet additional demand. While modern clusters allow users to expand as demand increases, there is increasing complexity associated with the management of resources across the cluster. To ease this complexity, many modern cluster systems, use NFS [1] mounted directories across a large set of machines. While NFS makes file-sharing easy, it typically uses networks like Gigabit Ethernet. Most Gigabit Ethernet networks only allow TCP or UDP for network communication. With most NFS installations being a single server, multiple client entity, the scalability of the NFS server may be constrained by the protocol overhead and copying cost of TCP or UDP, as well as by the network bandwidth of these networks.

Modern high performance networks like InfiniBand 4X allow applications to exploit low latency of a few microseconds and high bandwidth communication up to 10 Gbps. They also allow Remote Direct Memory Access (RDMA) based communication. RDMA allows the application to achieve zero-copy communication. This could potentially reduce communication overhead. Technologies like PCI-Express allow InfiniBand 4X to achieve lower latency as well as full bidirectional bandwidth [14] eliminating the constraints of legacy technologies like PCI-X.

With the emergence of modern interconnects like InfiniBand [13], protocols such as NFS/RDMA [10] were designed to exploit the low latency, high bandwidth, NIC offload capabilities provided by these interconnects. It is natural to ask whether NFS server storage exported across RDMA capable networks can really be deployed with acceptable performance, so that applications can benefit with enhanced performance on IO accesses. In addition, can the RDMA offload capabilities of InfiniBand reduce overhead at the server and enhance scalability. Or since NFS performance is largely dependent on many factors such as system-level scheduling and response time, availability of limited resources in the kernel, disk bandwidth and design of the back-end file system, will the improvements afforded by better communication primitives not offer potential benefit?

In this paper, we attempt to study the performance of the first implementation of NFS/RDMA on the OpenSolaris environment [10]. We do this using a variety of widely used benchmarks. The benchmarks characterize the performance impact of NFS/RDMA compared to that of NFS/IPoIB. The impact of NFS/RDMA on CPU utilization is also measured. In addition, the impact of PCI-Express on NFS/RDMA performance is also measured. These experiments show an im-

*This research is supported in part by Department Energy's grant #DE-FC02-01ER25506, National Science Foundation grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, and Sun Microsystems; and equipment donations from Intel, Mellanox, and SUN Microsystems.

provement of 126% in MPI Tile I/O write bandwidth for NFS/RDMA compared to NFS/IPoIB as well as a 73% reduction in kernel compile time. Additionally, we show that NFS critically depends on the back end filesystem being used. With design changes to the back end memory based filesystem, the performance of NFS irrespective of the underlying network protocol transport, show an order of magnitude improvement of 11 times.

Section 2 provides an introduction to NFS and the networking technology InfiniBand. Following that in Section 3, the evaluation in terms of micro-benchmarks is performed. In section 4, the impact of NFS/RDMA on CPU utilization is evaluated. Section 5 explores the impact of PCI-Express. Section 6 presents related work. Finally in Section 7, we present conclusions and future work.

2 Background

In this section, we provide brief overviews about InfiniBand, NFS and Sun MicroSystem's implementations of InfiniBand and NFS over RDMA for the OpenSolaris Operating System.

2.1 InfiniBand Overview

The InfiniBand Architecture (IBA) [13] is an open specification designed for interconnecting compute nodes, IO nodes and devices in a system area network. InfiniBand supports channel based semantics for reliable communication. Operations include traditional send/receive semantics as well as Remote Direct Memory Access (RDMA) primitives. Send/Receive (*Send*) primitives require the prior posting of a descriptor on the receiver side. RDMA operations allow secure access through protection domains to the memory of a remote node without involvement of that node. RDMA operations are mainly of two types Read and Write. *RDMA Read* allows a given node to directly read the contents of a remote node. Similarly, *RDMA Write* allows one to deposit data directly into the memory of the remote node. Implementations of an InfiniBand stack include OpenIB Gen2 [4] and OpenSolaris IBTL [5]. PCI-Express the next generation successor to the PCI-X architecture allows InfiniBand 4X to achieve a bandwidth of 10 Gbps in both directions [14]. It also offers improved small message latency. A comparison in terms of basic micro-benchmarks is shown in Figure 1.

