

Enhancing the performance of NFSv4 with RDMA *

Ranjit Noronha¹ Lei Chai¹ Spencer Shepler² Dhabaleswar K. Panda¹

¹ The Ohio State University
{noronha, chail, panda}@cse.ohio-state.edu

² Sun Microsystems
{Spencer.Shepler}@sun.com

Abstract

NFS is a widely deployed storage technology. It has gone through several revisions. One of the latest revisions, v4 has started to become deployed. NFSv4 on OpenSolaris uses TCP as the underlying transport. This has limited its performance. In this paper, we take on the challenge of designing an RDMA transport for NFSv4. Challenges include COMPOUND procedures, which might potentially have unbounded request and reply sizes. Performance evaluation shows that NFSv4 can achieve an IOzone Read throughput of over 700 MB/s and an IOzone Write bandwidth of over 500 MB/s. It also significantly outperforms NFSv4/TCP.

Keywords: *InfiniBand, NFS, Clusters*

1 Introduction

Network File System is widely deployed for storage access. The NFS standard itself has gone through several revisions since its initial inception. These revisions have addressed some of the limitations in the original protocol; such as performance. For example, NFSv4 added file based delegations to the protocol. Delegations under certain circumstances allow the client to own a particular file and maintain it locally, instead of going to the server for all accesses. This allows complete local access to files under specific circumstances and also reduces network traffic.

*This research is supported in part by funding internships and equipment donations from SUN Microsystems.

Modern networking technologies like InfiniBand offer improved performance to the user. For example, a Single Data Rate (SDR) NIC provides a small message latency of under $4\mu\text{s}$ and a bi-directional bandwidth of up to 2GB/s. Also, it offers mechanisms like Remote Direct Memory Access (RDMA). RDMA allows a user application to bypass the kernel and directly read or write from the memory space of a remote application. This allows us to design unique protocols for file systems, which can deliver higher performance than a traditional TCP transport.

Previous work has demonstrated the advantages of RDMA over InfiniBand to NFSv3 [5]. While NFSv3 has been widely deployed, NFSv4 has been gaining traction. Since, NFSv4 offers several advantages over NFSv3, it becomes crucial to design an RDMA transport for NFSv4. In this paper, we take on the challenge of designing an RDMA transport for NFSv4 on OpenSolaris.

NFSv4 introduced the concept of a single procedure COMPOUND, which can encompass several operations. This potentially reduces the number of trips to the server. However, it also potentially increases the size of each request sent to the server. A COMPOUND request may potentially have an unbounded length. Correspondingly, the response may also be unbounded. This raises a challenge in a design with RDMA, which requires the length to be precisely specified. Also, COMPOUND procedures may have interactions with other encompassed operations. We demonstrate a design of NFSv4 over an RDMA transport on In-

finiBand, that can achieve an IOzone Read Bandwidth of over 700 MB/s and an IOzone Write Bandwidth of over 500 MB/s. It attains performance close to that of the corresponding design of NFSv3 over RDMA and significantly outperforms NFSv4 over TCP.

The rest of the paper is structured as follows. Section 2 introduces InfiniBand, NFS and NFS over RDMA. In Sections 3, we describe our design. Following that, section 4 provides an evaluation of the design. Related work is described in section 5. Finally, we discuss conclusions and future work in section 6.

2 Background

In this section, we discuss InfiniBand, NFS and NFS/RDMA.

2.1 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 [4].

2.2. Introduction to NFS

The Network File System version 4 (NFSv4) [1], the successor to version 3 de-

rives substantially from the architecture of the previous protocols. The main difference in the protocol is the introduction of state in the protocol. In addition, the other main change is the introduction of the COMPOUND operation. In fact, there are only two RPC procedures in NFSv4; COMPOUND and NULL. The COMPOUND procedure is composed of a sequence of operations. The set of operations is processed sequentially at the server, and the responses are returned sequentially, in a single reply. This helps simplify the design of COMPOUND operations.

