Optimizing OpenSolaris NFS over RDMA

Weikuan Yu   Ranjit Noronha   Lei Chai   Shuang Liang   Dhabaleswar K. Panda

Technical Report
OSU-CISRC-4/06-TR43

Optimizing OpenSolaris NFS over RDMA [*]

Weikuan Yu   Ranjit Noronha   Lei Chai   Shuang Liang   Dhabaleswar K. Panda

Network-Based Computing Lab
Department of Computer Science & Engineering
The Ohio State University
{*yuw,noronha,chail,liangs,panda*}*@cse.ohio-state.edu*

Abstract

Network File System (NFS) is widely deployed as one of the reliable means for file sharing. A trend in NFS development is the use of Remote Direct Memory Access (RDMA) as the data transport protocol. With its capability of offloaded data movement and direct data placement, RDMA is able to reduce the host CPU involvement in bulky data movement, as well as the load on the memory and IO buses. NFS over RDMA has been first attempted on Solaris with a pure RDMA read-based design and available in the recent releases of OpenSolaris. In this paper, we further investigate and optimize the design of OpenSolaris NFS over RDMA. We focus on the appropriate, yet efficient use of RDMA read and RDMA write for different NFS operations. We also devise mechanisms for credit-based flow control for efficient and scalable communication of NFS over RDMA. To be interoperable with corresponding client/server implementations from other NFS over RDMA implementations, the client and server protocols are both designed to be fully compliant to the IETF drafts for NFS Direct Data Placement [11] and RPC over RDMA [12]. We have also evaluated our design of OpenSolaris NFS over RDMA protocol. Our results indicate that the optimized NFS over RDMA design can improve NFS read performance by up to 28% in single-threaded test cases, and up to 44% in multi-threaded cases. In addition, our design is also beneficial to the performance of some NFS metadata operations, such as readdir, by up to 15%.

Keywords: *NFS, OpenSolaris, RPC, RDMA, InfiniBand*

1. Introduction

Since its inception in the mid-80's, the Network File System (NFS) [28] protocol has been ubiquitously accepted as a standard means for sharing files in many operating systems. It has been deployed on a variety of architectures and platforms, including workstations and high end server farms. Despite the rapid development of different cluster file systems [35, 13, 5], incarnations of NFS continue to be used as the primary file system for maintaining user home directories largely because of its reliability and ease of deployment.

NFS has gone through several generations of development, including versions 2, 3 and 4 [28, 24, 29]. Evolutionary advances in NFS include better performance with significant feature additions like TCP-based transport, large block transfer, server retry caches and asynchronous writes. Recent technology advances, such as high speed, multi-core CPU architectures, high bandwidth system memory and IO buses, as well as high speed networks, continuously push the performance bottlenecks to the slowest evolving components. This exerts a Darwinian pressure on file system protocols like

[*]This research is supported in part by a DOE grant #DE-FC02-01ER25506 and NSF Grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco Systems and Sun MicroSystems; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, Microway, PathScale, IBM, SilverStorm and Sun MicroSystems.

NFS, which also need to keep abreast with these technological changes to remain competitive. In particular, state-of-the-art networking technologies, such as InfiniBand [18] and 10GigE [17, 15], provide high bandwidth of more than 10Gbps and submicrosecond latency. This trend of network acceleration leads to the various new communication protocols being designed, such as RDMA [26] and SDP [4]. These protocols typically try to offload the movement of data into the network hardware and directly transfer data between source and destination memory buffers. These appealing transport protocols and the physical capacity of their underlying networks have triggered the improvement of other subsystems in the computer architecture. In particular, networking has become an integral part of file system and storage subsystems in so called networked storage systems. These networked subsystems have integrated new generations of network communication interfaces either through the creation of new protocols such as DAFS [14], or by the introduction of a new data movement layer to an existing protocol, such as iSER [19] for SCSI.

RDMA, with its offloaded processing and direct data placement features, can reduce the overhead for both data and metadata exchange. This communication benefits are directly linked with critical NFS performance metrics in terms of IO bandwidth and transaction throughput. Hence, RDMA has been viewed as an opportunity for improving the performance of all NFS versions. It also allows NFS to gain benefits from the latest networking technologies such as InfiniBand [18] and iWARP [27] via 10GigE [17, 15]. Multiple IETF drafts [11, 12] have been proposed to publicize the problem statements and standardize the wire protocol for possible NFS over RDMA implementations.

There have been earlier studies to build NFS data movement on top of RDMA transport protocol. For example, Callaghan et. al. [9] have prototyped an RDMA read based transport protocol for NFSv3 and have demonstrated its benefits in terms of NFS READ bandwidth and CPU utilization. Talpey et. al. [33] have recently announced the availability of initial implementations for Linux NFS client and server. While the NFSv4 [29] being positioned as a cluster file system with its pNFS [8] parallel extension, an efficient NFS over RDMA implementation would be an indispensable component for NFS to demonstrate its capabilities of blending with the latest networking technology and maximizing the utilization of available network IO capacity. Such performance promises of RDMA will help the continuous adoption of NFS into more high end server and computing environments, as they do not have to switch to proprietary solutions for equivalent performance. Thus it is desirable to ensure that the existing implementations can deliver its optimal performance.

