Designing NFS with RDMA for Security, Performance and Scalability *

Ranjit Noronha[1] Lei Chai[1] Thomas Talpey[2] Dhabaleswar K. Panda[1]
[1] The Ohio State University
{noronha, chail, panda}@cse.ohio-state.edu
[2] Network Appliances
{Thomas.Talpey}@netapp.com

Abstract

NFS has traditionally used TCP or UDP as the underlying transport. However, the overhead of these stacks has limited both the performance and scalability of NFS. Recently, high-performance network such as InfiniBand have been deployed. These networks provide low latency of a few microseconds and high bandwidth for large messages up to 20 Gbps. Because of the unique characteristics of NFS protocols, previous designs of NFS with RDMA were unable to exploit the improved bandwidth of networks such as InfiniBand. Also, they leave the server open to attacks from malicious clients. In this paper, we discuss the design principles for implementing NFS/RDMA protocols. We propose, implement and evaluate an alternate design for NFS/RDMA on InfiniBand, which can significantly improve the security of the server, compared to the previous design. In addition, we evaluate the performance bottlenecks of using RDMA operations in NFS protocols and propose strategies and designs that tackle these overheads. With the best of these strategies and designs, we demonstrate throughput of 700 MB/s on the OpenSolaris NFS/RDMA design and 900 MB/s on the Linux design and an application level improvement in performance of up to 50%. We also evaluate the scalability of the RDMA transport in a multi-client setting, with a RAID array of disks. Our design has been integrated into the OpenSolaris kernel.

1. Introduction

The Network File System (NFS) protocol has become the *de facto* standard for sharing files among users in a distributed environment. Many sites currently have terabytes of storage data on their I/O servers. I/O servers with petabytes of data have also debuted. Fast and scalable access to this data is critical. The ability of clients to cache this data for fast and efficient access is limited, partly because of the demands on main memory on the client, which is usually allocated by memory hungry application such as in-memory database servers. Also, for medium and large scale clusters and environments, the overhead of keeping client caches coherent quickly becomes prohibitively expensive. Under these conditions, it becomes important to provide efficient low-overhead access to data from the NFS servers.

Modern high-performance networks such as InfiniBand provide low-latency and high-bandwidth communication. For example, the current generation Single Data Rate (SDR) NIC from Mellanox has a 4 byte message latency of less than $3\mu s$ and a bi-directional bandwidth of up to 2 GB/s for large messages. Applications can also deploy mechanisms like Remote Direct Memory Access (RDMA) for low-overhead communication. RDMA operations allow two appropriately authorized peers to read and write data directly from each others 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.

An initial implementation of NFS/RDMA [1] for the OpenSolaris operating system was designed by Callaghan, et.al.. This design allowed the client to read data from the server through RDMA Read. An important design consideration for any new transport is that it should be as secure as a transport based on TCP or UDP. Since RDMA requires buffers to be exposed, it is critical that only trusted entities be allowed to access these buffers. In most NFS deployments, the server may be considered trustworthy; the clients cannot be trusted. So, exposing server buffers makes the server vulnerable to snooping and malicious activity by the client. Callaghan's design exposed server buffers and therefore suffered from a security vulnerability. Also, inherent limitations in the design of RDMA Read reduce the number of RDMA Read operations that may be issued by a local peer to a remote peer. This throttles the number of NFS operations that may be serviced concurrently, limiting performance. Finally, Callaghan's design did not address the issue of multiple buffer copies. Our experiments with the original design of NFS/RDMA reveal that on two Opteron 2.2 GHz systems with x8 PCI-Express Single Data Rate (SDR) InfiniBand adapters capable of a unidirectional bandwidth of 900 MegaBytes/s (MB/s), the IOzone [8] multi-threaded

---

*This research is supported in part by grants from Sun Microsystems and Network Appliances, Department of Energy's grants #DE-FC02-06ER25755 and #DE-FC02-06ER25749, National Science Foundation grants #CNS-0403342, #CNS-0509452 and #CCF-0702675; grants from Mellanox; and equipment donations from Intel, Mellanox, and SUN Microsystems.

Read bandwidth saturates at just under 375 MB/s.

In this paper, we take on the challenge of designing a high performance NFS over RDMA for OpenSolaris. We discuss the principles for designing NFS protocols with RDMA. To this end we take an in-depth look at the security and buffer management vulnerabilities in the original design of NFS over RDMA on OpenSolaris. We also demonstrate the performance limitations of this RDMA Read based design. We propose and evaluate an alternate design based on RDMA Read and RDMA Write. This design eliminates the security risk to the server. We also look at the impact of the new design on buffer management.