2.3. 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) [3]. A design of NFS/RDMA for NFSv3

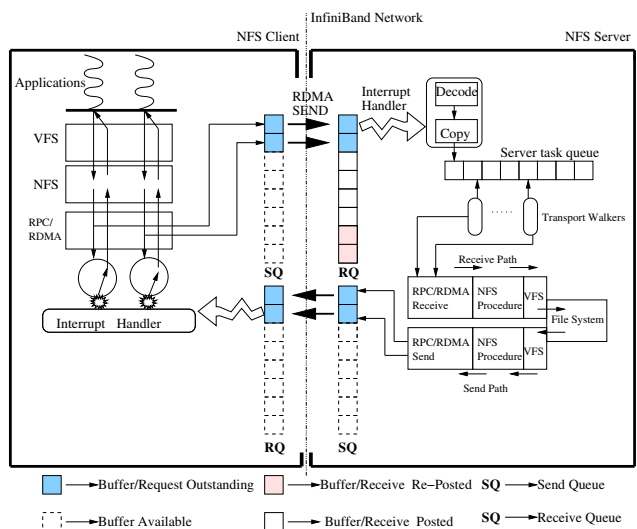


Fig. 1. Architecture of the NFS/RDMA stack in OpenSolaris

is discussed in [5]. This architecture is 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. Bulk data for procedures like

READ, WRITE, READLINK and REaddir use chunks. Chunk list can be *Read chunks*, *Write chunks* and *Reply chunks* and are discussed further in the next Section. Figure 3 shows the protocol exchange in the RDMA design. In the rest of the paper, we use the terms RPC/RDMA and NFS/RDMA interchangeably.

2.4. RDMA Protocol for bulk data transfer

NFS procedures such as READ, WRITE, READLINK, REaddir and COMPOUND can transfer data whose length is larger than the inline data threshold [3]. Also, the RPC call itself can be larger than the inline data threshold. The bulk data can be transferred in multiple ways. We use a combination of RDMA Read and Write operations discussed in [5]. To transfer bulk data, we use chunk lists described next.

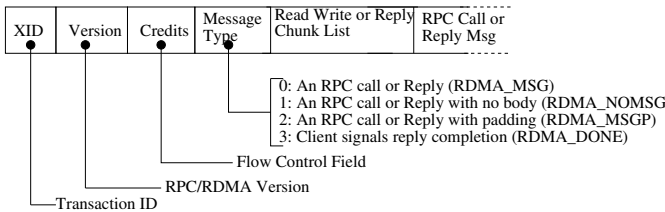


Fig. 2. RPC/RDMA header

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* to encode data that may be RDMA Read from the remote peer. *Write chunks* are used to RDMA Write data to the remote peer. *Reply chunks* 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.

3 Design Overview

In this section, we discuss our design for NFSv4 over RDMA. We first discuss COMPOUND procedures, followed by Read and Write operations and finally, Readdir and Readlink operations.

3.1 Compound Procedures

We first give an example of a COMPOUND procedure, followed by our design.

Example of a COMPOUND procedure: One of the benefits of a COMPOUND operation is that it allows the concatenation of several operations into a single RPC call. On the downside, a COMPOUND operation makes simple operation larger than necessary. Another disadvantage is the error processing involved in a COMPOUND operations. An error in an operation in a COMPOUND will result in the server rejecting and not processing all the remaining operations. As a result, the client may have to send multiple requests to the server to resolve the error. The error processing in a COMPOUND operation is discussed in detail in the IETF RFC 3530 [6]. An example where COMPOUND operation may benefit is the initial opening of a file. The opening of a file requires the procedures LOOKUP (to convert the file name to a file handle), followed by OPEN (to store the file handle at the server) and finally, READ (to read the initial data from the file). Note that OPEN is a NFSv4 specific operation which stores state at the server not present in version 3 and earlier of the protocol. So, in version 3, the sequence we are trying to optimize would be LOOKUP followed by READ. If the COMPOUND is success-

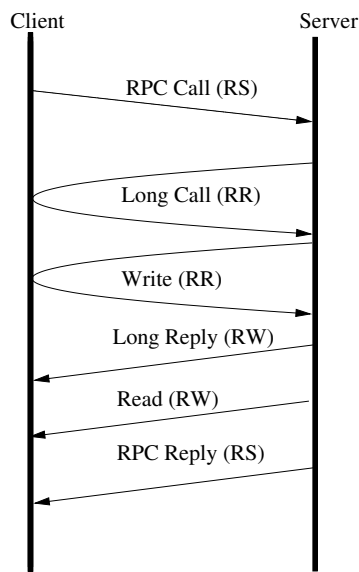


Fig. 3. RPC/RDMA protocol design

ful, three round-trip operations would have been achieved with a single round-trip to and from the server. However, if the lookup failed, bandwidth would have been wasted for sending the OPEN and READ operations. Whether or not this might have an impact depends on the properties of the underlying interconnect.