In this paper, we delve into the issues of designing efficient NFS over RDMA for OpenSolaris, while taking into account the advocated IETF RPC RDMA draft [12] requirements of NFS scalability, security and interoperability. For these purposes, we demonstrate the inappropriateness of the current RDMA read-based design of OpenSolaris NFS over RDMA. For example, such RDMA read based design exposes part of the server memory space to any client. This likely put the server at risk from misbehaving or even malicious clients. An RDMA read based design may significantly limits the number of concurrent RDMA operations. Over InfiniBand, this number is as small as eight (for current generations of InfiniBand HCAs) compared to several thousands for RDMA write. Instead, we provide an optimized design, which also conforms to NFS direct data placement and RPC over RDMA drafts [11, 12]. In our design, only the NFS server is allowed to initiate RDMA operations, including RDMA read and RDMA write as appropriate. The conformance of our design to these publicized IETF standards guarantees its interoperability with the corresponding client/server implementations from other NFS over RDMA implementations such as Linux NFS over RDMA [33]. We have also devised mechanisms for credit-based flow control for efficient and scalable communication of NFS over RDMA.

We have evaluated the new design of OpenSolaris NFS over RDMA protocol. IOzone benchmark tests indicate that, as expected, the current implementation does not affect the performance of NFS write, whose communication remains to go through RDMA read. On the other hand, for NFS read, our results show that RDMA write based RPC brings up to 28% performance improvement in single-threaded test cases, and up to 44% in multi-threaded cases. Moreover, our design is also shown to be beneficial to the performance of some NFS metadata operations, such as readdir, by up to 15%.

The rest of the paper is presented as follows. In Section 2, we discuss some related work. Section 3 provides an overview of InfiniBand, NFS and its related protocols including XDR and RPC. In Section 4, we describe the detail design of NFS over RDMA, focusing on the utilization of RDMA write in the new RPC/RDMA transport and its conformance to the IETF standards [11, 12]. Following that, in Section 5, we describe some resource management issues related to NFS over RDMA, including the server-side credit control and connection management. In Section 6, we provide the performance evaluation of the new design. Finally, Section 7 concludes the paper.

2. Related Work

There have been numerous studies in the optimization of NFS protocols. In this section, we discuss about some of the most related work in the evolution of NFS, as well as its relation to the advances in high performance networks and other network-based storage protocols.

Xu et. al. [34] investigated the performance benefits of client caching to concurrent read sharing over NFS. Peng et. al. [25] showed that a network-centric reorganization of the buffer cache can improve the NFS performance. Radkov et. al. [23] compared the performance of file-based NFS protocol and block-based iSCSI protocol and noted that aggressive meta-data caching can benefit the NFS protocol.

Martin et. al. [21] studied the sensitivity of NFS to high performance networks by introducing controlled delays into live systems in the late 90's. They observed that NFS was more sensitive to processor overhead rather than networking latency and bandwidth. However, the emergence of high speed networks with direct access protocols such as RDMA lead to both the design of new network file system and the revision of traditional network file systems to enable file accesses over RDMA-capable networks. For example, iSER was recently proposed and standardized by IETF as an extension for Internet Small Computer Systems Interface (iSCSI) protocol [19, 32]. DAFS [14, 20] designed a user space file system library that allows applications to bypass operating system kernel and take advantage of high performance user-level network directly. Goglin et. al. [16] replaced the RPC protocol of NFS with Myrinet GM protocol to achieve Optimized Remote File System Accesses (ORFA). Callaghan et. al. [9] provided an initial implementation NFS over RDMA on Solaris. An RDMA read based RPC transport is implemented as a proof of concept to show the performance benefit of NFS over RDMA compared to TCP.

Our work continues this endeavor further and optimizes the design of NFS over RDMA for the purposes of its performance enhancement and compliance to IETF drafts [11, 12] for wide interoperability. Our studies are performed with experiments over 10Gbps InfiniBand [18]. Note that, Talpey et. al. [33] have recently announced the availability of initial Linux NFS over RDMA implementations for comments. This project is carried out in parallel with our work in order to verify the mutual interoperability.

3. Overview

In this section, we discuss InfiniBand RDMA operations, the OpenSolaris implementation of InfiniBand and the NFS protocol.

3.1 Overview of RDMA operations in InfiniBand

InfiniBand [18] supports channel based operations for reliable communication. These operations include traditional send/receive semantics as well as Remote Direct Memory Access (RDMA) primitives. Send/Receive (*RDMA Send*) primitives require the prior posting of a descriptor on the receiver side. RDMA operations allow secure access to the memory of a remote node without involvement of that node. RDMA operations are mainly of two types Read and Write. *RDMA Read* allows a given node to directly read the contents of a remote node. Similarly, *RDMA Write* allows one to deposit data directly into the memory of the remote node. Fig. 1 shows the InfiniBand software stack in OpenSolaris. The Tavor HCA driver provides access to the InfiniBand to the Solaris InfiniBand Transport Framework (IBTF) which implements the InfiniBand Transport Layer (IBTL) and IBMF. Other access protocols such as IETF IP over InfiniBand (IPoIB) and user defined application programming layer (uDAPL) are implemented using IBTF. NFS/RDMA may access InfiniBand through IBTF.