We try to evaluate the bottlenecks that arise while using RDMA as the underlying transport. While RDMA operations may offer many benefits, they also have several constraints such as memory registration, that may essentially limit their performance, given the short bursty nature of NFS protocols. With these designs and performance optimizations in place, our experiments show that with appropriate registration strategies, an RDMA Write based design can achieve a peak IOzone Read throughput of over 700 MB/s on OpenSolaris and a peak Read bandwidth of close to 900 MB/s for Linux. Evaluation with an Online Transaction Processing (OLTP) workload show that the higher throughput of our proposed design can improve performance up to 50%. We also evaluate the scalability of the RDMA transport in a multi-client setting, with a RAID array of disks. This evaluation shows that the Linux NFS/RDMA design can provide an aggregate throughput of 900 MB/s to 7 clients, while NFS on a TCP transport saturates at 360 MB/s.

In this paper we make the following contributions:

- A comprehensive discussion of the design considerations for implementing NFS/RDMA protocols.
- A high performance implementation of NFS/RDMA for OpenSolaris, and a discussion of its relationship to a similar implementation for Linux.
- An in-depth performance evaluation of both designs.
- Design considerations for the relative limitations and potential solutions to the problem of registration overhead.
- Application evaluation of the NFS/RDMA protocols, and the impact of registration schemes such as Fast Memory Registration and All Physical Registration, and a buffer registration cache design on performance.
- Impact of RDMA on the scalability of NFS protocols with multiple clients and real disks supporting the back-end file system.

The rest of the paper is presented as follows. Section 2 provides an overview of the InfiniBand Communication model. Section 3 explores the existing NFS over RDMA architecture on OpenSolaris and the Linux. In Section 4, we propose our alternate design based on RDMA Read and RDMA Write and compare it to the original design based on RDMA Read only. Section 5 presents the performance evaluation of the design. We discuss related work in section 6. Finally, section 7 concludes the paper and looks at future work.

2 Overview of the InfiniBand Communication Model

InfiniBand primarily uses the Reliable Connection (RC) model. In this model, each initiating node needs to be connected to every other node it wants to communicate with through a peer-to-peer connection called a queue-pair (send and receive work queues). InfiniBand supports two-sided communication operations called *Channel Primitives*, that require active involvement from both the sender and receiver. One of the peers (receiver), posts a RDMA Receive (RV), that is matched to the corresponding RDMA Send (RS) from the sending peer. One-sided communication primitives, called *Memory Primitives*, do not require involvement by the receiver. Memory primitives RDMA Write (RW) allow one of the peers to directly write into the memory of the other peer, while RDMA Read (RR) allows it to directly read remote memory locations. The InfiniBand communication model is discussed further in [3].

A comparison of the different communication primitives in terms of Security (Receive Buffer Exposed), Involvement of the receiver (Receive Buffer Pre-Posted), Buffer protection (Steering Tag) and finally, Peer Message Exchanges for Receive Buffer Address and Steering Tag (Rendezvous) is shown in Table 1.

Table 1. Communication Primitive Properties

	Channel Primitives	Memory Primitives
Receive Buffer Exposed		✓
Receive Buffer Pre-Posted	✓	
Steering Tag		✓
Rendezvous		✓

3. Overview of NFS/RDMA Architecture

NFS is based on the single server, multiple client model. Communication between the NFS client and the server is via the Open Network Computing (ONC) remote procedure call (RPC). Callaghan et.al. designed an initial implementation of RPC over RDMA [1] for NFS, as shown in Figure 1. The RPC Call is prepended with the header shown in Figure 2 and generally being small will go as an *inline request* using RDMA Sends. Inline requests are discussed further in [8]. In the rest of the paper, we use the terms RPC/RDMA and NFS/RDMA interchangeably.

3.1. RDMA Protocol for bulk data transfer

NFS procedures such as READ, WRITE, READLINK and READDIR can transfer data whose length is larger than the inline data threshold. Also, the RPC call itself can be larger than the inline data threshold. The bulk data can be transferred in multiple ways. The existing approach is to use RDMA Read only and is referred to as the Read-Read design. Our approach is to use a combination of RDMA

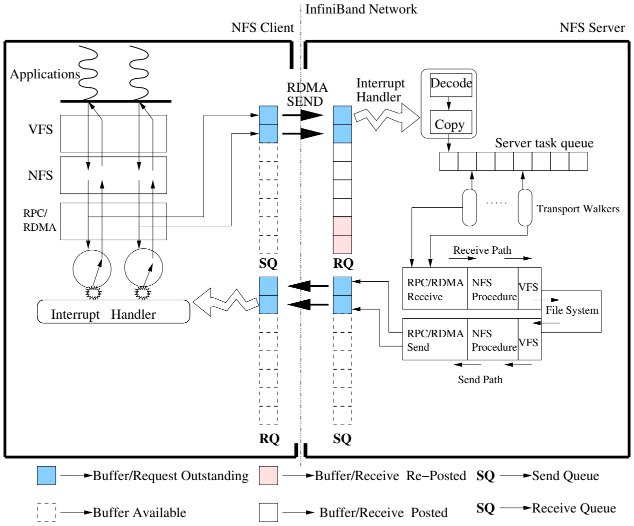


Figure 1. Architecture of the NFS/RDMA stack in OpenSolaris

Read and RDMA Write operations and is called the Read-Write design. We describe both these approaches in detail. Before we do that, we define some essential terminologies.

