NFS over RDMA

Brent Callaghan

(brent.callaghan@sun.com**)**

Network bandwidth is growing by orders of magnitude. Yet conventional processing of NFS traffic over gigabit networks gobbles CPU. Using RDMA protocols, we expect NFS to make full and efficient use of gigabit networks.

January 28, 2002

1.0 Introduction

NAS file access protocols, like NFS, allow files to be shared among many servers running different operating systems. Yet this flexibility of NFS is tempered by its requirement for TCP/IP networks, which as yet, do not perform at the speed of SAN fibrechannel networks. Additionally, the protocol processing and data movement required by TCP/IP make heavy demands on the client and the server CPU.

The following sections describe a way to run file access protocols at high speed over fast storage networks without significant CPU overhead. You can enjoy the file sharing advantages of NFS with the performance of a SAN.

2.0 NFS Background

The NFS protocol was originally designed as a LAN protocol for file access over 10Mb Ethernet. Since the NFS protocol emerged in the mid-1980's, it has been used exclusively with, and strongly associated with TCP/IP Networking. The first NFS products ran over UDP, but in the last decade a growing number of implementations provide NFS over TCP connections. Work is under way within the Internet Engineering Task Force (IETF) to extend the NFS protocol to function well over TCP connections that span the Internet: NFS version 4. Like TCP/IP, NFS has a long history, implementations are very robust, the serious bugs were fixed long ago and NFS performance has increased significantly with improvements in host CPU and network bandwidth. Like TCP/IP, NFS implementations are available from all major vendors and yet it is highly interoperable due to annual testing at Connectathon events. It's a safe assumption that any NFS client will work with any Filer. NFS is well known as a tough protocol that is undeterred by transient network failures or Filer outages. Filers can be built from highly-available clusters and clients can easily switch between read-only, replica Filers. NFS provides a robustness that cannot easily be matched by block-based storage on SANs.

3.0 NFS in the Server Room

Today, a common NFS storage configuration is a pool of NFS filers that keep files for a large array of "stateless" application servers. Because the application servers do not have any dedicated storage and are not responsible for providing access to any storage, the failure of an application server doesn't block access to files. Since it keeps no files, a failed application server can be replaced easily. Application processing capacity can be increased simply by adding new servers.

The filers are located within a few meters of the application servers in a controlled, machine-room environment, and are connected to the application servers via fast or Gigabit Ethernet. The same machine rooms often host

FibreChannel connections from application hosts or filers to storage arrays.

4.0 Performance of Storage Networking

Typically, the FibreChannel connections move SCSI data significantly faster than Gigabit connections move NFS data, even though the media bandwidths are the same. Not only do the FibreChannel connections move data faster, but they use less host CPU than NFS running over TCP/IP. The FibreChannel and SCSI protocols are lightweight, short-haul protocols, easily implemented in I/O controllers. These controllers use Direct Memory Access (DMA) to move aligned data directly between the Fibrechannel network and host memory without burden to the host CPU. Credit is due to decades of I/O engineering that have produced architectures that offload protocol processing and data movement from the host CPU. File data moved via the NFS protocol has the potential to be just as fast, if it weren't for the movement of data through TCP/IP stacks that make heavier use of the host CPU and create additional data copies.

NFS has the potential to use similar high-speed networks and CPU offloading to accelerate file sharing to SAN speeds.

5.0 Direct Memory Access

Direct Memory Access (DMA) accelerates the movement of data between host memory and a network interface or I/O controller. Without DMA the device driver must use the host CPU to copy data between memory and the device buffers. DMA allows the device to "take over" the host memory bus and transfer the data itself, leaving the host CPU to work on less mundane work - like transaction processing. The device driver simply notifies the device where the data is (or where it is to go) then "kicks off" the DMA operation. Another benefit of DMA is that it allows a network adapter to use host main memory directly rather than depend entirely on local on-card buffering. This gives the device a lot of flexibility in handling very large data transfers in excess of its local buffering capacity.

6.0 Remote Direct Memory Access

Remote Direct Memory Access is an extension of DMA across an interconnect from one host to another. Rather than transfer data between host memory and a device buffer, RDMA moves data between memory on one host and memory on another host. As with DMA, the host processor(s) just "kick off" the transfer, while the details of moving the data from memory to memory via device buffers and network interconnect are handled completely by the hardware. Like DMA, RDMA is a hardware abstraction that is supported by a variety of application programming interfaces, hardware adapters, and network media. The most commonly implemented RDMA driver interface is VI (Virtual Interface) though more recent developments include the Direct Access Transport API (DAT) and RDMA support for Infiniband.

7.0 Doing RPC over RDMA

RDMA hardware gives us speedy and efficient movement of data from one host memory to another. If the of data represents an RPC call or reply message, then we have the basis for high-speed RPC transport. We are already familiar with the use of RPC over transports like UDP or TCP for NFS, or for memory-to-memory loopback operation between kernel and user address spaces for protocols like autofs. The RPC transport model already accommodates a variety of transports. It is straightforward to add RDMA as an additional transport without any need to modify the RPC protocol, or the application protocols that utilize it. The "plug-in" nature of RPC transports means that it is easy to provide a substantial acceleration for RPC-based protocols without change to the RPC applications, or their administration.