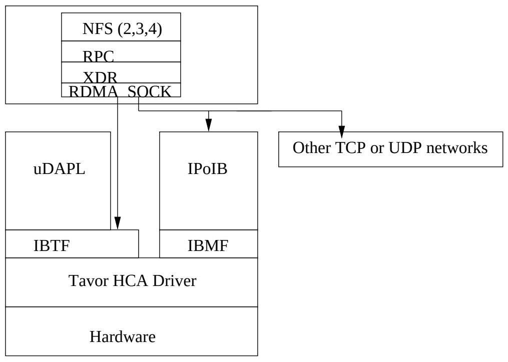

Fig. 1. OpenSolaris Implementation of InfiniBand and NFS

3.2. Overview of NFS

Network File System (NFS) [7] is ubiquitously used in most modern clusters. It allows users to transparently share file and IO services on a variety of different platforms. Figure 1 shows a diagram of the NFS architecture on OpenSolaris. NFS is based on the single server, multiple clients model. Communication between the NFS client and the server is via the Open Network Computing (ONC) remote procedure call (RPC). RPC is an extension to the local procedure calling semantics, and allows programs to make calls to nodes on remote nodes as if it were a local procedure call. RPC traditionally uses TCP or UDP as the underlying communication transport. RDMA transport enabled RPC specifications and protocols have become available [12]. RPC is a standard defined by IETF RFC 1831 [30]. Since the RPC calls may need to propagate between machines in a heterogeneous environment, the RPC stream is usually serialized with the eXternal Data Representation (XDR) standard (IETF RFC 1832 [31]) for encoding and decoding data streams. NFS has seen three major generations of development. The first generation, NFS version 2 (RFC 1094 [3]), provided a stateless file access protocol between the server and client using RPC over UDP. NFS version 3 [24] (RFC 1813 [10]) added to the features of NFSv2 and provided several performance enhancements, including larger block data transfer, TCP-based transport and asynchronous write, among many others. The latest version NFS version 4 [1] specification was developed by IETF (RFC 3530 [29]) and includes features for improved access and performance.

4. Designing OpenSolaris NFS over RDMA

In this section, we describe the new design of OpenSolaris NFS over RDMA. We focus on these issues: RPC message layout for NFS, leveraging RDMA write in transport, and XDR stream encoding/decoding of RDMA chunks. In particular, we describe the importance of leveraging RDMA write in the transport of direct (RDMA chunk) messages as well as long RPC messages. In addition, as we describe these issues, we illustrate in what aspects the design can improve the current NFS over RDMA implementation.

4.1. RPC Message Layout

In this section we describe the formats of RPC calls and replies. RPC call messages are composed of two parts - an RDMA chunk which includes an RDMA header and zero or more read/write chunk lists, and an RPC message which includes an RPC header and non-chunked arguments. The layout is shown in Fig. 2. With NFSv3, at most one of *read chunk list*, *write chunk list*, and *long reply chunk* is not null. When a chunk list is null, there is a single *False* at the corresponding position.

The layout of the RPC reply is similar to that of the RPC call. The *write chunk list* and *long reply chunk* should be the same as received from the client, but with updated *length* information. In our RDMA write based implementation, since the client shall never issue RDMA operations to the server, the *read chunk list* is always null.

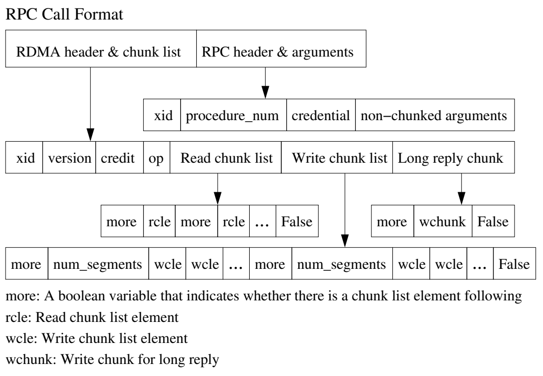

Fig. 2. Layout of RPC Call/Reply Messages

4.2. Leveraging RDMA Write in transport

The original RDMA read based design has several drawbacks. These include the security and performance concerns because the original design allows RDMA read from the client to access memory regions on the server. But for security reasons, an NFS server shall avoid exposing its memory segments to any client to prevent incorrect or malicious uses of its memory resources. In addition, RDMA write on some interconnects, e.g. InfiniBand, allows more messages concurrently in transit. This can help the new design to achieve better data throughput. Moreover, the original RDMA read based design also requires the client to send an RDMA Done message to the server when it finishes reading the Reply Chunks. Thus RDMA write based design can eliminate this extra control message for the completion of an RPC transaction, potentially enhancing performance. The details are described in Section 4.2.2. Finally, we design a DIRECT_IO path in the RDMA write based design for NFSv3 READ operations. This allows the server to directly write into the client application buffer, reducing client overhead.