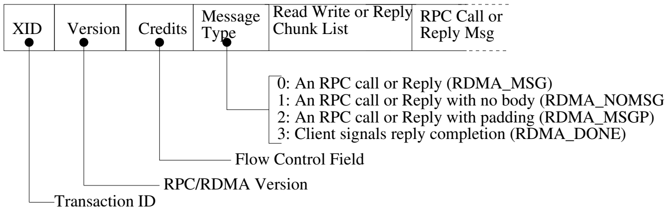
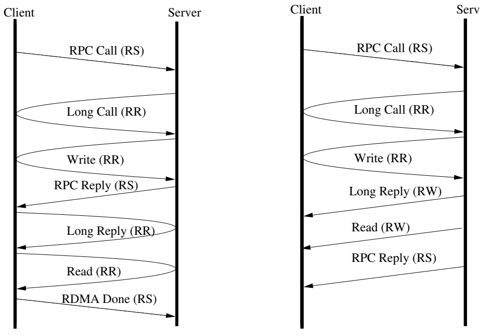


Figure 2. RPC/RDMA header

Chunk Lists: These lists provide encoding for bulk data whose length is larger than the inline data threshold or *inline threshold* and should be moved via RDMA. A chunk list consists of a single counted array of segments of one or more lists. Each of these lists is in turn a counted array of zero or more segments. Each segment encodes a steering tag for a registered buffer, its length and its offset in the main buffer. Chunks can be of different types; *Read chunks*, *Write chunks* and *Reply chunks*. *Read chunks* used in the Read-Read and Read-Write design encode data that may be RDMA Read from the remote peer. *Write chunks* used in the Read-Write design are used to RDMA Write data to the remote peer. *Reply chunks* used in the Read-Write design are used for procedures such as READDIR and READLINK, and are used to RDMA Write the entire NFS response.

The *RPC Long Call* is typically used when the RPC request itself is larger than the inline threshold. The *RPC Long Reply* is used in situations where the RPC Reply is larger than the inline size. Other bulk data transfer operations include *READ* and *WRITE*. All these procedures are discussed in the next section.



(a) Read-Read (b) Read-Write

Figure 3. RPC/RDMA Designs

4. Proposed Read-Write Design and Comparison to the Read-Read Design

In this section, we discuss our proposed Read-Write design, which is based on a combination of RDMA Read and RDMA Write. We also compare it with the original Read-Read based design, which is based on RDMA Read. We discuss the limitations of the Read-Read based design. Following that, we also discuss the advantages of the Read-Write design. We look at registration strategies and designs in section 4.3. The Read-Read based design is show in Figure 3(a). The Read-Write design is shown in Figure 3(b).

RPC Long Call: The RPC Long Call is typically used when the RPC request itself is larger than the inline threshold. In this case, the client encodes a chunk list along with a RDMA_NOMSG flag in the header shown in Figure 2. It is always combined with other NFS operations. The RPC Long Call is identical in both the Read-Read and Read-Write based designs. If the RPC Call message is larger than the inline size, the RPC Call from the client includes a Read Chunk List. The message type in the header in Figure 2 is set to RDMA_NOMSG. When the server sees an RDMA_NOMSG message type, it decodes the read chunks encoded in the RPC/RDMA header and issues RDMA Reads to fetch these chunks from the client. The data from these chunks constitutes the remainder of the header (the fields *Read, Write or Reply Chunk List* onwards in Figure 2, which are overwritten by the incoming data). The remainder of the header usually constitutes other NFS procedures and is then decoded.

NFS Procedure WRITE: The NFS Procedure WRITE is similar in both the Read-Read and Read-Write based designs. For an NFS procedure WRITE, the client encodes a Read chunk list. On the server side, these read chunks are decoded, the RDMA Reads corresponding to each segment are issued and the server thread blocks till the RDMA Reads complete. The operation is then handled by the NFS layer. Once the operation completes, control is returned to the RPC layer, that sends an RPC Reply via the inline protocol. In the simplest case, an NFS Procedure WRITE would generate an RPC Call (RS) from the client to the server, followed by the WRITE (RR) issued by the server to fetch the data from the client, and finally, the RPC Reply (RS) from the server to the client.