Design of a COMPOUND operation with RDMA: As discussed earlier, a COMPOUND procedure consists of a sequence of operations. So, the first issue is to transfer a COMPOUND operation from the client to the server. If the sequence of operations constitutes a small message (less than the inline threshold of 1KB), the request may go inline from the client to the server as a RDMA Send operation. If the sequence of operations constitutes a larger message, an RPC long call needs to be generated from the client to the server. In this case, the client will encode the data from the COMPOUND operation in a single RDMA chunk, that is registered. It will then do an RDMA send from the client to the server with the RDMA chunk encoded and the RDMA_NOMSG flag set. For a given implementation of NFSv4, the client may be aware of the potential length of the reply for all procedures. But, in the gen-

eral case, the transport may be used for a minor revision to NFSv4 (currently this revision is NFSv4.1). There might be additional procedures introduced in a minor version, that could generate a long reply. To make the design more general, the client always encodes a long reply chunk for every COMPOUND procedure. To reduce the client memory usage, the client should only send a long reply for sequences of procedures that may generate a long reply. We will consider this in a followup work.

The client should also take into account the preceding REaddir's and READLINK's when computing the length of the long reply. Additional chunks may be encoded to compensate for the preceding REaddir and READLINK procedures. When the server receives the RPC Call, it will decode and store the READ, WRITE and Reply chunks. Finally, control will be passed to the NFS layer for further processing. On return from the server, the RPC/RDMA will need to return the results of the COMPOUND procedure to the client. If the results are small, the results can be returned inline. Otherwise, the last reply chunk sent from the client is used to return the data to the client. So, the reply chunk sent from the server to the client may remain unused. From this, it may seem that the client may need to waste memory for COMPOUND operations. Also, these memory regions would need to be registered. But, with appropriate design strategies in the design, such as a server registration cache, or physical registration [5], the impact of resource allocation and/or memory registration is likely to be minimal.

3.2 Read and Write Operations:

Read operations would follow the design of the READ procedure in previous work [5], with one caveat. Write chunk lists now must account for the sequential ordering of Read operations in a COMPOUND procedure. The RPC layers at the client and the server need to take this into account, by appropriately marshaling the chunks at the client, storing them at the server and then us-

ing them to RDMA Write data back to the client. Similarly, Write Operations follow the design of the WRITE procedure in NFSv3. Sequential ordering of operations is taken into account.

3.3 Readdir/Readlink Operations

Readdir and Readlink operations always encode a long reply chunk. This chunk needs to take into account the sequential ordering of operations in a COMPOUND procedure. The encapsulating COMPOUND procedure itself will include a long reply chunk as the last chunk in the list. The design should account for this interaction.

4 Experimental Evaluation

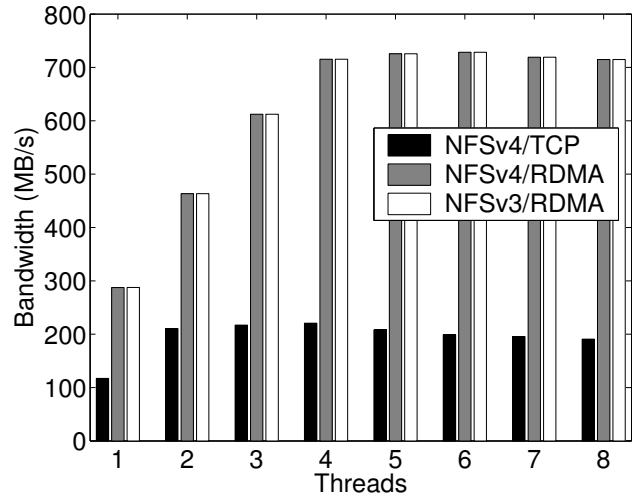
In this section, we discuss the experimental evaluation of NFSv4 with an RDMA transport. First, we describe the setup in Section 4.1. Following that we compare the performance of NFSv4/RDMA with NFSv3/RDMA in section 4.2. Finally, we compare NFSv4/RDMA with NFSv4/TCP in section 4.3.