8.0 Use of queues and RDMA ops

The RDMA model for moving data is somewhat different to that of the "data stream" model of TCP connections or the "datagram" model of UDP sockets. With these more conventional models, data can be transmitted "in the blind". Streams or blocks of data can be transmitted to a network address, but the ultimate disposition of the data when it reaches that address is left completely up to the receiving system. The RDMA read and write operations

require that the reader or writer have knowledge of the location and size of the destination buffer at both the source and the destination.

If you write a block of data from the memory of one host to another via RDMA you have to supply the virtual address of the destination buffer - and make sure that the buffer is big enough to accept all the data. Similarly, when reading data from a remote host, the virtual address from which the data is to be read must be known, as well as the location of a correctly sized local buffer. Both VI and Infiniband RDMA models provide message queues that allow short messages to be exchanged. These messages can be used to prompt RDMA read or write operations as well as providing source and destination virtual addresses and message size.

For example, if an RPC client needs to send a call message to a server, it might first send a short message to the server giving the virtual address and size of the call message. This is notification to the server that there is a call message waiting. On receiving this notification message, the server can allocate a buffer of the size needed, then initiate an RDMA read operation to transfer the RPC call message from the client to the server's buffer. When the RDMA read is complete, the server sends a short message to the client to notify it that the message has been received successfully.

A call message could also be sent to the server with an RDMA write operation. The client would send a message to the server requesting the address of a destination buffer big enough to hold the RPC call message. On receiving the server's response, the client initiates an RDMA write operation directly into the server's buffer. When the RDMA operation is complete, the client sends a short message to the server to notify it that the call message is transferred and waiting for attention.

Similar methods can be used to return RPC reply messages. There are lots of different ways that can be used to transfer RPC messages using a combination of RDMA operation and message queues. It is possible to optimize the process to minimize the number of messages exchanged and reduce latency as well as to adapt to differences in RDMA read and write speeds, to optimize bandwidth.

9.0 Additional services (NLM, ACL)

We have been looking at how RDMA can be used to accelerate RPC messages. Yet, we have referred only obliquely to the protocols that use RPC. Of course, the most important protocol in this context is the NFS protocol. However, the NFS protocol does not stand alone. File locking is handled by a separate Network Lock Manager protocol. Because this protocol is also RPC-based, it is also accelerated by the RDMA transport. Additionally, the NFS ACL protocol is used to provide control and viewing of Access Control Lists in filesystems that support them. Again, this protocol is RPC-based, and it too is accelerated by RDMA.

10.0 Protocol Offloading

The use of RDMA to accelerate NFS traffic can be used to improve performance for existing TCP/IP clients. Many customers use NFS filers to provide file services to TCP/IP clients. These clients cannot use RDMA acceleration directly because they are too far away from the Filers, or because they are low-cost workstations that cannot be outfitted economically with RDMA hardware. NFS filers for these clients must handle heavy TCP/IP processing loads when providing NFS services.

This TCP/IP processing load can be off-loaded to a box that moves NFS traffic between TCP/IP and RDMA networks. Incoming RPC calls can be extracted from their TCP/IP packets and sent on to the Filers via fast RDMA. RPC replies from the Filers are obtained via RDMA from Filer memory and returned to the client via TCP/IP. Because the Filer is no longer handling TCP/IP processing, its service latency is reduced and it can absorb a higher NFS load.

11.0 Benefits of NFS over RDMA

At first glance, the benefits of NFS over RDMA may appear to be just "faster NFS". Indeed, applications that already use NFS will benefit from the increased data bandwidth, reduced CPU overhead, and reduced latency.

But if NFS over RDMA performance matches that of "direct attach" or SAN-connected filesystems, then NFS is

no longer a bottleneck, and we can appreciate the file sharing benefits of NFS more widely, even in applications that previously required “raw” disk access.

For example, a database provides high-volume, transaction-based access to large volumes of data. Typically, the database provides its own access methods, along with its own caching and I/O scheduling. It is normal for a database to have “raw” access to a disk partition and organize its own on-disk layout. In this role, a database fills a similar role to a filesystem, it maps data access semantics onto disk data blocks. The downside of having the database manage its own storage is that it burdens the system administrator with additional tools because database partitions are managed differently, e.g. UNIX system administration tools like “df” and “mount” do not work with database partitions. Databases do not benefit from filesystems features like automatic growth of file (table) space, or snapshots.

A database can benefit from filesystem features if tables are mapped onto files within a filesystem. With a properly tuned filesystem, a database can have good performance and benefit from more flexible filesystem administration. By far, the easiest filesystem storage for to install and manage is provided by NFS filers, so databases can have the best of both worlds: improved storage management through NFS filesystems, while maintaining performance.

12.0 Project Status

We have run a prototype implementation over an SCI (SCI (Scalable Coherent Interconnect) connection. The RPC layer in the client and server was modified to transport RPC messages as memory-to-memory operations. This experience confirmed that the RPC abstraction on which protocols like NFS are built, makes it quite easy to run RPC-based protocols over radically different transports.

We are further developing this RPC code to adapt readily to a variety of RDMA transports, such as VI, DAT (Direct Access Transport) and Infiniband®.