NFS Procedure READ: In the Read-Read design the NFS server needs to encode a *Read chunk list* in the RPC Reply for an NFS READ Procedure. The RPC Reply is then returned to the client via the inline protocol described earlier. The client decodes the Read chunk lists and issues the RDMA Reads. Once the RDMA Reads complete, the client issues an RDMA_DONE to the server, that allows it to free its pre-registered buffers. So, the simplest possible sequence of operations for an NFS Procedure READ is; RPC Call (RS) from the client to the server, followed by an RPC Reply (RS) from the server to the client, then a READ (RR) issued by the client to fetch the data from the server, and finally, an RDMA_DONE (RS) from the client to the server.

In the Read-Write design, for a NFS READ procedure, the client needs to encode a Write chunk list in the RPC Call. The server decodes and stores the Write chunk list. When the NFS procedure READ returns, the data is RDMA written back to the client. The server then sends the RPC Reply back to the client with an encoded Write Chunk List. The client uses this Write chunk list to determine how much data was returned in the READ call. So, the simplest possible protocol operations would be; RPC Call from the client to the server, then a Read (RW) from the server to the client, and finally, an RPC Reply (RS) from the server to the client.

NFS Procedure READDIR and READLINK (RPC Long Reply): The RPC Long Reply is typically used when the RPC Reply is larger than the inline threshold. The RPC Long Reply is used in both the Read-Read and Read-Write designs but the mechanisms are different. It may either be used independently, or combined with other NFS operations.

The design of the NFS procedure READDIR/READLINK in the Read-Read design is similar to the NFS Procedure READ in the Read-Read design. The server encodes a *Read chunk list* in the RPC Reply, that the client decodes. The client then issues RDMA Read to fetch the data from the server. Once the RDMA Reads complete, the client issues an RDMA_DONE to the server which allows the server to free its pre-registered buffers.

NFS Procedure READDIR and READLINK in the Read-Write design follows the design of the NFS READ procedure in the Read-Write design. The client needs to encode a Long Reply chunk list in the RPC Call. The server decodes and stores the Long Reply chunk list. When the NFS procedure returns, the server uses the long reply chunk to RDMA Write the data back to the client. The server then sends the RPC Reply back to the client with an encoded Long Reply Chunk List. The client uses this chunk list to determine how much data was returned in the READDIR/READLINK call. In the simplest case, an RPC Long Reply would entail the following sequence; RPC Call from the client to the server, then a Long Reply (RW) from the server to the client, and finally, an RPC Reply (RS) from the server to the client.

Zero Copy Path for Direct I/O for the NFS READ procedure: In addition to the basic design, we also introduce a zero copy mechanism for user space addresses on the NFS READ procedure path. This eliminates copies on the client side and translates into reduced CPU utilization on the client.

4.1. Limitations in the Read-Read Design

The Read-Read design has a number of limitations in terms of Security and Performance, and we discuss these issues in detail.

Security:

Server buffers exposed: An important design consideration for an RDMA enabled RPC transport is that it must not be less secure than other transports such as TCP. In the Read-Read design, the server side buffers are exposed for RDMA operations from the client. Since the steering tags are 32-bits in length, a misbehaving or malicious client might attempt to guess them and thereby possibly read a buffer for which it did not have access to.

Malicious or Malfunctioning clients: The client needs to send an *RDMA_DONE* message to the server to indicate that the buffers used for a Read or Reply chunk may be freed up. A malicious of malfunctioning client may never send the *RDMA Done* message, essentially tying up the server resources.

Performance:

Synchronous RDMA Read Limitations: The RDMA Read issued from the NFS/RDMA server are synchronous operation. Once posted, the server typically has to wait for the RDMA Read operation to complete. This is because the InfiniBand specification does not guarantee ordering between a RDMA Read and a RDMA Send on the same connection. This may add considerable latency to the server thread.

Outstanding RDMA Reads: The number of RDMA Read that can be typically serviced on a connection is governed by two parameters, the Inbound RDMA Read Queue Depth (IRD) and the Outbound RDMA Read Queue Depth (ORD). The IRD governs the number of RDMA Read that can be active at the remote peer; the ORD governs the number of RDMA Read that might be actively issued concurrently from the local peer. In the current Mellanox implementation of InfiniBand, the maximum allowed value for IRD and ORD is typically 8. So, parallelism is reduced at the server, especially for multi-threaded workloads.

