Why Compromise?

A discussion on RDMA versus Send/Receive and the difference between interconnect and application semantics

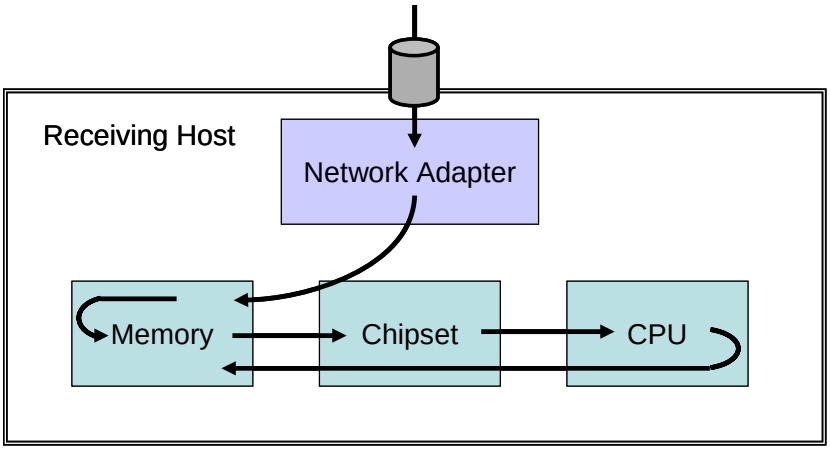
Mellanox Technologies Inc. 2900 Stender Way, Santa Clara, CA 95054 Tel: 408-970-3400 Fax: 408-970-3403 http://www.mellanox.com

Introduction

During the 1990s, universities like Princeton and Cornell conducted research in memory mapped communication (SHRIMP and U-Net). The results drove Compaq (now HP), Intel and Microsoft into drafting a new programming interface called Virtual Interface Architecture (VIA) in 1997. VIA was the foundation for two new serial, high-speed, connectivity proposals. The first proposal, named Future I/O, was driven by Compaq, HP and IBM. The second proposal, named Next Generation I/O, was driven by Intel, Microsoft, and Sun. In 1999, the two initiatives determined they had a common goal and agreed to merge into a single development effort for an optimal interconnect architecture to connect servers and storage. This architecture development was known as System I/O. In October 2000, the first specification of this new interconnect architecture was released, which is now known today as InfiniBand.

One of the driving forces for this new initiative was to find a solution for the most common issues associated with interconnect architectures – the bottlenecks. There are three distinct problems that slow down data transfers from or to the host and involve processing overhead: context transition from application to the kernel and back, protocol processing, and memory copy.

The first and the second are addressed by off-loading protocol processing from the CPU for saving its resources and saving context switching from application to the kernel processing. If the CPU is busy moving data and handling network protocol processing, it is unable to perform computational work, and the overall productivity of the system is severely degraded. InfiniBand adapters have the capability to off-load all the processing of the transport layers.



Memory copy

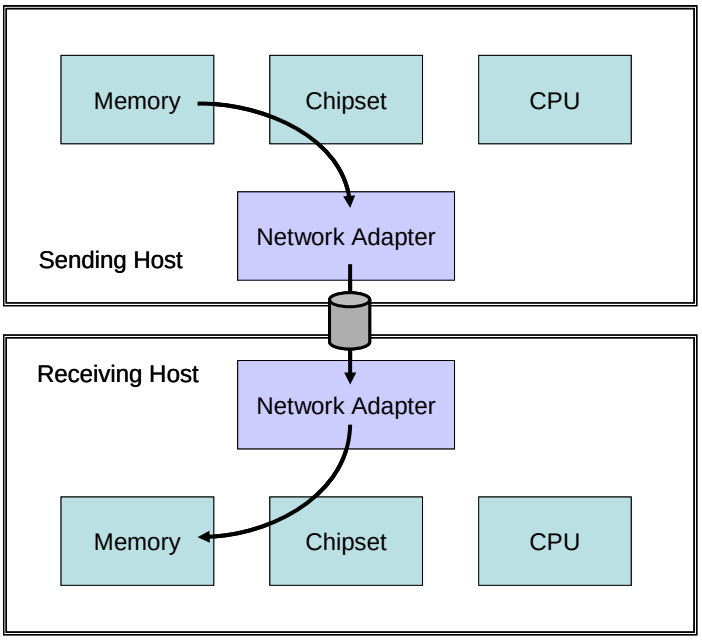
The memory copy overhead includes the resources required to copy data buffers from the network device to the kernel memory and then from the kernel memory to the application memory. This approach requires multiple memory accesses before the data is placed in its final destination. While it is not a major problem for

small data transfers, it is a big problem for larger data transfers. This is where zero-copy capabilities eliminate memory bandwidth bottleneck without involving the CPU in the network data transfer. Mellanox InfiniBand adapters provides zero-copy capabilities with both Send/Receive and Remote Direct Memory Access (RDMA) semantics

In this article, a discussion on RDMA versus Send/Receive and the difference between interconnect and application semantics will be outlined. Must we choose one semantic over the other or are both essential to provide the application the desired performance, flexibility and scalability now and in the future?

