

# Designing a High-Performance Clustered NAS: A Case Study with pNFS over RDMA on InfiniBand\*

Ranjit Noronha, Xiangyong Ouyang, and Dhabaleswar K. Panda

Network Based Computing Lab  
Department of Computer Science and Engineering  
The Ohio State University, Columbus, OH 43210  
{noronha, ouyangx, panda}@cse.ohio-state.edu

**Abstract.** Large scale scientific and commercial applications consume and produce petabytes of data. This data needs to be safely stored, cataloged and reproduced with high-performance. The current generation of single headed NAS (Network Attached Storage) based systems such as NFS is not able to provide an acceptable level of performance to these types of demanding applications. Clustered NAS have evolved to meet the storage demands of these demanding applications. However, the performance of these Clustered NAS solutions is limited by the communication protocol being used, usually TCP/IP. In this paper, we propose, design and evaluate a clustered NAS; pNFS over RDMA on InfiniBand. Our results show that for a sequential workload on 8 data servers, the pNFS over RDMA design can achieve a peak aggregate Read throughput of up to 5,029 MB/s, a maximum improvement of 188% over the TCP/IP transport and a Write throughput of 1,872 MB/s; a maximum improvement of 150% over the corresponding TCP/IP transport throughput. Evaluations with other type of workloads and traces show an improvement in performance of up to 27%. Finally, our design of pNFS over RDMA improves the performance of BTIO relative to the Lustre file system.

## 1 Introduction

The explosive growth in multimedia, Internet and other content have caused a dramatic increase in the volume of media that needs to be stored, cataloged and accessed efficiently. In addition, high-performance applications on large supercomputers process and create petabytes of application and checkpoint data. Modern single-headed nodes with a large number of disks (single headed Network Attached Storage (NAS)) may not have the adequate capacity to store this data. Also, the single head or single server may potentially become a bottleneck with accesses from a large number of clients. Also, a failure of the node or the disk may lead to a loss of data.

To deal with several of these problems, clustered NAS solutions have evolved. Clustered NAS solutions attempt to store the data across a number of storage servers. This

---

\* This research is supported in part by Department of Energy's grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation's grants #CNS-0403342 and #CCF-0702675; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Cisco Systems, Intel, Mellanox, QLogic, Sun Microsystems and Linux Networks; and equipment donations from Advanced Clustering, AMD, Appro, Intel, Mellanox, Microway, Qlogic and Sun Microsystems.

has a number of benefits. First, we are no longer limited to the capacity of a single node. Second, depending on the way data is striped across the nodes, with accesses from a large number of clients, the load will be more evenly distributed across the servers. Third, for large files, this architecture has the advantages of multiple streams of data from different nodes for better aggregate bandwidth for larger file sizes. Finally, clustered NAS allows data to be stored redundantly across a number of different nodes [1], reducing the likelihood of data loss.

Even though clustered NAS provides several benefits in term of capacity, enhanced load capacity, better aggregate throughput and better fault-tolerance, they bring with them their own set of unique problems. First, since the data-servers have now been de-coupled, any given stream of data will require multiple network, usually TCP/IP, connections from the clients to the data servers and metadata servers. TCP/IP connections have been shown to have considerable overhead, mainly in terms of copying costs, fragmentation and reassembly, reliability and congestion control. In addition, with multiple streams of incoming data from multiple data-servers, TCP/IP connections have been shown to suffer from the problem of incast [2], which severely reduces the throughput. Second, TCP/IP with multiple copies and considerable overhead is unable to take advantage of the high-performance networks like InfiniBand and 10GigE. Third, with a single headed NAS, there is only a single point of failure, making it easier to protect the data on the NAS. However, with a clustered NAS, we have multiple data servers, with multiple failure points.

Modern high-performance networks such as InfiniBand provide low-latency and high-bandwidth communication. For example, the current generation ConnectX NIC from Mellanox has a 4 byte message latency of around  $1\mu s$  and a bi-directional bandwidth of up to 4 GB/s for large messages. Applications can also deploy mechanisms like Remote Direct Memory Access (RDMA) for zero-copy low-overhead communication. RDMA operations allow two appropriately authorized peers to read and write data directly from each other's address space. RDMA requires minimal CPU involvement on the local end, and no CPU involvement on the remote end. Designing the stack with RDMA may eliminate the copy overhead inherent in the TCP and UDP stacks and reduce CPU utilization. As a result, a high-performance RDMA enabled network like InfiniBand might potentially reduce the overhead of TCP/IP connections in clustered NAS.