4.2. Potential Advantages of the Read-Write Design

The key design difference between the Read-Read (Figure 3(a)) and Read-Write (Figure 3(b)) protocol is that RPC long replies and NFS READ data may be directly issued from the server. To enable these, the client needs to encode either a Write chunk list or a long reply chunk list (Section 3.1). Moving from a Read-Read based design to a Read-Write based design has several advantages. The Mellanox InfiniBand HCA has the ability to issue many RDMA

Write operations in parallel. This reduces the bottleneck for multi-threaded workloads. Also, since completion ordering between RDMA Write and RDMA Sends is guaranteed in InfiniBand, the server does not have to wait for the RDMA Writes from the long reply or the NFS READ operation to complete. The completion generated by the RDMA Send for the RPC Reply will guarantee that the earlier RDMA Writes have completed. This optimization also helps reduce the number of interrupts generated on the server. The RDMA_DONE message and its resulting interrupt is also eliminated. The generation of the send completion interrupt on the server is sufficient to guarantee that the RDMA operations from the buffers have completed and they may be deregistered. A similar guarantee also exists at the client, when an RPC Call message is received. The elimination of an additional message helps improve performance. Since the server buffers are no longer exposed and the client cannot initiate any RDMA operations to the server, the security of the server is now enhanced. One potential disadvantage of the Read-Write design is that the client buffers are now exposed and may be corrupted by the server. Since the server is usually a trusted entity in an NFS deployment, this issue is less of a concern. The final advantage of the Read-Write design is that the server no longer has to depend on the RDMA_DONE message from the client to deregister and release it buffers.

4.3. Proposed Registration Strategies For the Read-Write Protocol

InfiniBand requires memory areas to be registered for communication operations. Registration is a multi-stage operation. Registration involves assigning physical pages to the virtual area. Once physical pages have been assigned to the virtual area, the virtual to physical address translation needs to be determined. In addition, the physical pages need to be prepared for DMA operations initiated by the HCA. This involves making the pages unswappable by the operating system, by pinning them. The virtual memory system may perform both these operations. In addition, the HCA needs to be made aware of the translation of the virtual to physical addresses. The HCA also needs to assign a *steering tag* that may be sent to remote peers for accessing the memory region in RDMA operations. The virtual to physical translation and the steering tag are stored in the HCA's Translation Protection Table (TPT). This involves one transaction across the I/O bus. However, the response time of the HCA may be quite high, depending on the load on the HCA, the organization of the TPT, allocation strategies, overhead in the TPT, and so on. Because of the combination of these factors, registration is an expensive operation and may constitute a considerable overhead, especially when it is in the critical path. Deregistering a buffer requires the actions from registration to be done in reverse. The virtual and physical translations and steering tags need to be flushed from the TPT (this involves a transaction across the I/O bus). Once the TPT entries are invalidated, each of them is released. The pages may then be unpinned. If the physical pages were assigned to the virtual memory region at the time of registration, this mapping is torn down and the physical pages are released back into the memory pool. Registration cost is evaluated quantitatively in [8].

The registration/deregistration points in the Read-Write design are shown in Figure 4. For example, an NFS procedure READ requires a buffer registration at points 2 and 5, and a deregistration at points 8 and 10. From Figure 4, we can see that the registration overhead comes about mainly because the transport has to register the buffer and deregister the buffer on every operation at the client and server. The registration occurs once at the client, and then at the server in the RPC call path. Following that, deregistration happens once at the server, and then once at the client. To reduce the cost of memory registration, different optimizations and registration modes have been introduced. These include *Fast Memory Registration* and *Physical Registration*. In addition, we propose a buffer registration cache. We discuss these next.

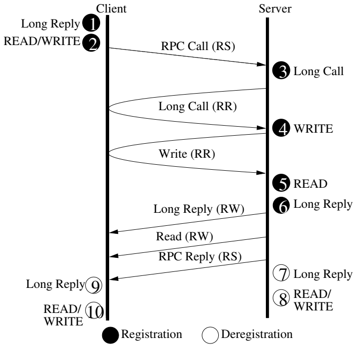


Figure 4. Registration points (Read-Write)

Fast Memory Registration (FMR): Fast Memory Registration allows for the allocation of the TPT entries and steering tags at initialization, instead of at registration time. The other operations of memory pinning, virtual to physical memory address translations and updating the HCA's TPT entries remain the same. The allocated entries in the TPT cache are then mapped to a virtual memory area. This technique is therefore not dependent on the response time of the HCA to allocate and update the TPT entries and consequently, may be considerably faster than a regular registration call. The limitations of FMR include the fact that it is restricted to privileged consumers (kernel), and the fact that the maximum registration area is fixed at initialization.

The Mellanox implementation of FMR introduces additional optimizations to the InfiniBand specification, which are discussed in the technical report [8]. We have incorporated FMR calls (Mellanox FMR) in the regular registration path in RPC/RDMA. To allow FMR to work transparently, we use a fall-back path to regular registration calls in case the memory region to be registered is too large.

Design of the Buffer Registration Cache: An alternate registration strategy is to create a buffer registration cache. A registration cache [10] has been shown to considerably improve communication performance. Most registration caches have been implemented at the user level and cache virtual addresses. Caching virtual addresses has been shown to cause incorrect behavior in some cases [7]. Also, unless static limits are placed on the number of entries in the registration cache, the cache tends to expand endlessly, particularly in the face of applications with poor buffer reuse patterns. Finally, static limits may perform poorly depending on the dynamics of the application.