RDMA Semantics

RDMA (Remote Direct Memory Access) usually refers to three features: Remote direct memory access (Remote DMA), asynchronous work queues, and kernel bypass. Remote DMA is the ability of the network adapter to place data directly to the application memory. RDMA is also known as a “one-sided” operation in the sense that the incoming messages are being processed by the adapter without involving the host CPU. The data comes with information about where it’s supposed to go, and the receive side does not need to interfere with the data placement - a.k.a direct placement.



Zero-copy flow

Asynchronous work queue is the common interface of RDMA capable adapters between the adapter and the software, also known as verbs interface. The queue objects named queue pair (QP), includes a pair of work queues: a send queue and a receive queue, and completion queues (CQ). The user post an operation

on one of the work queues, then the operation executes asynchronously, and once it is done, the adapter places work completion information in the CQ. Operating asynchronously like this makes it easier to overlap computation and communication.

Kernel bypass is typically an RDMA capable adapter ability. It allows user space processes to do fast-path operations (posting work requests and retrieving work completions) directly with the hardware without involving the kernel. Saving system call overhead is a big advantage, especially for high-performance, latency-sensitive applications.

RDMA capability does not mean it is the only capability

InfiniBand supports both message semantics (a.k.a. Send/Receive) and RDMA. RDMA operations include RDMA Write (one node writes data directly into a memory buffer of a remote node), RDMA Read (one node reads data directly from a memory buffer of a remote node) and RDMA Atomics (combined operation of reading a memory location, optionally the value, and changing/updating the value if necessary).

With Send/Receive (also know as two sided-operations) operations, the source node sends a message and destination node indicates where the data is going to be placed. While in RDMA operation, the source side has all the necessary information on the target placement of the data. For Send/Receive operations, the two sides need to take part in the data transfer.

Both InfiniBand RDMA and Send/Receive semantics can avoid memory copy (also called zero-copy operations). For TCP/IP networks the case is quite different, where iWARP is essential for avoiding memory copy.

Data transfer semantics

InfiniBand is capable of placing data directly to the user or kernel space by using RDMA or Send/Receive operations. In both cases, the destination's adapter figures the location of the data in host memory, either according to the data included in the message for RDMA operations, or according to the appropriate receive work request set by the destination node. The destination buffer can be in the user space or the kernel space.

The difference between RDMA and Send/Receive is the way the destination node finds the host memory destination address for the incoming data. RDMA messages carry the information and therefore do not need the destination CPU cycles for the data transfer. Send/Receive messages do not carry this information and the destination node CPU needs to post receive work requests for the data placement. Lack of receive WQE at the time an incoming message arrives is handled by the adapter (with pure hardware mechanisms without any

software involvement) and does not cause a fatal error. A notification is sent back to notify the sender that the receiver is not ready for data transfer. Furthermore, the hardware resources that are needed for RDMA are the same as for Send/Receive.

When comparing the raw performance of Send/Receive and RDMA semantics on a specific interconnect, different architectures will show different results, but this is related to the adapter implementation. Mellanox InfiniBand implementations show the same bandwidth numbers, but there is a gap of several hundreds nanoseconds in favor of RDMA operations. On the other hand, Mellanox is about to introduce a new HCA architecture where Send/Receive latency will match those of RDMA. Myrinet-GM can show a difference of up to 10% in favor of Send/Receive, and the new generation Myrinet-MX does not officially support RDMA. QLogic InfiniPath does not have the capability of native RDMA and therefore its RDMA software implementation demonstrate higher latency than Send/Receive.

RDMA and Send/Receive create the perfect match for connecting servers and storage. There is no need to compromise and use only one option for every need, when both are available in the same adapter. The decision on which option to use at a given time or maybe even both, and it is up to the application to decide what is more suitable depends on the application. RDMA is typically associated with large data movement (as it does not require the remote side to be involved) and Send/Receive with small data transfers. In fact, RDMA is being used in many other ways, in order to improve and optimized application performance.