In our earlier work, we designed a Network File System (NFS) (which is a single headed NAS) with RDMA operations in InfiniBand [3] for NFSv3 and NFSv4. In this paper, we propose, design and evaluate a clustered Network Attached Storage (NAS). This clustered NAS is based on parallel NFS (pNFS) with RDMA operations in InfiniBand. While other parallel and clustered file systems such as Lustre [1] exist, we choose pNFS since NFS is widely deployed and used. In this paper, we make the following contributions:

- An in-depth discussion of the tradeoffs in designing a high-performance pNFS with an RPC/RDMA transport.
- An understanding of the issues with sessions that provides exactly once semantics in the face of network faults and the trade-offs in designing pNFS with sessions over RDMA.
- A comprehensive performance evaluation with micro-benchmarks and applications of a RDMA enabled pNFS design.

Our evaluations show that by enabling pNFS with an RDMA transport, we can decrease the latency for small operations by up to 65% in some cases. Also, pNFS enabled with RDMA allows us to achieve a peak IOzone Write and Read aggregate throughput of 1,872 MB/s and 5,029 MB/s, respectively using a sequential trace with 8 data servers. The RDMA enabled Write and Read aggregate throughput is 150% and 188% better than the corresponding throughput with a TCP/IP transport. Also, evaluation with a Zipf trace distribution allows us to achieve a maximum improvement of up to 27% when switching transports from RDMA to TCP/IP. Finally, application evaluation with BTIO shows that the RDMA enabled transport with pNFS performs better than with a TCP/IP transport by up to 8.8% and better than Lustre by up to 22%.

The rest of the paper is presented as follows. Then, Section 2 looks at the parallel NFS and sessions extensions to NFSv4.1. Following that, Section 3 looks at the design considerations for pNFS over RDMA. After that, Section 4 evaluates the performance of the design. We present related work in Section 5. Finally, Section 6 discusses conclusions and future work.

## 2 NFSv4.1: Parallel NFS (pNFS) and Sessions

In this section, we discuss pNFS and sessions, which are defined by the NFSv4.1 semantics.

**Parallel NFS (pNFS):** The NFSv4.1 [4] standard defines two main components; namely parallel NFS (pNFS) and sessions. The focus of pNFS is to make an NFSv4.1 client a front-end for clustered NAS or parallel file-system. The pNFS architecture is shown in Figure 1. The NFSv4.1 client can communicate with any parallel file using the *Layout* and *I/O driver* in concert with communications with the NFSv4.1 server. The NFSv4.1 server has multiple roles. It acts as a metadata server (MDS) for the parallel/cluster file system. It sends information to the client on how to access the back-end cluster file system. This information takes the form of *GETDEVICEINFO*, which returns information about a specific data-server in the cluster file system, usually an IP address and port number that is stored by the client layout driver. The NFSv4.1 server is also responsible for communicating with the data servers for file creation and deletion. The NFSv4.1 server may either directly communicate with the data servers, or it may communicate with a metadata server, that is responsible for talking to and controlling the data servers in the parallel file system. The pNFS client uses the file layout and I/O driver for communicating with the data servers. The layout driver is responsible for translating READ and WRITE requests from the upper layer into the corresponding protocol that the back-end parallel/cluster file system uses; namely object, block and file. This is achieved through the additional NFS procedures *GETFILELAYOUT* (how the file is distributed across the data servers), *RETURNFILELAYOUT* (after a file is closed), *LAYOUTCOMMIT* (commit changes to file layout at the metadata server, after writes have been committed to data servers). Examples of pNFS designs are discussed further in the technical report [5].

**NFSv4.1 and sessions:** Sessions are aimed at making the NFSv4 non-idempotent requests resilient to network level faults. Traditionally, non-idempotent requests are taken

care of through the Duplicate Request Cache (DRC) at the server. The DRC has a limited number of entries, and these entries are shared among all the clients. So, eventually some entries will be evicted from the cache. In the face of network-level partitions, duplicate requests that arrive that have been evicted from the DRC, will be re-executed. Sessions solve this problem by requiring each connection to be allotted a fixed number of RPC slots in the DRC. The client is only allowed to issue requests up to the number of slots in the connection. Because of this reservation policy, duplicate requests from the client to the server in the face of network-level partitions will not be re-executed. We will consider design issues with sessions and RPC/RDMA in the following section.

**RPC/RDMA for NFS:** The existing RPC/RDMA design for Linux and OpenSolaris is based on the Read-Write design [3]. It consists of two protocols; namely the inline protocol for small requests and the bulk data transfer protocol for large operations. The inline protocol on Linux is enabled through the use of a set of persistent buffers; (32 buffers of 1K each for Send and 32 buffers of 1K each for receives on Linux). RPC Requests are sent using the persistent inline buffers. RPC replies are also received using the persistent inline buffers. The responses for some NFS procedures such as READ and READDIR might be quite large. These responses may be sent to the user via the bulk-data transfer protocol, which uses RDMA Write to send large responses from the server to the clients without a copy and RDMA Reads to pull data in from the client for procedures such as Write. The design trade-offs for RPC/RDMA are discussed further in [3].