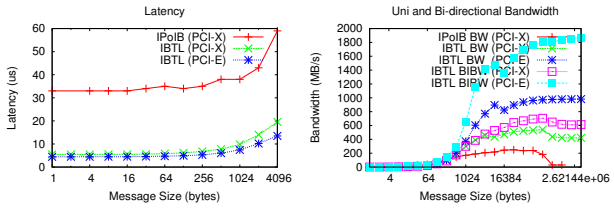

Fig. 1. Comparison between IPoIB (PCI-X), IBTL (PCI-X) and IBTL (PCI-Express)

2.2 OpenSolaris Implementations of InfiniBand

Figure 2 shows the InfiniBand software stack in OpenSolaris. At the bottom most layer the Tavor HCA driver provides access to the InfiniBand hardware. On top of the Tavor driver, the OpenSolaris InfiniBand Transport Framework (IBTF) implements the InfiniBand Transport Layer (IBTL). In addition to basic communication primitives, IBTF also includes device management modules and a communications manager. IBMF provides management access features to clients. Other access protocols such as IETF IP over InfiniBand (IPoIB) and user defined application programming layer (uDAPL) are implemented using IBTF. IPoIB implements the basic IP protocol over InfiniBand. This enables existing TCP stacks to work over InfiniBand. Traditional applications using sockets may run over InfiniBand using IPoIB. At the top most level, user-level software such as MVAPICH [3] (an implementation of MPI) and NFS/RDMA may access InfiniBand through uDAPL and IBTF respectively.

2.3 NFS Overview

Network File System (NFS) was originally developed at Sun MicroSystems in the 1980's. It has since been deployed ubiquitously in most modern clusters. NFS allows users to transparently share file and IO services on various different platforms. NFS has seen three major generations of development. The first generation, NFS version 2, provided a stateless file access protocol between the client and the server via remote procedure calls (RPCs) over UDP. On top of NFSv2, NFS version 3 provided several performance enhancements, including larger block data transfer, TCP-based transport and asynchronous write, among many others. The latest NFS protocol, NFS version 4 [1] specification was developed by IETF based on an initial draft from Sun MicroSystems. Its requirements include: improved access and performance; strong, built-in security; enhanced interoperability; extensibility; as well as improvement on locking and data sharing. The key features of NFSv4 includes compound operations, stateful file access and file delegation for aggressive client cache. Initial NFSv4 implementations have become available over several common Operating Systems, such as Linux and OpenSolaris.

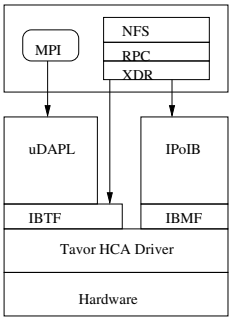

Fig. 2. OpenSolaris InfiniBand Software Stack

2.4 OpenSolaris Implementations of NFS over RDMA

As shown in Figure 2, the NFS protocol communication between the client and server is through the Remote Procedure Call (RPC) protocol. The RPC protocol can implement different network protocols such as TCP, UDP and RDMA. This allows the NFS layer to stay network independent. The OpenSolaris implementation of the RPC over RDMA protocol for IBTL is shown in Figure 3. The client

communicates with the server through the RPC Call, Reply semantics. The RPC call or request propagates from the client to the server through an RDMA Send operation. If the client has additional data which cannot be sent in the request (such as for file WRITE requests with sizes larger than 1K), the server RDMA Reads it from the client. The server processes the Request and sends a RPC Reply using RDMA Send. The client may need to RDMA Read data from the server (such as for file READ and directory list requests, with sizes larger than 1K). Finally, the client can send an RDMA Done message to the server (using RDMA Send). This allows the server to deregister buffers if needed.

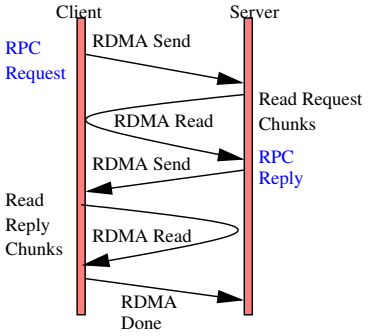

Fig. 3. OpenSolaris RPC over RDMA implementation

2.5 Pageable Memory Based File System (tmpfs)

The memory based filesystem tmpfs is manipulated by other sub-systems such as NFS through the virtual filesystem layer (VFS) [6]. The VFS layer provides functionality for mounting, unmounting, accessing the root node of the file-system, etc. Access to a particular vnode (which represents a file) is through the *vfs_vget* function. The design of the vfs_vget currently does a linear search through all the vnodes. This might cause a degradation in performance particularly for applications which frequently access the same vnode among a list of many vnodes. This might also impact relative network comparisons between different NFS implementations. We have designed a *VGET cache* to reduce the impact of the linear lookup. This cache uses an LRU eviction policy to store vnodes and provides on-demand frequently used vnodes. This *enhanced tmpfs* shown in Figure 4 is used for several experiments in the paper.