To alleviate some of these deficiencies, we have designed an alternate buffer registration cache on the server. As shown in Figure 1, the NFS server state machine is split into two parts. The first part is on the RPC Call receive path where the NFS call is received and is issued to the file system. The second component is on return of control from the file system. Buffer allocation is done when the request is received on the server side and registration is executed when control returns from the file system. To model this behavior, we override the buffer allocation and registration calls and feed them to the registration cache module. This module allocates buffers of the appropriate size from a slab cache, for the request and then registers them when the registration request is made. If the buffer from the cache is already registered, no registration cost is encountered. The advantages of this setup are that the cache is no longer based on virtual address, and it is also linked to the systems slab cache, that may reclaim memory as needed. Since the server never sends a virtual address or steering tag to the client for any buffers in the registration cache, this is as secure as regular registration. The server registration cache scheme described above can also be applied to the client side, as discussed in the technical report [8].

All Physical Memory Registration: In addition to virtual addresses, communication in InfiniBand may also take place through physical addresses using the *Global Steering Tag* optimization. The *Global Steering Tag* available to privileged consumers (such as kernel processes) allows communication operations to use a special remote steering tag. The communication operation must use physical addresses. The consumer must pin the memory before communication starts and obtain a virtual to physical mapping, but does not need to register the mapping with the HCA. All Physical Registration should be used in environments where there is confidence in the integrity of the server. This is discussed further in the technical report [8].

5. Experimental Evaluation

In this section, we evaluate our proposed RDMA design with NFSv3. We first compare the Read-Write design with the existing Read-Read design on OpenSolaris in Section 5.1 (Linux did not have a Read-Read design). Following that, Section 5.2 discusses the impact of different registration strategies on NFS/RDMA performance, both at the microbenchmark and at the application-level. Finally, in Section 5.3 we discuss how RDMA affects the scalability of NFS protocols in an environment where the server stores the data on a back-end RAID array and services multiple clients.

5.1. Comparison of the Read-Read and Read-Write Design

Figures 5 and 6 show the IOzone [8] Read and Write bandwidth respectively with direct I/O on OpenSolaris. Performance of the Read-Read design are shown as RR. Performance of Read-Write design are shown as RW. The results were taken on dual Opteron x2100's with 2GB memory and Single Data Rate (SDR) x8 PCI-Express InfiniBand Adapters. These systems were running OpenSolaris build version 33. The back-end file system used was tmpfs which is a memory based file system. The IOzone file size used was 128 MegaBytes to accommodate reasonable multi-threaded workloads (IOzone creates a separate file for each thread). The IOzone record size was varied from 128KB to 1MB.

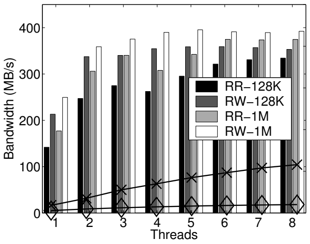


Figure 5. IOzone Read Bandwidth on Solaris

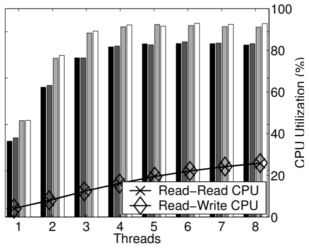


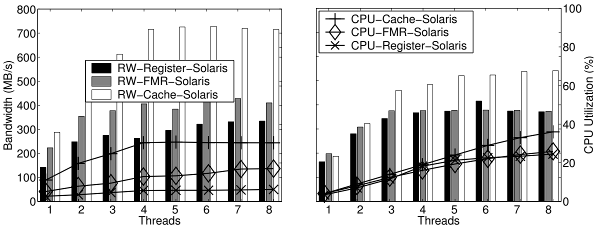
Figure 6. IOzone Write Bandwidth on Solaris

The Read-Write design performs better than the Read-Read design for all record sizes, for the READ procedure. The improvement in performance is approximately 47% with one thread at a record size of 128 KB, but decreases to about 5% at 8 threads. This improvement is primarily due to the elimination of the RDMA_DONE message as well as the improved parallelism of issued RDMA Writes from the server. The READ bandwidth for the Read-Read design saturates at 375 MB/s; the Read-Write design saturates at 400 MB/s. The client CPU utilization (we show only a single line for both record sizes) for the Read-Write design (NFS READ procedure) remains flat starting at only 2% at 1 thread increasing to about 5% at 8 threads. On the other hand, the CPU utilization for the Read-Read design increases from about 4% at 1 thread to about 24% at 8 threads. This is primarily because of elimination of data copies on the client direct I/O path in the Read-Write design. These results are discussed in detail in [8].

5.2. Impact of Registration Strategies

From section 4.3, we see that registration can constitute a substantial overhead in the RPC/RDMA transport. We evaluate the impact of Fast Memory Registration (FMR) and buffer registration cache at the micro-benchmark and application-level. We also look at the performance benefits from the All Physical Registration mode in Linux.