### 3 Design Considerations for pNFS over RDMA

In this section, we examine the considerations for a high-performance design of pNFS over RDMA. First, we look at the detailed architecture of pNFS with a file layout driver.

#### 3.1 Design of pNFS Using a File Layout

As discussed in Section 2, pNFS can potentially use an object, block or file based model. In this paper, we use the file-based model for designing the pNFS architecture. We now discuss the high-level design of the pNFS architecture.

**pNFS Architecture:** The detailed architecture is shown in Figure 2. The NFSv4.1 clients use a file layout driver which is responsible for communicating with the NFSv4 servers, that act as the data-servers. At the NFSv4.1 server, the sPNFS daemon runs in user-space. The sPNFS daemon communicates with the NFSv4.1 server in the kernel via the RPC PipeFS. The RPC PipeFS is essentially a wait queue in the kernel. The NFSv4.1 server enqueues requests from the clients via the control path, and these requests are then pushed to the sPNFS daemon via an upcall. The sPNFS daemon then processes each of these requests and makes a downcall into the kernel with the appropriate data/response for the requests. The NFSv4.1 requests which are sent up to the sPNFS daemon for processing include the NFSv4.1 procedures *GETFILELAYOUT*, *RETURNFILELAYOUT*, *LAYOUTCOMMIT* and *GETDEVICEINFO*. These procedures are discussed in Section 2. In-order to work on the processing of the requests, the sPNFS

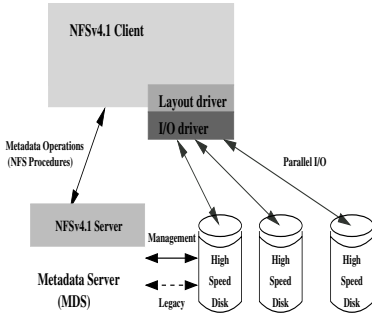


Fig. 1. pNFS high-level architecture

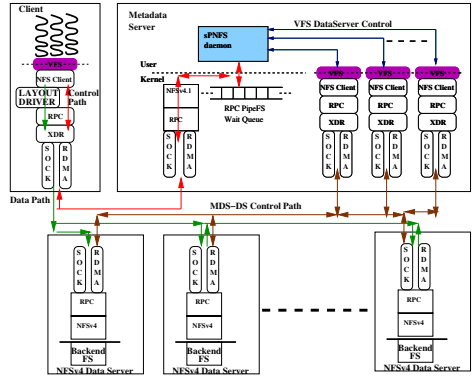


Fig. 2. pNFS detailed design

daemon mounts an NFSv3 directory from each of the data-servers. For example, when a file layout is requested (*GETFILELAYOUT*), the sPNFS daemon may need to create the file on each of the data servers or open the file through the *VFS DataServer Control Path*.

**sPNFS file creation:** To create a file, the sPNFS daemon will open the file on the mount of each of the data servers in create mode. It will then do a *stat* to make sure that the file actually got created or exists. It will then close the file (the file handle is static). This traffic will propagate via RPC calls through the *MDS-DS control path*. Finally, it will return the set of open file descriptor to the NFSv4.1 server as part of the response to the upcall. The NFSv4.1 server will then reply to the NFSv4.1 client with the file layout. The client will then use the layout received (through the file layout driver) to communicate with the NFSv4 data servers using the *Data Paths*.

**sPNFS file deletion:** File deletes are initiated by the NFS REMOVE procedure. The REMOVE procedure is sent up to the sPNFS daemon through RPC PipeFS. The process of deleting a file is opposite to that of creation. The sPNFS daemon will try to delete each of the file from the mount points. Once this is achieved, sPNFS will send a message to the NFS kernel thread about this.

### 3.2 RPC Connections from Clients to MDS (*Control Path*)

The RPC connections from the clients to the MDS may be through either RDMA or TCP/IP. A majority of the communication from the clients to the metadata server is expected to be small operations or metadata intensive workloads. As a result, these workloads may potentially benefit from the lower latency of RDMA. However, since NFS and RPC are in the kernel, there is the cost of a context switch from user-space to kernel-space, in addition to the copying costs with the NFS and RPC stacks. Depending on factors such as the CPU speed and memory bus-bandwidth, these costs might dominate. Correspondingly, the lower latency of RDMA might not provide much of a benefit in these cases. Another important factor that needs to be considered is the memory utilization and scalability of the MDS. The MDS is required to maintain RDMA enabled