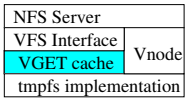

Fig. 4. Enhanced tmpfs with a vget cache

3 Micro-benchmark Evaluation

In this section, we measure the impact of various benchmarks such as IOzone, Fileop, PostMark, kernel untar and kernel compile. First the experimental setup used is described.

3.1 Experimental Setup

The hardware setup consisted of four Sun Fire V20z's and one Sun Fire V40z. All five servers have PCI-X InfiniBand MT23108 HCA's. They are connected to a SilverStorm 5000 switch. In addition two x2100 servers with MT25208 PCI-Express HCA's are used. They are also connected to the SilverStorm 5000 switch. All servers are connected with Gigabit-ethernet in addition to InfiniBand. OpenSolaris build 33 was used on all the systems. For the evaluation, the non debug version of the Open Solaris kernel source release date 02/22/06 was used. Since NFSv3 is the most commonly deployed NFS version, we use it for all our experiments.

3.2 IOzone

IOzone [2] is a sequential IO benchmark for measuring various performance characteristics of a file system. In the *read* portion of the benchmark, IOzone creates a file of size *s* in the current directory and then reads from the file in different number of chunks, each chunk being a record of size *r*. In our tests, we have used a record size of 64KB and run the benchmark for file sizes ranging from 256KB up to 1GB. We have used IOzone to mainly measure the read and write bandwidth of NFS. In our setup, the NFS server exports a tmpfs directory, which clients will mount as a NFS file system over either IPoIB or RDMA. To actually stress the network, the RAM based file system was used as the underlying file system, as discussed in Section 2.5. The resulting read and write bandwidth are shown in Fig. 5. Read bandwidth is comparable because of client side caching. There is a larger difference in the write bandwidth of approximately 15%. This is because the client must always contact the server, and as a result the network is exercised much more in this benchmark. Additionally, the write bandwidth is lower than the read bandwidth for the range of messages when the messages are fetched from the server-side cache. This is due to the fact that the server must commit the write data before sending an acknowledgment back to the receiver. This adds additional overhead at the server as compared to the read path.

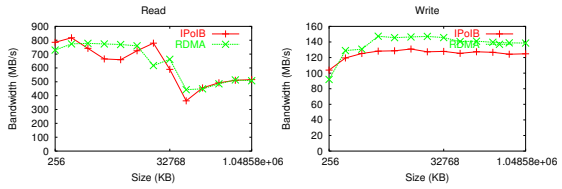

Fig. 5. Bandwidth achieved with IOzone using tmpfs

3.3 Fileop

The file operations (fileop) benchmark is included with IOzone. It measures various commonly used file system operations. The operations are file creation (create), file stat (stat), file permission access (access), directory read (readdir), file link creation (link), file link deletion (unlink) and file deletion (delete). The results are reported in the unit of operations/sec. Our setup consists of a single client NFS mounting a single directory from the NFS server over the different underlying transports RDMA and IPoIB. Results from these experiments when the server mount is on tmpfs is shown in Fig. 6. Out of these operations, create, link, unlink, readdir and delete will always generate a request to the NFS server. Therefore they benefit more from the better

performance of RDMA compared with IPoIB. On the other hand, many of the stat and access operations may be satisfied locally and therefore don't show much difference in performance.

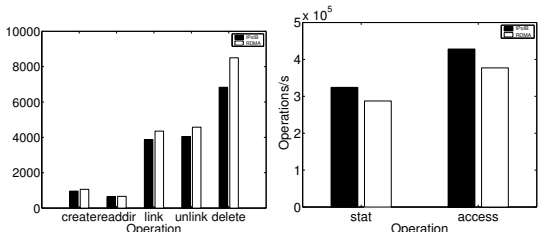

Fig. 6. Performance of different file operations with increasing system size (tmpfs)

3.3.1 PostMark

PostMark [7] from Network Appliances is a well known benchmark for measuring the performance of file systems for small files and metadata intensive applications. These types of workloads are typically seen in computing systems which process e-mail and network groups and other communication intensive environments. Postmark works by creating a set of random texts file whose sizes varies between configurable bounds. These files are continuously being modified with operations pairs consisting of create/delete or append/delete. While this is being done, file statistics are being generated. The different parameters used are shown in Table 1. We have compared the performance of RDMA to IPoIB using a single client and server with the server using tmpfs.