Fast Memory Registration (FMR): We now look at the impact of FMR discussed in section 4.3 on RPC/RDMA performance. The maximum size of the registered area was set to be 1MB. In addition, the FMR pool size was set to 512, which is sufficient for up to 512 parallel requests of 1MB. We evaluate the IOzone read and write bandwidth. Since the bandwidth from the different record sizes are similar, we present results with only a 128KB record size and a 128 MB file size. The results are shown in Figure 7(a) and Figure 7(b). FMR can help improve Read bandwidth from about 350 MB/s to approximately 400 MB/s, though this comes at the cost of increased client CPU utilization. Improvement in write bandwidth is modest, mainly because the time saving from the reduction in registration cost is dwarfed by the serialization of RDMA Reads (section 4.1).



(a) Read (b) Write

Figure 7. IOzone Bandwidth with different registration strategies on OpenSolaris

Buffer Registration Cache: The performance impact of the server registration cache on the IOzone Read and Write bandwidth is shown in Figure 7(a) and Figure 7(b) respectively. The registration cache dramatically improves performance for both the Read and Write bandwidth which goes up to 730 MB/s and 515 MB/s, respectively. The client CPU utilization is also increased, though this is to be expected with an increasing operation rate from the client. Again, the limited number of outstanding RDMA Reads bounds the improvement in Write throughput.

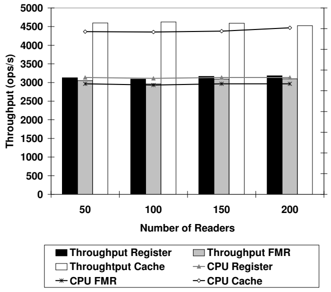
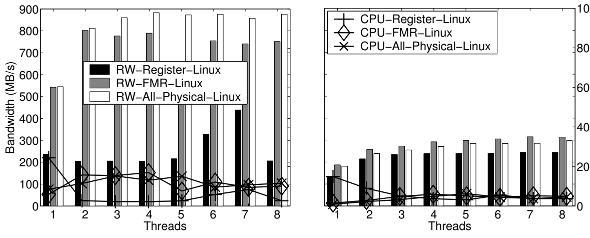


Figure 8. FileBench OLTP Performance

Impact of registration schemes on application performance: To evaluate the impact of memory registration schemes on application performance, we have conducted experiments using the online transaction processing (oltp) workload from FileBench [8]. We tune the workload to use the mean I/O size equal to 128KB. The results are shown in Figure 8. The bars represent the throughput (operations/sec) and the lines represent the client CPU utilization (cpu/operation). From Figure 8 we can see that the registration cache scheme improves throughput by up to 50% compared with the dynamic registration scheme. This indicates that the improvement in raw read/write bandwidth has been translated into application performance. The CPU utilization is slightly higher as expected. The FMR scheme performs comparably with the dynamic registration scheme in this benchmark.

All Physical Memory Registration: From Figure 9(a) we can see that the *all physical memory registration* mode yields the best Read throughput on Linux. It degrades the Write performance compared with the FMR mode as shown in Figure 9(b) because in all-physical mode the client cannot do local scatter/gather and so has to build more read chunks, therefore, each write request issues multiple RDMA Reads from the server that hits the limit of incoming/outgoing RDMA Reads in InfiniBand.