4.2.1. Transport of Short and Direct (RDMA chunk) Messages

Depending on the type of NFS operations, RPC messages may vary in length. For example, sizes for RPC call and reply for NFSv3 GETATTR are usually less than 120 bytes; RPC messages for NFS read/write can be as long as the maximum fragment size allowed. Based on the fact that the communication of RDMA chunks requires the communication of extra steering tags and message offset/lengths, it makes sense to put short message inline with the RPC messages, while exploiting the use of direct (RDMA chunk) transfer when the message size is large than a threshold, RDMA_MINCHUNK (adjustable value, 1KB is the default).

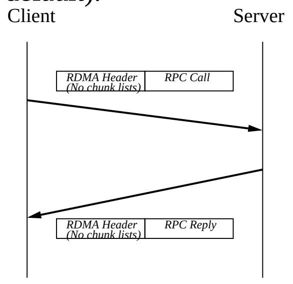

Fig. 3. Inline Transport of Short Messages

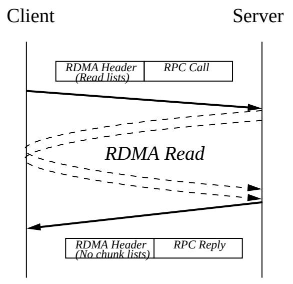

Fig. 4. RPC via RDMA Read

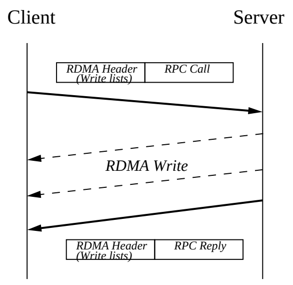

Fig. 5. RPC via RDMA Write

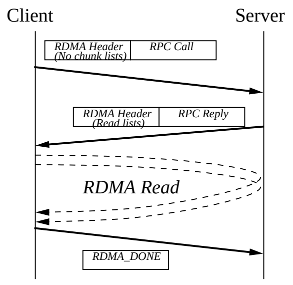

Fig. 6. NFS Read via RDMA Read

Figures 3, 4 and 5 illustrate the design of Inline, RDMA read and RDMA write based RPC transport, respectively. For NFSv3, no RPC message will include both Read and Write chunklist, so Figures 4 and 5 exactly correspond to the communication for large NFS write and read operations. NFSv4, in order to save the number of network messages, introduces compound operations. Such compound operations lead to the combination of multiple read/write operations in one RPC call. For these compound operations, the server side can satisfy the requirement by combining the processing of both read and write chunk lists and invoking RDMA read and write as needed. It should be noted that in the current OpenSolaris NFS over RDMA, NFS READ operations are built entirely on top of RDMA read. Fig. 6 shows the communication flow diagram of NFSv3 Read operations with RDMA read. In contrast, the RDMA write based design for NFSv3 READ readily saves an extra RDMA_DONE message. More important, the throughput could potentially be much higher as the server can issue several concurrent RDMA write messages, compared with what it can do with RDMA read.

4.2.2. RDMA-based Long Message Transport

NFS can have RPC procedures that transfer very large amount of data in either RPC call or reply, whose sizes exceed the maximum allowable size for an RDMA send operation. These situations can be more frequent with NFSv4 compound operations as it combines more operations into a single RPC call. These chunks can be in the size of megabytes. RDMA send operations require pre-pinned memory buffers. Forcing the data to be sent inline in RDMA send operations can lead to unacceptably large pre-pinned send and receive buffers. This might impact scalability of the server memory resource management. Additionally, the expensive data copies needed to process the received data might tie down the CPU, impacting utilization and the ability to serve more clients. It makes sense to leverage RDMA transport for transferring these chunks of data because it can be more efficiently carried out with RDMA's direct placement and less involvement of host CPU.

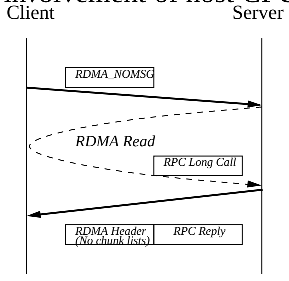

Fig. 7. Long RPC Call via RDMA Read

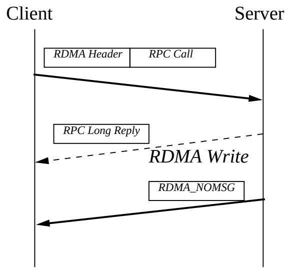

Fig. 8. Long RPC Reply via RDMA Write

The long RPC call and reply can happen in direct mode of RPC message transport or inline mode of transport, because the number of fragments can be very large while each of the fragments is less than RDMA_MINCHUNK. Again, NFSv4 compound operations can lead to long chunk lists and, hence, long RPC call and reply messages. Figures 7 and 8 show the diagrams of RDMA read and write based transport for such long RPC call and reply messages. In contrast, as shown in Fig. 9 with the original RDMA read based RPC transport, a long RPC reply message has to be pulled by the client from the server, followed by an RDMA_DONE message sent to the server. As a result, such RDMA write based RPC transport can benefit both the NFS server throughput and the response time as seen from the client.