Clearly with the enhanced tmpfs design, the running time of PostMark over the same underlying transport is reduced by 2-3 times. This helps to more clearly show the difference between RDMA and IPoIB. We see that RDMA performs better than IPoIB by 10%-19%. PostMark is metadata intensive. Metadata operations are small and thus sensitive to the latency of the underlying transport. This is responsible for improved performance with RDMA.

Table 1. Postmark parameters

Notation	Number of files	Number of transactions
PM-1	8192	20000
PM-2	16384	40000
PM-3	32676	60,000

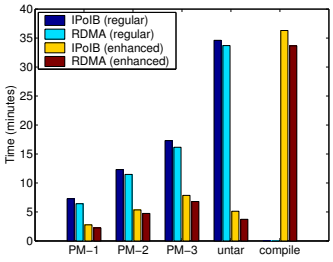

Fig. 7. Benchmarks results using the enhanced version of tmpfs (since kernel compile time with regular tmpfs is very large, these numbers are not included)

3.3.2 Kernel untar

We also measured the time required to untar a kernel tar ball source. Since OpenSolaris tar does not implement the z option (uncompress), we only measured the time required to do tar xvf opensolaris-20060222.tar with the output redirected to /dev/null. This was measured over both the regular tmpfs as well as the enhanced tmpfs described in Section 2.5. These results are shown in Figure 7. The enhanced tmpfs can dramatically reduce the time needed to untar a kernel source much as 11 times (keeping the network component the same). In addition, with the overhead of the filesystem reduced, RDMA shows a greater improvement over IPoIB by 25%. The untar process consists mainly of the following operations a. create a directory b. fetch a directory handle c. create a file (usually a C source code file). There is also a single file READ operation at startup. The file READ operation at startup, as well as many of the file WRITE's are large. The other operations are small. Small operations go inline in the RPC Call (Figure 3). The improvement in small message latency and bandwidth for RDMA over IPoIB (Figure 1) helps improve performance.

3.3.3 Kernel Compile

The time required to compile an OpenSolaris kernel (release date 02/02/2006) was also measured. These results are shown in Figure 7. Since the time with the regular tmpfs was very large (several hours), these numbers are not included. With the enhanced tmpfs design, the kernel compile time with RDMA is reduced up to 73%. This is not suprising, since kernel compile involves many file READ operations in addition to those in the untar operation explained in Section 3.3.2. In addition, there are several file READ/WRITE operations for object files created during the kernel compile. These file WRITEs vary in size from small to fairly large. The improvement in latency and bandwidth of RDMA over IPoIB (Figure 1) helps improve performance.

4 CPU Utilization

To measure the impact of CPU utilization, two parallel IO benchmarks BTIO and MPI Tile I/O were used.

4.1 BTIO

The BTIO benchmark is part of the NASA Parallel Benchmark suite of programs [17]. We use the full mode in BTIO which uses collective MPI-IO [20] routines to perform IO accesses. The default behavior of BTIO is to write to the file on every five timesteps. To stress the IO component of the system, we have modified the benchmark to write on every time step and on every two timestamps. We have used class A and B and run the applications at four processes on two nodes. To stress the network, we have used tmpfs. These results are shown in Figure 8. The outer letter corresponds to the NAS benchmark class size, while the number in brackets corresponds to the number of writes per interval. For example, A(1) corresponds to a class size A which writes to the file at every timestamp. There is a benefit for both sizes and write intervals of up to 40%. Using the OpenSolaris system utility dtrace, the time spent by

the CPU in processing NFS (IO) and network related activities is measured (time spent blocking is excluded). Figure 8 shows the breakdown of timing from these runs averaged over all clients. From these timings it is clear that the benefit comes from the reduction of processing time spent in the socket layer in the kernel.