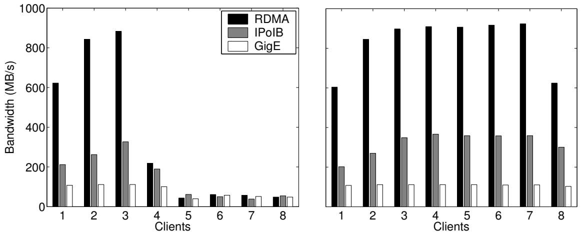


(a) Read (b) Write

Figure 9. IOzone Bandwidth with different registration strategies on Linux

5.3. Multiple Clients and Real Disks

In this section, we discuss the impact of RDMA on an NFS setup with multiple clients. We use the Linux NFS/RDMA design with the *All Physical Memory Registration* mode described in Section 4.3 for multiple client experiments. The server and clients are dual Intel 3.6 Xeon boxes with an InfiniBand DDR HCA. The clients have 4GB of memory. The server was configured with 4GB and 8GB of memory for each of the experiments below. The server has eight HighPoint SCSI disks with RAID-0 stripping, formatted with the XFS file system,with each disk capable of 30 MB/s. Further details are available in [8]. A 1GB file size per process with a 1MB record size is used for all the experiments. We compare the aggregate Read bandwidth of the Linux NFS/RDMA (RDMA) implementation with the regular NFS implementation over TCP on InfiniBand (IPoIB) and Gigabit Ethernet (GigE). Figure 10(a) shows the IOzone read bandwidth with multiple clients and a server with 4GB main memory. RDMA and IPoIB reach a peak aggregate bandwidth at three processes. RDMA peaks at 883 MB/s, while IPoIB reaches 326 MB/s. In comparison, GigE saturates at 107 MB/s with a single process and then the aggregate bandwidth goes down as the number of processes increases. The limited bandwidth of Gigabit Ethernet (peak theoretical bandwidth of 125 MB/s) may become a bottleneck with future high performance disks and server with large amount of memory. Figure 10(b) shows the IOzone read bandwidth with 8GB on the server. RDMA is able to maintain a peak bandwidth of above 900 MB/s up to seven threads, while IPoIB saturates at about 360 MB/s. From Figures 10(a) and 10(b), we can conclude that NFS/RDMA is limited by the ability of the back-end server to service data requests. NFS/TCP is a bottleneck on current generation systems.



(a) 4GB

(b) 8GB

Figure 10. Multiple clients IOzone Read Bandwidth

6. Related Work

The emergence of high speed networks with direct access protocols such as RDMA lead to both the design of new network file system and the revision of traditional network file systems to enable file accesses over RDMA-capable networks such as iSER, an extension for Internet Small Computer Systems Interface (iSCSI) protocol [4] and DAFS [5], a user space file system library. Memory registration optimizations, such as pre-registered buffers, are used in DAFS. Goglin et. al. [2] replaced the RPC protocol of NFS with Myrinet GM protocol to achieve Optimized Remote File System Accesses (ORFA). Callaghan et. al. [1] provided an initial implementation NFS over RDMA on Solaris. This work has identified the security and performance shortcomings in the work done by Callaghan et. al. [1] and proposed alternate designs.

7 Conclusions and Future Work

In this paper, we have designed and evaluated an NFS/RDMA protocol for high performance RDMA networks such as InfiniBand. This design is based on a combination of RDMA Read and RDMA Write. The design principles considered include NFS server security, performance and scalability. To improve performance of the protocol, we have incorporated several different registration mechanisms into our design. Our evaluations show that, the NFS/RDMA design can achieve throughput, close to that of the underlying network and improve throughput of an OLTP workload by 50%. Finally, we also studied the scalability of NFS/RDMA with multiple clients. This evaluation shows that the Linux NFS/RDMA design can provide an aggregate throughput of 900 MB/s to 7 clients, while NFS on a TCP transport saturates at 360 MB/s. We observe that a TCP transport is itself a bottleneck when servicing multiple clients. By comparison, NFS/RDMA is able to maintain throughput even with multiple clients; provided the back-end file system is able to sustain it. As part of future work, we would like to study buffer management and credit flow control schemes to further enhance the multi-client scalability of our NFS/RDMA design.

Software Distribution: The proposed NFS/RDMA design has been incorporated into the OpenSolaris kernel, and may be downloaded from [6, 9].

Acknowledgements

We would like to thank Dr. Weikuan Yu from Oakridge National Laboratory (ORNL) for his initial contribution to this work when he was a Ph.D. student at The Ohio State University. We are also grateful to Dr. Spencer Shepler and William Taylor from Sun MicroSystems for their helpful discussions and suggestions.

References

[1] 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*, pages 196–208. ACM Press, 2003.

[2] 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.

[3] InfiniBand Trade Association. InfiniBand^{TM}Architecture Specification, Release 1.2, September 2004. http://www.infinibandta.org/specs.

[4] M. Ko, M. Chadalapaka, U. Elzur, H. Shah, P. Thaler, and J. Hufferd. iSCSI Extensions for RDMA Specification. http://www.ietf.org/internet-drafts/draft-ietf-ips-iser-05.txt.

[5] K. Magoutis, S. Addetia, A. Fedorova, and M. I. Seltzer. Making the Most out of Direct-Access Network Attached Storage. In *Proceedings of Second USENIX Conference on File and Storage Technologies (FAST '03)*, San Francisco, CA, March 2003.

[6] Network Based Computing Lab, Dept. of Computer Science and Engg., The Ohio State University. OpenSolaris NFS over RDMA Project. http://nowlab.cse.ohio-state.edu/projects/nfsrdma/index.html.

[7] P. Wyckoff and J. Wu. Memory Registration Caching Correctness. In *CCGrid*, May 2005.

[8] R. Noronha and L. Chai and T. Talpey and D.K. Panda. Designing NFS With RDMA for Security, Performance and Scalability. Technical Report OSU-CISRC-6/07-TR47, The Ohio State University, 2007.

[9] Sun Microsystems. OpenSolaris Project: NFS RDMA transport update and performance analysis. http://www.opensolaris.org/os/project/nfsrdma/.

[10] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. Pin-down cache: A virtual memory management technique for zero-copy communication. pages 308–315.