RPC connections with all the clients. Each of these connection holds 32 1K send buffers and 32 2K receive buffers. These buffers are not shared across all the connections. With a very large number of client connections using RPC over RDMA, the MDS server might run out of buffers that might be appropriately utilized. In these cases, using RPC over TCP might be more appropriate for the majority of clients, though the high copying cost associated with TCP/IP connections needs to be considered. If an RDMA enabled RPC transport can provide adequate benefit for small operations, it might be appropriate to use a few connections with RDMA for some clients that communicate frequently with the MDS and a TCP enabled RPC transport for the remaining connections. A final factor that needs to be considered is the disconnect time for a RDMA enabled RPC transport. RDMA enabled RPC connections are disconnected after 2 minutes idle time. Reestablishing a RDMA enabled RPC connection is a very expensive operation because of the high-overhead of registering memory and reestablishing the eager protocol [3]. In comparison, RPC over TCP does not have such high-latencies for reestablishing the connections.

### 3.3 RPC Connections from MDS to DS (*MDS-DS Control Path*)

It might be potentially possible to use RPC over RDMA or RPC over TCP connections between the MDS and DSes. The *MDS-DS control path* allows the MDS to control the NFSv4 data-servers. This control is in the form of file creations and deletions. There are a number of factors that affect the choice of a RPC enabled with RDMA or TCP connection from the metadata server to the data servers. As discussed earlier, the sP-NFS daemon is multi-threaded. As a result, there are expected to be a large number of requests in flight, in parallel. So, the lower potential latency of RPC with RDMA is likely to provide a benefit in completion of these requests. Also, the fixed number of buffers per connection is expected to provide a better flow-control mechanism for a large number of outstanding parallel requests. Finally, the number of data servers is relatively small in comparison to the number of clients. As a result, the *MDS-DS control path* is not likely to be severely affected by the buffer scalability issue that may potentially affect the *Control Path*.

### 3.4 RPC Connections from Clients to Data Servers (*Data Paths*)

The expected traffic patterns from the client to the data servers is expected to consist of small, large and medium size traffic. Since 32K is the maximum payload for the cached I/O case, this is likely to be the most common transfer over the network, depending on the stripe size of the file at the data servers. We also need to consider the case of buffer scalability. Since data-servers are expected to have connections from a large number of clients, and since each connection will have persistent buffers, this might cause a memory scalability issue. However, clients do not connect to a particular data server unless the data server is in the list of DSes returned in the file layout. As a result, not all clients will be connected to all data servers at any given time. Depending on the load on the back-end file system, using an RPC over RDMA connection from the data-servers to the client might not cause a large amount of overhead at the data-servers. Also, quiescent clients will be disconnected from the data-servers, further reducing the

overhead. Since an RPC transport enabled with RDMA has been shown to provide considerable benefits via-viz large transfers, it might be beneficial to use RPC over RDMA between the clients and data-servers.

### 3.5 Sessions Design with RDMA

As discussed earlier, sessions provides exactly once semantics for all NFS procedures in the wake of network-level faults. To do this, sessions provide dedicated slots of buffers to each connection between the client and the servers. The client may only send requests up to a maximum number of slots per session. In order to design sessions with a RDMA enabled RPC transport, we associate the inline buffers in each connection with the minimum number of slots required from the connection. If the number of slots requested is lower than the number available, and the caller cannot accept a lower number, the session create request will fail. The disadvantages of the sessions design with RDMA is that advanced features of the InfiniBand network such as the Shared Receive Queue (SRQ) cannot be used. SRQ enhances the buffer scalability by having the buffers shared across all the InfiniBand connections. When the number of buffers falls below a certain watermark, an interrupt may be generated to post more buffers. Since sessions require that slots be guaranteed per connection, SRQ cannot be used.

## 4 Performance Evaluation

In this section, we evaluate the performance of pNFS designed with an RPC over RDMA transport. First, We discuss the experimental setup in Section 4.1. Following that, in Sections 4.2, 4.3 and 4.4, we look at the relative performance advantages of using an RDMA enabled RPC transport over a TCP/IP transport in different configurations involving the metadata server (MDS), Data Server (DS) and Client. Since sessions only requires reservation of RDMA inline buffers, we do not evaluate the sessions portion of the design.

### 4.1 Experimental Setup

To evaluate the performance of the RPC over RDMA enabled pNFS design (pNFS/RDMA), we used a 32-node cluster. Each node in the cluster is equipped with a dual Intel Xeon 3.6 GHz CPU and 2GB main memory. For InfiniBand communication, each node uses a Mellanox Double Data Rate (DDR) HCA. The nodes are equipped with SATA drivers, which are used to mount the backend ext3 filesystem. A memory based filesystem ramfs is also used for some experiments. The pNFS with sockets uses IP over InfiniBand (IPoIB) and we refer to this transport as pNFS/IPoIB. We use pNFS/IPoIB and pNFS/TCP interchangeably. All experiments using IPoIB are based on Reliable Connection mode (IPoIB-RC) and an MTU of 64KB, unless otherwise noted. We explicitly use pNFS/IPoIB-UD to explicitly mean an unreliable datagram mode of transport. IPoIB-UD uses a 2K MTU size.