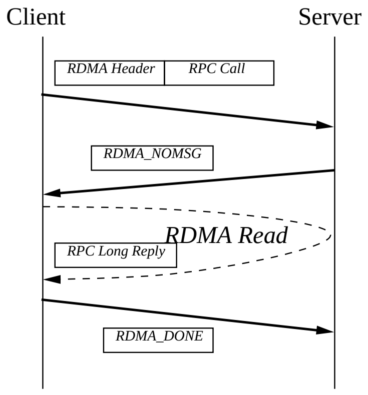

Fig. 9. Original Long RPC Reply via RDMA Read

4.3. XDR stream encoding and decoding of RDMA chunks

Through XDR encoding and decoding, application data can be converted into a form that is suitable for transmission from one computer to another in a network order and architecture endian independent manner. For references of XDR, please refer to IETF XDR RFC 1832 [31]. There are different type of XDR streams. For NFS over RDMA to work, a new type, XDR_RDMA, is introduced for communicating data via RDMA. As described in Section 4.1, each RPC message is encapsulated into XDR_RDMA stream in which an RDMA header in wrapped ahead of RPC header. Within RDMA header, the chunk lists are provided. We describe the processing of forming read and write chunk lists,

4.3.1. Creating RDMA Chunk Lists

The new design of NFS over RDMA in OpenSolaris employs three different types of RDMA chunk lists including, read chunk lists, write chunk lists and long RPC reply chunk lists. The original design has only RDMA read chunk lists. RDMA read and write chunk lists are created based on the buffers for the RPC arguments and results, respectively. The length for each chunk are available based on the length of memory IO vectors. If the RPC operation is expecting a long reply (or a reply message of unknown length) from the server, a long RPC reply chunk of LONG_RPC_LEN (64KB) is allocated and created as a reply chunk list of only one chunk. For NFSv3, there is exactly only one type of chunk lists amongst these three. With NFSv4, there can be any combination of these three. If the RPC message is large than the buffer size for RDMA send operations, this results in a long RPC call. A new buffer is allocated from a slab cache and created as an extra read chunk put at the zero offset, i.e. prepended to the existing read chunk list. The resulting RDMA RPC message is marked as RDMA_NOMSG, indicating the server that an extra RDMA read is needed to get the actual RPC message and the rest of read chunk lists describes the RDMA read chunks for the actual read arguments.

4.3.2. Encoding and Decoding of RDMA Chunk Lists

Once the chunk lists are created, they will be marshaled into the RDMA header in the order of read chunk lists, write chunk lists and long reply chunk list, as described in Section 4.1. The absence of any list will be marked with a FALSE boolean value. In addition, for a regular RPC call message, the RDMA header will be typed as RDMA_MSG (or RDMA_MSGP when padding is introduced for alignment). For a long RPC call, its RDMA header is typed as RDMA_NOMSG, as mentioned in Section 4.3.1. When an NFS RPC call is received, the read chunk list is decoded and the corresponding read chunks are pulled over from the client. The write chunk list or the long reply chunk list (of a single element only for NFSv3) is decoded and stashed for use until the actual data is available.

When the NFS server is ready to return the results for an RPC call request, the originally stashed write chunk list or long reply chunk list is used as destinations for directly placing the data back to the client. At the same time, the actual length of data transferred for each chunk is updated in the chunk list and encoded into the RDMA header in the RPC reply message to the client. This chunk length update is needed because the server actual file content may not the length expected from the client, for example, if the end of file (EOF) is reached before fetching the expected length of data. Or in the case of long reply chunk, the reply chunk is allocated of length, LONG_REPLY_LEN (64KB), by the client. The actual length of the results may either be shorter or longer than LONG_REPLY_LEN. For example, an NFS3 READDIR RPC call may fall on directories of very different sizes, for which a length update is needed to inform the client to issue one or more requests for leftover results.

5. Managing Resources RPC over RDMA

In this section, we discuss several resource management issues for both performance optimization and RPC over RDMA configuration. Specifically, we focus on scalability related flow control and interoperability related connection management issues.

5.1. Flow Control

RPC calls and replies are exchanged through RDMA Send/Receive operations. It is required that receive buffers must be pre-posted on a connection to accept incoming RDMA Sends from that connection [18]. If no receive buffers are available, RDMA Receive operations will fail. It is straightforward to meet this requirement on the client side. Whenever the client sends out an RPC call, it expects an RPC reply from the server. Therefore, the client can pre-post a receive buffer prior to every RPC call. However, it is impossible for the server to predict which clients are going to send RPC calls and the number of requests that they are going to send, thus flow control is a critical issue to make sure that the RDMA connections between the server and the clients work properly.

The Internet Draft ”RDMA Transport for ONC RPC” [12] has identified that it is not scalable to provide fixed credit limits to the clients. With fixed credit limits, buffers may not be used efficiently at the server side, because the posted buffers are dedicated to the associated connection until they are consumed by receive operations. Therefore, the draft specifies a request/grant protocol in the RPC over RDMA header associated with each RPC message. The client can request for a certain amount of credits based on its needs, and the server can grant the client credits based on its resource usage policy. The client should never send out more unacknowledged RPC calls than granted.