4.1 Experimental Setup

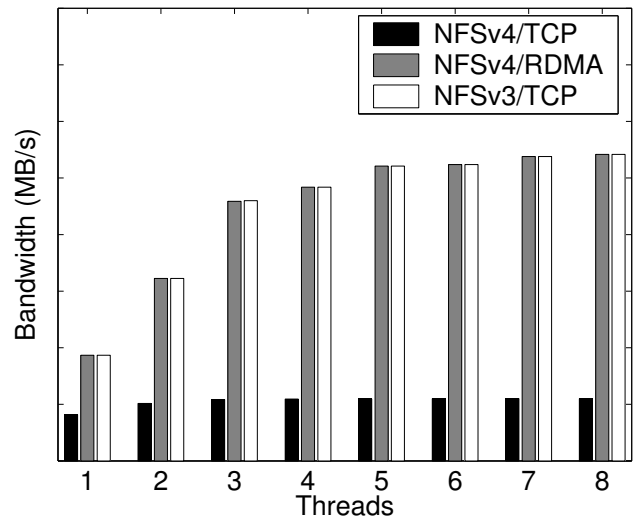
We measure the IOzone [2] Read and Write bandwidth with direct I/O on OpenSolaris. The results were taken on dual Opteron x2100's with 2GB memory and Single Data Rate (SDR) x8 PCI-Express InfiniBand Adapters. These systems were running OpenSolaris build version 33. The back-end file system used was tmpfs 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 set to be 128KB.

4.2 Impact of RDMA on NFSv4

Figures 4(a) and 4(b) shows the IOzone Read and Write bandwidth for NFSv4 and NFSv3 over an RDMA transport. From the figures, we can see that v4 performs comparably with v3. The



(a) Read



(b) Write

Fig. 4. IOzone Bandwidth Comparison

Read bandwidth saturates at 714 MB/s for both NFSv4 and NFSv3. Similarly, Write Bandwidth saturates at 541.71 MB/s. It should be noted that NFSv4 performs slightly worse than NFSv3. This is to be expected because of the additional overhead of COMPOUND operation, as discussed in Section 3.1.

4.3 Comparison between NFSv4/TCP and NFSv4/RDMA

We also compare the performance of NFSv4 when the underlying transport is TCP. This is shown in Figures 4(a) and 4(b) (IOzone Read and Write Bandwidth, respectively). For TCP, we used IPoIB (IP over InfiniBand) as the transport over the same InfiniBand link. Clearly, NFSv4/RDMA outperforms NFSv3/RDMA by an order of magnitude. While, NFSv4/RDMA saturates at over 700 MB/s for IOzone Reads, NFSv4/TCP saturates at just over 200 MB/s. Similarly, for Writes, NFSv4/RDMA saturates at 541.71 MB/s, while NFSv3/TCP saturates at slightly under 100 MB/s.

5 Related Work

The Linux NFS over RDMA design and the Netapp filer allow for NFSv4 operations [7]. Our design will be used on OpenSolaris as the default main transport for NFSv4.

6 Conclusions and Future Work

In this paper, we have designed and evaluated an RDMA transport for NFSv4. The design challenges included allowing COMPOUND operations which can potentially have an unbounded number of v3 operations to use RDMA operations, which are length limited. Evaluation shows that NFSv4 with RDMA has performance similar to NFSv3 for an RDMA transport. IOzone Read bandwidth saturated at over 700 MB/s, while IOzone Write bandwidth saturates at over 500 MB/s. NFSv4 over RDMA outperforms NFSv4 over TCP (IPoIB) by an order of magnitude.

As part of future work, we will explore extending the design of RPC/RDMA for parallel NFS (pNFS). This is expected to address some of the limitations in using TCP in parallel file systems, such as copying overhead from multiple TCP connections, as well as ACK implosion, which may severely hurt aggregate performance with multiple data servers.

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] B. Callaghan. NFS Illustrated. In *Addison-Wesley Professional Computing Series*, 1999.
- [4] InfiniBand Trade Association. InfiniBand™ Architecture Specification, Release 1.2, September 2004. <http://www.infinibandta.org/specs>.
- [5] Ranjit Noronha, Lei Chai, Thomas Talpey, and Dhabaleswar K. Panda. Designing NFS with RDMA for Security, Performance and Scalability. In *The 2007 International Conference on Parallel Processing (ICPP-07)*, Sept 2007.
- [6] S. Shepler, B. Callaghan, D. Robinson, R. thurLOW, C. Beame, M. Eisler, and D. Noveck. NFS version 4 Protocol. <http://www.ietf.org/rfc/rfc3530.txt>.
- [7] T. Talpey et. al. NFS/RDMA ONC Transport. <http://sourceforge.net/projects/nfs-rdma>.