## 4.2 Impact of RPC/RDMA on Performance from the Client to the Metadata Server

The clients communicate with the MDS using either NFSv4 or NFSv4.1 procedures. As Section 2 mentions, the vast majority of NFSv4.1 requests from the clients to the MDS are expected to be procedures such as *GETDEVICEINFO*, *GETDEVICELIST*, *GETFILELAYOUT* and *RETURNFILELAYOUT*. These small procedures will potentially carry small and medium size payloads. For example, *GETFILELAYOUT* returns a list of file handles, which is only a small amount of payload. A file handle can be encoded with no more than 16 bytes of information (although a native file handle size may vary depending on platforms). One of the largest deployments of a parallel file system Lustre [1] in recent times is the TACC [6] cluster with 8,000 nodes containing 64,000 cores, serviced by a bank of 1,000 data server nodes. With 1,000 data server nodes and the assumption that a file is striped across all the data server nodes, the payload from *GETFILELAYOUT* will only be 16K. Also, some of these operations such as *GETDEVICEINFO* are only executed at mount time and are not in the critical path. On the other hand, operations such as *CREATED*, *GETFILELAYOUT*, *RETURNFILELAYOUT* are executed every time a file is created, opened and closed. With a workload consisting of a large number of such operations (metadata intensive workloads) RPC/RDMA is likely to provide some benefit. Also, *LAYOUTCOMMIT* is executed once a WRITE operation completes and is likely to be in the critical path for workloads dominated by write operations. To understand the relative performance of small operations when switching transports from RPC/TCP to RPC/RDMA, we measured the latency of issuing a *GETFILELAYOUT* (at the RPC layer) from the client to the MDS and the time required for it to complete, averaged over 1024 times, while the payload from the MDS to the client was varied from 1 to 32K bytes. A 32K message can contain the information for more than 2,000 file handles and might be considered large for contemporary, high-performance parallel file system deployments. The measured latency is shown in Figure 3. As shown in Figure 3, the latency with a 1 byte payload is  $68\mu\text{s}$  with RPC/RDMA and  $71\mu\text{s}$  with RPC/TCP. The relatively low improvement in performance is because the high access latency of the disk which is a dominant portion of the latency. With larger access, the disk blocks are prefetched because of sequential access and the performance improvement from using RPC/RDMA is increased by up to 65%. The performance benefit of the RPC/RDMA connection from the client to the MDS is taken in the context of the inline buffers, which need to be statically allocated per-client at the MDS. With an increasing number of clients, the RPC/RDMA connections may consume considerable memory resources. Since the MDS is likely to be the target of a mainly metadata intensive workload, it becomes imperative to maintain a large number of inline buffers in order to guarantee a high throughput performance.

## 4.3 RPC/RDMA Versus RPC/TCP on Metadata Server to Data Server

The connection from the MDS to the DSeS may also consist of RPC/RDMA. The sPNFS daemon controls the DSeS by mounting the exported directories from the data servers. The sPNFS daemon creates, open and deletes files in the exported directories. These calls are translated through the VFS layer to RPC/RDMA calls. Thus the scalability of

these calls is directly impacted by the time required by the RPC operations to complete. To gain insight into the relative scalability of the RPC/RDMA and RPC/TCP transports, we measured the performance of create portion of the sPNFS daemons operation. In this multi-process benchmark, each process is synchronized in the start phase by a barrier. After being released from the barrier, each process performs a stat operation on the target file to check its state, then opens this file in creation mode. These two operations are followed by a chmod to set the mode of this file, and a close operation to close this file. The close operation is a portion of the process to open a pNFS file, and it is included to avoid running out of open file handles, a limited operating system resource. Each process performs each of these operations on every one of the DS mounts. The time required for 1024 of these operations is measured and averaged out. This test is performed for a RPC/RDMA and RPC/TCP transport from the MDS to the DSes. These numbers are shown in Figures 4 and 5. In Figures 4 and 5, we observe the following trends. RPC/RDMA performs worse than RPC/TCP (indicated as IPoIB) for 1 process. Note that in this case, we are measuring the time at the VFS level, whereas in Section 4.2, we are measuring the time at the RPC layer. In the current scenario, the IPoIB-RC driver uses a ring of 128 receive buffer of size 64K and 64 send buffers. On the other hand, RPC/RDMA uses 32-buffers of 1K. As a result, with an increasing number of data servers, and 1 process, more create and stat operations can be issued in parallel with IPoIB-RC than with RPC/RDMA (we issue 1,024 create operations and 1,024 stat operations for a total of 2,048 operations). However, with IPoIB-RC all 128 receive buffers are shared across all the connections using SRQ. With RPC/RDMA, each connection from an MDS to a DS is allocated a set of 32-buffers. As a result, when the number of connections increases, RPC/RDMA has a dedicated set of buffers in which to receive messages, while IPoIB has a fixed number of buffers, and this might result in dropped messages with IPoIB. Also in RPC/IPoIB, there are up to 5 copies from the application to the IP-level. With RPC/RDMA, there are up to 3 copies from the application down to the RPC/RDMA layer. With an increasing number of processes, the larger number of copies in the case of IPoIB begins to dominate and IPoIB performs worse than RDMA. The copying cost with IPoIB and 1 client does not totally consume the CPU and so is not the dominant factor. As a result, with 1 process, RPC/TCP is able to perform better than RPC/RDMA. At 2 processes per-node, RPC/RDMA and RPC/TCP