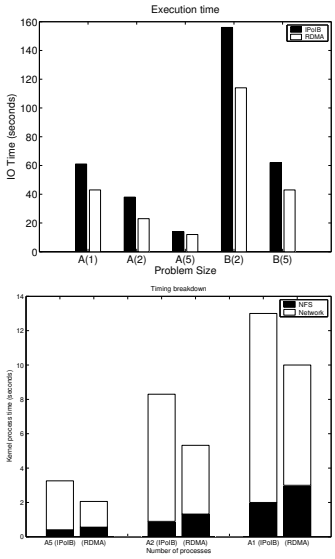

Fig. 8. IO time for different BTIO configurations (tmpfs)

4.2 MPI Tile I/O

MPI Tile I/O [8, 19] attempts to measure the impact of non-contiguous file accesses. These access patterns are commonly found in visualization and other scientific applications. We have evaluated the performance of MPI Tile I/O at two, four and six processes on one, two and three nodes, respectively. Each tile is 1024x768 pixels and each pixel is 32 bytes. When IPoIB is replaced with RDMA as the underlying transport, MPI Tile I/O read bandwidth is improved by 75% at six nodes while write bandwidth is improved by 126%. Using the OpenSolaris system utility dtrace, the time spent by the CPU in actively performing NFS (IO) and network related activities is measured (time spent blocking is excluded). From figure 10, it is clear that the reduction in execution time when the socket layer is replaced with the RDMA layer comes from the reduction in processing time spent in the socket layers.

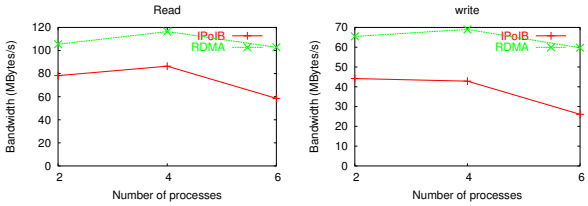

Fig. 9. MPI Tile I/O Read and Write bandwidth

5 Impact of PCI-Express

We have also evaluated the impact of PCI-Express on NFS/RDMA. PCI-Express x8 rectifies the bandwidth restrictions of PCI-X and allows InfiniBand 4X to achieve higher aggregate bandwidth [14]. It also offers lower latency. We have used IOzone [2] with multiple threads to measure the Write throughput. Since Read's may be cached as discussed in Section 3.2, we do not present these numbers here. In this experiment, two sets of machines were used. The first set of machines has PCI-Express HCA's. The second set of machines has PCI-X HCA's. One machine in each set acted as the NFS server and the other machine acted as the client. On the client IOzone with multiple threads was used. Since this is a multi-threaded test, the directio option was used. Directio bypasses the systems buffer cache. This helps reduce copying costs at the client, allowing us to increase the number of threads. The number of threads on the client was varied from 1 to 16. A filesize of 32M (because of restrictions in available server memory size) with a record size of 16K was chosen. tmpfs was used as the backend storage. From Figure 11, we can see write throughput with PCI-X saturate at eight threads. However, the write throughput with PCI-Express increases (24% better than PCI-X at 16 threads). This is because of the higher number of outstanding RDMA Read's allowed by the PCI-Express HCA's, which allows more thread requests to be processed by the server.

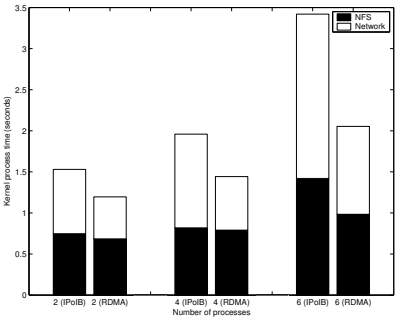

Fig. 10. MPI Tile I/O execution time breakdown (write)

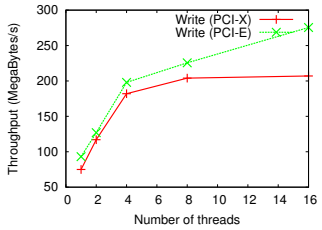

Fig. 11. Write throughput of IOzone with PCI-X and PCI-Express while varying the number of threads

6 Related Work

The performance of network file-access protocols has been the topic of a lot of research, covering different aspects of network file systems, such as caching strategies, sensitivity to networking parameters and comparisons of NFS to other network-based storage solutions. Xu et. al. [21] investigated the performance benefits of client caching to concurrent read sharing over NFS. Peng et. al. [18] showed that a network-centric reorganization of the buffer cache can improve the NFS performance. Radkov et. al. [16]

compared the performance of file-based NFS protocol and block-based iSCSI protocol and noted that aggressive metadata caching can benefit the NFS protocol.