In this paper, we designed algorithms for the client and the server to dynamically determine the number of credits they shall request or grant based on their current status. The algorithms for the client and the server are shown in Fig. 10 and 11 respectively.

In the client algorithm (shown in Fig. 10), *inflight* indicates the number of pending RPC calls the client has sent in the recent past, which may predict the trend in the near future. Therefore, *request* should increase as *inflight* increases. On the other hand, the server executes an algorithm as shown in Fig. 11. The main idea for this algorithm is that the server should adjust *grant* based on the total number of clients and total number of receive buffers available. When the amount of resources is sufficiently large, the server shall satisfy the client's request as much as possible.

5.2. Connection Management

Besides providing flow control for buffer credits per client connection, connection management is also an important issue for accomplishing interoperability between different computing environment with different NFS over RDMA implementations. Currently, Solaris InfiniBand stack provides a switch-based Address Translation Service (ATS) for connection queries, while Linux-based InfiniBand stack, e.g. OpenIB Gen2, introduces RDMA CMA based connection management interface, as an enhancement from its original InfiniBand specific Connection Management (IBCM) model. Since RDMA CMA provides a device independent interface for setting up new connections, it is currently being proposed as an IETF draft for connection management interface for all RDMA-capable hardware devices including iWARP [27] and InfiniBand. Thus it is necessary to adapt to the RDMA CMA model from Solaris ATS. Collaborative efforts from Sun MicroSystems, Network Appliance and The Network-based Computing Laboratory at The Ohio State University has been initiated to ensure basic network interoperability and the interoperability between different NFS over RDMA implementations. A final consensus will be materialized on a revised design of OpenSolaris connection management.

6. Performance Evaluation

In this section, we describe the performance evaluation of our NFS over RDMA implementation compared to that from the original OpenSolaris. Benchmarks including IOzone [2], Fileop (a metadata benchmark distributed along with IOzone) and Postmark [6] are used in our experiments. Our experimental testbed consists of four Sun Fire V20z server nodes. All nodes have PCI-X InfiniBand MT23108 HCA's. They are connected to a SilverStorm 5000 switch. The original OpenSolaris build version 33 was used in all experiments. In this evaluation, we focus on the performance of NFSv3 over the RDMA-based RPC transport, comparing the RDMA write capable design to the original design. We plan to devise a set of NFSv4 testing programs, and further investigate the performance benefits of the new RDMA write-capable design to NFSv4 specific features, such as compound operations. Note that in our experiments, unless explicitly specified, a memory-based file system (/tmp in Solaris) is exported from an NFS server for clients to mount over RDMA-based RPC transport. The intent of using a memory-based file system is to avoid the bottleneck that could be imposed by the disk access so that the benefits of high speed network communication can be maximally exposed to network IO in NFS.

```
Input:
min_client:  Minimum number of credits that
             the client should request for
threshold:   Low credit threshold
inflight:    Number of pending RPC calls
granted:     Number of credits granted last time
f_client:    Adjustment factor

Output:
request:     Number of credits the client will request

Algorithm:
if (granted−inflight >= threshold) then
    request = min_client
else
    request = min_client + inflight * f_client
endif
```

Fig. 10. Algorithm for the client to dynamically adjust credit requests

6.1. IOzone Read and Write

IOzone [2] performs a variety of file IO tests on a file system, such as read/re-read, write/re-write. To investigate the performance of a file system under different IO patterns, these tests can be carried with a variety of parameters, options and I/O modes.

In our experiment, we use DIRECT_IO option to bypass file system buffer cache to compare the network impact with different transport protocols. Fig. 12 shows the performance results of NFS read operation for RDMA write and RDMA read design for different file sizes. Our RDMA write design performs consistently about 20% better than RDMA read based design. These results indicate that the RDMA write based RPC transport indeed provides better throughput compared to RDMA read based RPC transport. This is because, compared to RDMA read based RPC, the new design allows more concurrent RDMA write operations to be aggregated together and it has also eliminated an extra control message, RDMA_DONE.

At the same time, as shown in Fig. 13, the NFS write operation still performs the same. This is because, in our new design, the actual network IO for NFS write remains to be RDMA read based, in which the NFS server pulls client data based on the read chunk list inside the RPC call.