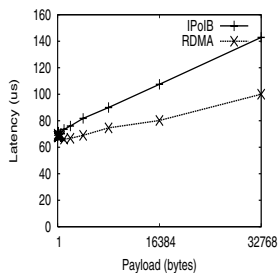


Fig. 3. Small Operations

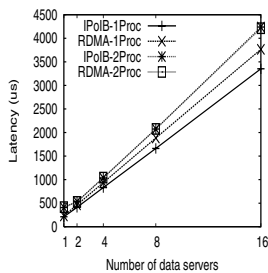


Fig. 4. Latency for 1 and 2 processes

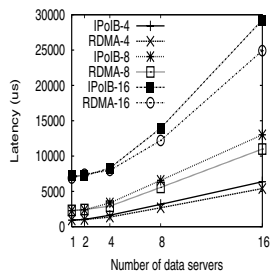


Fig. 5. Latency for 4,8 and 16 processes

perform comparably with an increasing number of data servers. At 4 processes/node and above with RPC/RDMA, the time required to perform the create operations is lower than RPC/TCP. At 16 processes at the MDS, the improvement with 16 DSEs there is a maximum decrease in latency of 15%. The trends we have observed indicate that RPC/RDMA will perform better than RPC/TCP with a larger volume of operations. We have conducted a test with 32 client threads with both RPC/RDMA and RPC/TCP. RPC/RDMA exhibits similar degree of improvement over RPC/TCP. Analysis shows that the SRQ used in IPoIB plays a role in the performance reduction with RPC/TCP and is discussed further in the technical report [5].

#### 4.4 RPC/RDMA Versus RPC/TCP from Clients to DS

We measure the relative performance impact of changing the transport from RPC/RDMA to RPC/TCP from the client to the data servers. To measure the performance impact, we use three different benchmarks: sequential throughput with IOzone, throughput of a Zipf trace and a parallel application BTIO.

**Sequential Throughput.** We use IOzone [7] in cluster mode to measure the performance of a sequential workload modeling the throughput from the client to the DSEs. 8 nodes act as data servers, 8 nodes act as clients, and 1 node is designated as the metadata server. Each client node hosts one IOzone process. The benchmark is run on both the IPoIB Reliable Connection mode (IPoIB-RC) and IPoIB Unreliable Datagram mode (IPoIB-UD) to compare against RPC/RDMA. The IOzone record size is kept at 32KB, the default cached I/O maximum size and the total file size per client used is 512MB. The Write and Read throughput while varying the number of data servers and clients (aggregate throughput) is shown in Figures 6 and 7, respectively.

For Write, RPC/RDMA begins to outperform RPC/TCP as the number of data server is increased beyond two. At 8 data servers and 8 clients, RPC/RDMA reaches its peak write throughput of 1,872 MB/s, which is 22% higher than IPoIB-RC and 150% higher than IPoIB-UD. For Read, there is an improvement in performance for all cases. Using RPC/RDMA achieves a peak read throughput of 5,029 MB/s at 8 clients and 8 data servers, which outperforms IPoIB-RC by 89% and IPoIB-UD by 188%.