Martin et. al. [15] studied the sensitivity of NFS to high performance networks by introducing controlled delays into live systems in the late 90's. They that observed NFS was more sensitive to processor overhead rather than networking latency and bandwidth. However, the emergence of high speed networks with direct access protocols such as RDMA lead to both the design of new network file system, such as DAFS [11], and the revision of traditional network file systems, such as NFS [10], to enable file accesses over RDMA-capable networks. For example, Goglin et. al. [12] replaced the RPC protocol of NFS with Myrinet GM protocol to achieve Optimized Remote File System Accesses (ORFA). Callaghan et. al. [9] provided an initial implementation NFS over RDMA on Solaris. Our work continues this endeavor and studies the performance benefits of NFS over RDMA on top of InfiniBand [13].

7 Conclusions and Future Work

In this paper, we have studied the performance of NFS/RDMA on InfiniBand as compared to NFS running over TCP (IPoIB) with a variety of benchmarks. These results show that NFS/RDMA depends critically on the performance of the back-end file system. With an enhanced form of the back-end temporary file-system (tmpfs), on the same underlying transport, there is often an order of magnitude improvement in performance of up to 11 times. The enhanced tmpfs shows that RDMA can improve running time up to 73% for a kernel compile and boost bandwidth by up to 126% for MPI Tile I/O. In addition, experiments show reduced CPU utilization with NFS/RDMA. There is also improvement for other benchmarks and applications. Finally, with multiple threads, PCI-Express can scale IOzone aggregate Write bandwidth better than PCI-X. With PCI-Express at 16 threads on NFS/RDMA, there is a 24% improvement in IOzone Write bandwidth compared to PCI-X.

As part of the future work, we would like to explore design enhancements to the NFS/RDMA implementation. These include enhancing the design to make better use of RDMA Read and Write operations at both the client and the server. This would potentially boost performance in addition to reducing the load at the server and enhancing security.

References

[1] General Information and References for the NFSv4 protocol. In *http://www.nfsv4.org/*.

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

[3] MPI over InfiniBand Project. In *http://nowlab.cse.ohio-state.edu/projects/mpi-iba/*.

[4] OpenIB Consortium. In *http://www.openib.org*.

[5] The Open Solaris Project. In *http://www.opensolaris.org*.

[6] Vnodes: An Architecture for Multiple File System Types in Sun UNIX. In *1986 Summer USENIX Conference*.

[7] PostMark: A New File System Benchmark. Tech. Rep. TR3022, october 1997.

[8] A. Ching, et.al. Efficient Structured Data Access in Parallel File Systems. In *IEEE Cluster Computing*, 2003.

[9] 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: Experience, Lessons, Implications*, 2003.

[10] B. Callaghan and T. Talpey. RDMA Transport for ONC RPC. http://www1.ietf.org/proceedings_new/04nov/IDs/draft-ietf-nfsv4-rpcrdma-00.txt, 2004.

[11] M. DeBergalis, P. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, and M. Wittle. The Direct Access File System. In *Proceedings of Second USENIX Conference on File and Storage Technologies (FAST '03)*, march 2003.

[12] B. Goglin and L. Prylli. Performance Analysis of Remote File System Access over a High-Speed Local Network. In *Workshop on Communication Architecture for Clusters, in Conjunction with International Parallel and Distributed Processing Symposium '04*, April 2004.

[13] Infiniband Trade Association. http://www.infinibandta.org.

[14] J. Liu, A.Mamidala, A. Vishnu and D. K. Panda. Performance Evaluation of InfiniBand with PCI Express. *Hot Interconnect 12 (HOTI 04)*, August 2004.

[15] R. P. Martin and D. E. Culler. NFS Sentivity to High Performance Networks. In *ACM Sigmetrics*, 1999.

[16] P. Radkov. A Performance Comparision of NFS and iSCSI for IP-Networked Storage. In *FAST*, 2004.

[17] P. Wong. NAS Parallel Benchmarks I/O Version 2.4. In *Technical Report NAS-03-002, Computer Science Corporation, NASA Advanced Supercomputing (NAS) Division*, 2004.

[18] G. Peng, S. Sharma, and T. Chiueh. A case for network-centric buffer cache organization. In *Hot Interconnect 11*, August 2003.

[19] R. Ross. Parallel I/O Benchmark Consortium. In *http://www-unix.mcs.anl.gov/rross/pio-benchmark/html/*.

[20] W. Gropp, et.al. Using MPI-2: Advanced Features of the Message-Passing Interface. In *MIT Press*, 1999.

[21] 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.