From the above experiments, the peak bandwidth for both read and write operations falls far below the physically capability of 10Gbps InfiniBand.

```
Input:
average:     Average number of buffers available
             to each client
posted:      Number of buffers posted  to the client
requested:   Number of credits requested
f_server:    Adjustment factor

Output:
grant:       Number of credits the server will grant

Algorithm:
if (requested <= posted) then
    grant = posted
else if (requested <= average) then
    grant = requested
else
    grant = average + (requested − average) * f_server
endif
```

Fig. 11. Algorithm for the server to dynamically adjust credit grants

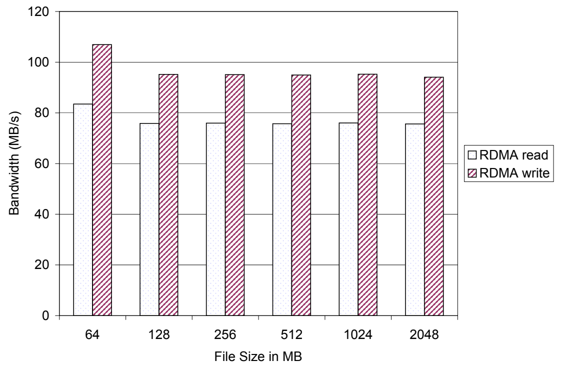

Fig. 12. IOzone Read Performance

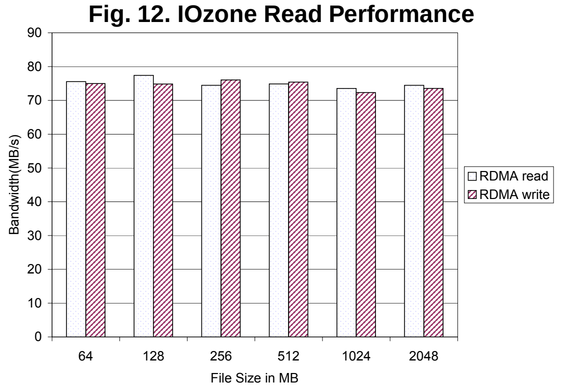

Fig. 13. IOzone Write Performance

We have carried out another set of experiments to find the achievable IO bandwidth with more number of client threads. With the same IOzone benchmark, we let an NFS client initiate NFS read and write requests via varying number of threads. As shown in Fig. 14, the aggregated IO bandwidth is clearly increasing with an increasing number of IO threads from the client. Eventually, the bandwidth number levels off at around 200MB/sec. This experiment indicates that the current design of RPC over RDMA still has a drawback of not being able to request enough RPC transactions to the network. Further efforts need to be put on how to increase the level of concurrent requests through either modifications of the RPC over RDMA interface or the number of readahead window. Also shown in the figure is the performance comparison of NFS read between the RDMA read- or RDMA write-based RPC transport. RDMA write-based design provides consistently better performance, which suggests RDMA write can be beneficial to the throughput of RPC requests.

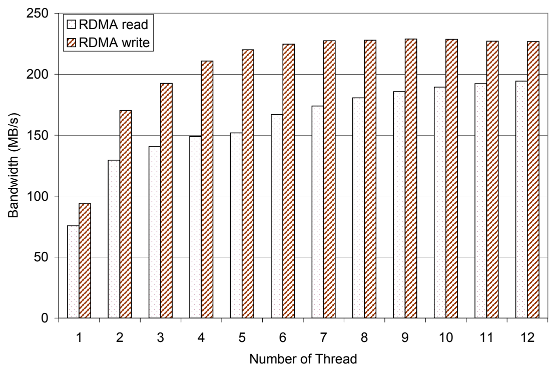

Fig. 14. IOzone Multi-threading Read Performance

6.2. Fileop

Another benchmark, fileop, is distributed along with IOzone. It tests the performance of a variety of meta-data operations including mkdir, create, close, stat, chmod, readdir, link, unlink. As shown in Fig. 15, the performance of such NFS metadata operations largely remains the same compared to the original NFS implementation. However, one operation to note is the NFS readdir. An NFS client invokes this operation in order to fetch the content of a directory. Compared to the original design, a long RPC reply chunk is first allocated and the NFS server provides the directory content through an RDMA write operation so that NFS readdir can take advantage of RDMA write as shown in Fig. 8. Fig.15 shows the performance of NFS readdir is improved by about 15% due to this design.

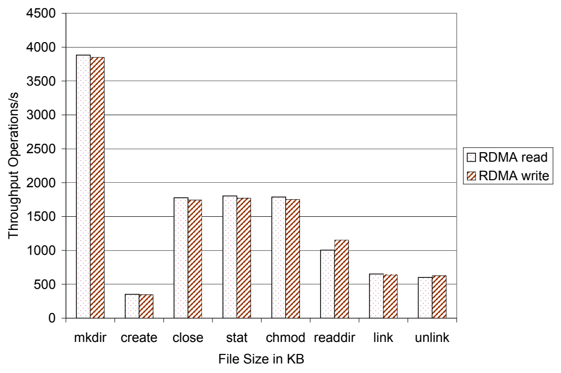

Fig. 15. Fileop Performance

6.3. Postmark

Postmark [6] is a benchmark that measures file system performance on small short-lived files. It first creates a pool of text files and then performs two different sets of transactions on the pool, either create/delete or read/append a file. These types of workloads are typically seen in computing environments such as Internet e-mail and news servers. Table 1 shows the performance comparisons of Postmark between the RDMA write-based design and the original read based design. However, for all different operations in Postmark, the transaction rates appear to be comparable. This is reasonable considering that Postmark transactions are on short-lived, small files, and more importantly, a majority of traffic is NFS write in nature. Therefore, Postmark does not reveal the benefits of our new design.