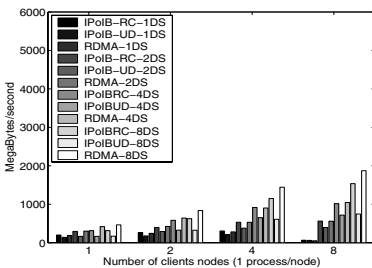


Fig. 6. IOzone Throughput (Write)

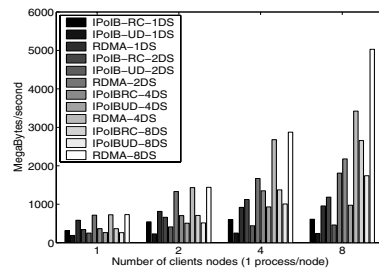


Fig. 7. IOzone Throughput (Read)

**Throughput with a Zip Trace.** Zipf's law, named after the Harvard linguistic professor George Kingsley Zipf (1902-1950), is the observation that frequency of occurrence of some event ( $P$ ), as a function of the rank ( $i$ ) when the rank is determined by the above frequency of occurrence, is a power-law function  $P_i \sim 1/i^\alpha$  with the exponent  $\alpha$  close to unity. Zipf distributions have been shown to occur in a variety of different environments such as word distributions in documents, web-page access patterns and file and block distributions in storage sub-systems [8].

We modified IOzone to issue write and read requests, where the size and location of the Read or Write request follows a Zipf distribution with an  $\alpha=0.9$ . We used IOzone to measure the throughput of the trace on a single node with one thread, issuing requests where the location and I/O size of the issued request follows a Zipf distribution. We used a 512MB file size on both an ext3 as well as a *ramfs* backend file system. The *ramfs* file system streams data from memory and serves as an upper bound on the performance improvement we can expect with pNFS/RDMA. We compare pNFS/RDMA with pNFS/IPoIB-RC while varying the number of data servers. The results for Write are shown in Figure 8, while the results for Read are shown in Figure 9. We observe that the RPC transport used does not have a large impact on performance for Writes. Disk Filesystem Write performance is generally sensitive to the performance of the backend storage subsystem. The large majority of disks exhibit poor random Write performance. Also, depending on the organization of the in-memory file system, *ramfs* based systems have also been shown to exhibit poor performance for random Write operations. Correspondingly, for the Zipf based Write distribution, we see a very poor throughput of around 500 MB/s for both pNFS/RDMA and pNFS/IPoIB-RC. Since the CPU is not fully utilized for TCP/IP when the throughput is lower, we expect little improvement from pNFS/RDMA over pNFS/IPoIB-RC. On the other hand, the IOzone Read throughput is impacted by the underlying RPC transport. With a *ramfs* based file system, we see an improvement of 22% from pNFS/IPoIB-RC to pNFS/RDMA with 1 data server. The improvement in throughput from pNFS/IPoIB-RC to pNFS/RDMA increases to 27% at 8 data servers. We are also able to achieve a peak throughput of 2073 MB/s with the Zipf trace at 8 data servers. Since, the Zipf trace has an element of randomness, a portion of the Read data is cached at the client. As a result we see some amount of cache effect in addition to network-level transfers, which reduces the potential performance improvement with pNFS/RDMA. Techniques for improving the performance of pNFS/RDMA for a Zipf workload are discussed further in the technical report [5].

**Performance with a Scientific Kernel NAS BTIO.** The NAS Parallel Benchmarks (NPB) suite discussed further in the technical report [5], is used to measure the performance of Computational Fluid Dynamic (CFD) codes on a parallel machine. One of the benchmarks BT measures the performance of block-triangulation equations in parallel. In addition to the computational phase of BT, BTIO adds additional code for checkpointing and verifying the data from the computation. We use the Full MPI I/O mode in which MPI collective calls are used to aggregate Read and Write operations. We run BTIO with a class A size (that uses a 64x64x64 array) over pNFS/RDMA, pNFS/IPoIB and Lustre. In this configuration, BTIO generates a file of size 400 MB. The results with an ext3 back-end file system at eight data servers are shown in Figure 10.

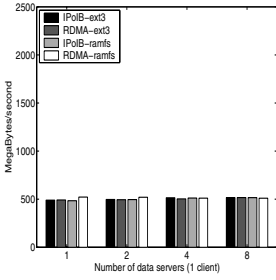


Fig. 8. Zipf trace (Write)

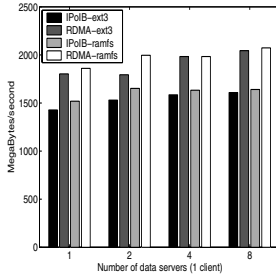


Fig. 9. Zipf trace (Read)

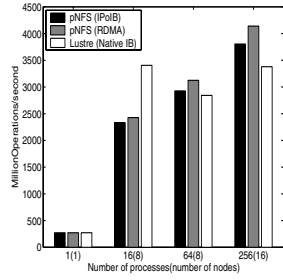


Fig. 10. BTIO with ext3

In this experiment, we measured the performance of BTIO with eight data servers. We used one, 16, 64 and 256 processes (BTIO requires a square number of processes). For the 16 process case, we use eight client nodes (two processes/node). For the 64 and 256 process case, we also use eight and sixteen client nodes respectively (eight processes per node and 16 processes/node respectively). We observe the following trends. First, pNFS/IPoIB-RC and pNFS/RDMA perform comparably at one process on one node. As the number of processes increases, pNFS/RDMA begins to perform better than pNFS/IPoIB-RC. At 16 processes (two processes/node), BTIO over a pNFS/RDMA transport performs up to 4% better than over a pNFS/IPoIB-RC transport. At 64 processes (eight processes/node) and 256 processes (16 processes/node), this increases to approximately 7% and 8.8%, respectively. The better bandwidth of the underlying transport helps improve MPI collective I/O performance and is discussed further in the technical report [5]. Finally, we also compared with Lustre using a native InfiniBand transport and eight data servers. pNFS/RDMA outperforms Lustre by up to 22% at 256 processes.

## 5 Related Work

There are a large number of single headed NAS, clustered NAS storage system and parallel file-systems. Network File System (NFS) is one of the most popular single headed NAS systems. RPC over RDMA transport for NFS exists on both OpenSolaris and Linux [3]. In our work, we design an RPC over RDMA transport for parallel NFS. Lustre [1] is another popular parallel file system. It also allows access to native InfiniBand through the IB Network Access Layer (NAL). The native InfiniBand NAL uses RDMA operations. Our work differs from the IB NAL of Lustre in that we design RPC directly over RDMA, whereas Lustre uses RPC over portals, which in turn calls the NAL functionality.

## 6 Conclusions and Future Work

In this paper, we propose, design and evaluate a high-performance clustered NAS. The clustered NAS uses parallel NFS (pNFS) with an RDMA enabled transport. We consider a number of design considerations and trade-offs, in particular, buffer management at the client, DS and MDS, scalability of the connections with increasing number

of clients and data servers. We also look at how an RDMA transport may be designed with sessions which gives us exactly once semantics. Our evaluations show that enabling pNFS with a RDMA transport, we can decrease the latency for small operations by up to 65% in some cases. Also, pNFS enabled with RDMA allows us to achieve a peak IOzone Write and Read aggregate throughput of 1,800+ MB/s (150% better than TCP/IP) and 5,000+ MB/s (188% improvement over TCP/IP) respectively, using a sequential trace and 8 data servers. Also, evaluation with a Zipf trace distribution allows us to achieve a maximum improvement of up to 27% when switching transports from RDMA to TCP/IP. Finally, application evaluation with BTIO shows that the RDMA enabled transport with pNFS performs better than a transport with TCP/IP by up to 8.8% and better than Lustre by up to 22%.

As part of future work, we would like to explore how to design a fault tolerant pNFS enabled with RDMA. pNFS allows us to use multi-pathing to enable redundant data-servers. We would also like to explore how the shared receive queue (SRQ) optimization may be used with an RDMA enabled RPC transport that uses sessions. Sessions require the reservation of slots or RDMA eager buffers per RPC connection. Dedicating a fixed number of buffers might have an impact on the scalability of larger systems deployed with pNFS. Finally, we would like to evaluate the scalability of our RDMA enabled pNFS design.

## References

1. Zahir, R.: Lustre Storage Networking Transport Layer, <http://www.lustre.org/docs.html>
2. Phanishayee, A., Krevat, E., Vasudevan, V., Andersen, D.G., Ganger, G.R., Gibson, G.A., Seshan, S.: Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In: Proc. USENIX Conference on File and Storage Technologies, San Jose, CA (February 2008)
3. Noronha, R., Chai, L., Talpey, T., Panda, D.: Designing NFS with RDMA on InfiniBand for Security, Performance and Scalability. In: International Conference on Parallel Processing, Xian, China (2007)
4. Shepler, S., Eisler, M., Noveck, D.: NFS Version 4 Minor Version 1., <http://tools.ietf.org/html/draft-ietf-nfsv4-minorversion1-19>
5. Noronha, R., Ouyang, X., Panda, D.K.: Designing a High-Performance Clustered NAS: A Case Study With pNFS over RDMA on InfiniBand. Technical Report OSU-CISRC-5/08-TR28, Department of Computer Science and Engineering, The Ohio State University (2008)
6. Ranger: Sun Constellation Linux Cluster, <http://www.tacc.utexas.edu/resources/hpcsystems/>
7. IOzone Filesystem Benchmark, <http://www.iozone.org>
8. Batsakis, A., Burns, R., Kanevsky, A., Lentini, J., Talpey, T.: AWOL: An Adaptive Write Optimizations Layer. In: FAST 2008: Proceedings of the 6th USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, pp. 1–14. USENIX Association (2008)