Table 1. Postmark Performance

Postmark	Transaction/s	Create/s	Read/s	Append/s	Deletes/s
RDMA write	344	180	172	170	180
RDMA read	348	182	174	172	182

6.4. CPU Utilization

One of the most important metrics is the CPU utilization, which reflects how much overhead the NFS operations impose to the host. If the CPU utilization is low, the host can use the CPU cycles to do other useful work thus improves the overall system performance. In this experiment we have measured the CPU utilization for the duration of IOzone. DIRECT_IO is specified in the IOzone tests. The CPU utilization is measured by the *statit* tool [22]. The results are shown in Fig. 16. From this figure we can see that the CPU utilization for the RDMA write based implementation is as low as 4%, which is 2 to 3 times better than the RDMA read based design.

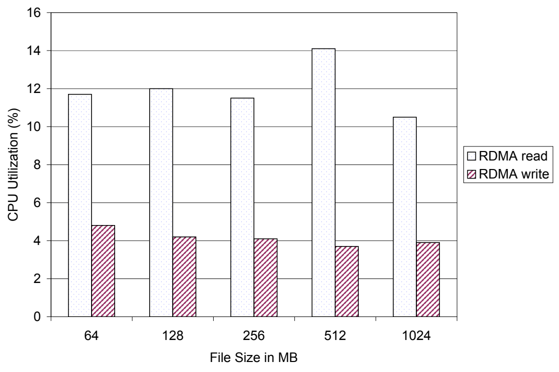

Fig. 16. CPU Utilization

7. Conclusions

In this paper, we have investigated the design of NFS over RDMA in recent releases of OpenSolaris and discussed its inappropriate exposure of NFS server memory to clients. Accordingly, we have proposed an optimized NFS over RDMA design that is based only on server-initiated RDMA operation, including both RDMA read and write. This design also removes the potential security leak in RDMA read based design and allows more concurrency in file serving. Our design is compliant to NFS direct data placement and RPC over RDMA drafts [11, 12]. Moreover, we devised credit-based flow control mechanisms for efficient and scalable communication of NFS over RDMA. The resulting implementation is intended to be interoperable with other available NFS over RDMA implementations such as Linux NFS over RDMA [33]. Our evaluation results show that RDMA write based RPC transport provides up to 28% improvement to the performance of NFS read operation in single-threaded test cases, and up to 44% in multi-threaded cases. It is also beneficial to the performance of some NFS metadata operations, such as readdir, by up to 15%. In addition, the current implementation does not affect the performance of other NFS operations, such as NFS write, which remains to be built on top of RDMA read.

In the future, we intend to investigate the performance of OpenSolaris NFS over RDMA using NFSv4. We plan to propose possible optimizations and evaluate its performance while maintaining interoperability with implementations from other operating systems such as Linux. We also intend to exploit the use of Advanced InfiniBand capabilities such as scatter/gather and shared received queue (SRQ) for further performance and scalability optimizations of NFS over RDMA.

Acknowledgment

We gratefully acknowledge Tom Talpey from Network Appliance and Dr. Spencer Shepler from Sun MicroSystems for many technical discussions. We also would like to thank members from the Sun MicroSystems InfiniBand team for their technical help.

Additional Project Information – Additional information for this project as well as future code releases can be found on the following website: http://nowlab.cse.ohio-state.edu/projects/nfs-rdma/overview.html.

References

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

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

[3] NFS: Network File System Protocol Specification. http://www.ietf.org/rfc/rfc1094.txt.

[4] SDP Specification. http://www.rdmaconsortium.org/home.

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

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

[7] B. Callaghan. NFS Illustrated. In *Addison-Wesley Professional Computing Series*, 1999.

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

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

[10] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification. http://www.ietf.org/rfc/rfc1813.txt.

[11] B. Callaghan and T. Talpey. NFS Direct Data Placement. http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-nfsdirect-02.txt.

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

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

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

[15] W. Feng, J. Hurwitz, H. Newman, S. Ravot, L. Cottrell, O. Martin, F. Coccetti, C. Jin, D. Wei, and S. Low. Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study. In *SC '03*.

[16] 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.

[17] J. Hurwitz and W. Feng. End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems. *IEEE Micro '04*.

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

[19] 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.

[20] 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.

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

[22] J. Mauro and R. McDougall. The statit program. Solaris Internals: Core Kernel Architecture.

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

[24] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS version 3: Design and implementation. In *USENIX Summer 1994*, pages 137–152, 1994.

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

[26] R. Recio, P. Culley, D. Garcia, and J. Hilland. An rdma protocol specification (version 1.0), October 2002.

[27] A. Romanow and S. Bailey. An Overview of RDMA over IP. In *Proceedings of International Workshop on Protocols for Long-Distance Networks (PFLDnet2003)*, 2003.

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

[29] 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.

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

[31] R. Srinivasan. XDR: External Data Representation Standard. http://www.ietf.org/rfc/rfc1832.txt.

[32] Storage Networking Industry Association. iSCSI/iSER and SRP Protocols. http://www.snia.org.

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

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

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