Message Passing Interface (MPI)

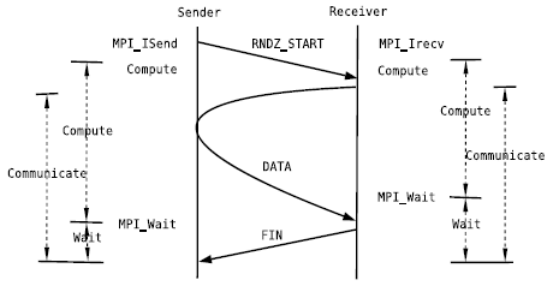
One needs to have a distinction between the native adapter RDMA and Send/Receive semantics and the application RDMA and Send/Receive ones. You can execute MPI Send/Receive operations with either InfiniBand RDMA or Send/Receive operations. The device-specific driver uses the lowest latency options available. Thus, Myrinet will use Send/Receive and Mellanox RDMA. QLogic has a proprietary interface to the adapter and therefore uses its proprietary semantics.

MPI protocols can be broadly classified into two types: Eager and Rendezvous. In the Eager protocol, the sender sends the entire messages to the receiver, which needs to provide sufficient buffers to handle those incoming messages. This protocol has minimal startup overhead and is typically used for small messages. Send/Receive operations are the common implementation. The destination MPI layer controls the Eager buffers allocation and performing the MPI tag matching. MPI tag matching can be done by the adapter (but this is not common), by a kernel process or by a user process (MVAPICH). In this protocol, the MPI is responsible for the message copy from the eager buffers to the

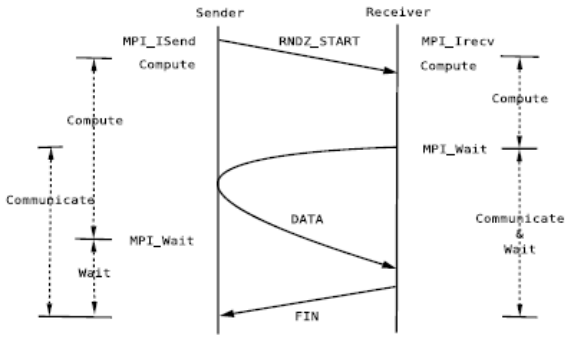
application buffers. The MPI Send/Receive operation can be implemented with InfiniBand RDMA or Send/Receive, as one can RDMA the data to the eager buffers (or any buffer). It is a matter of the low-level MPI implementation.

The Rendezvous protocol is typically used for large data transfers. Since the message is too large to be handled by the eager buffers, the sender and the receiver negotiate the buffer availability prior to the actual transfer. It is critical to avoid unnecessary message copies for higher performance. Since the buffer location is known before the data transfer, RDMA operations are the perfect match. RDMA Write or Read based approaches can totally eliminate intermediate copies. RDMA Read can increase the computation and communication overlap for higher total system efficiency. The usage of RDMA Read will also save interrupts on the sender side, reducing the sender side CPU overhead.

Dhabaleswar K. Panda et al, the Ohio State university, presented the benefits of using RDMA Read operations in a paper “RDMA Read Based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits”, Symposium on Principles and Practice of Parallel Programming (PPOPP'06), March 29-31, 2006, Manhattan, New York City. In the paper, Dhabaleswar K. Panda has show how new designs can achieve nearly complete computation and communication overlap.

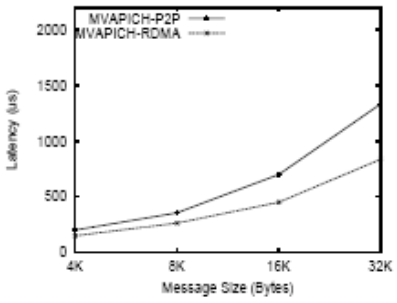


(b) RDMA Read with Interrupt Computation Overlap



(b) RDMA Read Computation Overlap

In another paper, “High Performance RDMA Based All-to-all Broadcast for InfiniBand Clusters” presented at the International Conference on High Performance Computing (HiPC 2005), December 18-21, 2005, Goa, India, D. K. Panda at el showed the advantages of using RDMA for collective operations. Collective operations are being used in many applications such as matrix multiplication, lower and upper triangle factorization, solving differential equations, and basic linear algebra operations. RDMA offers

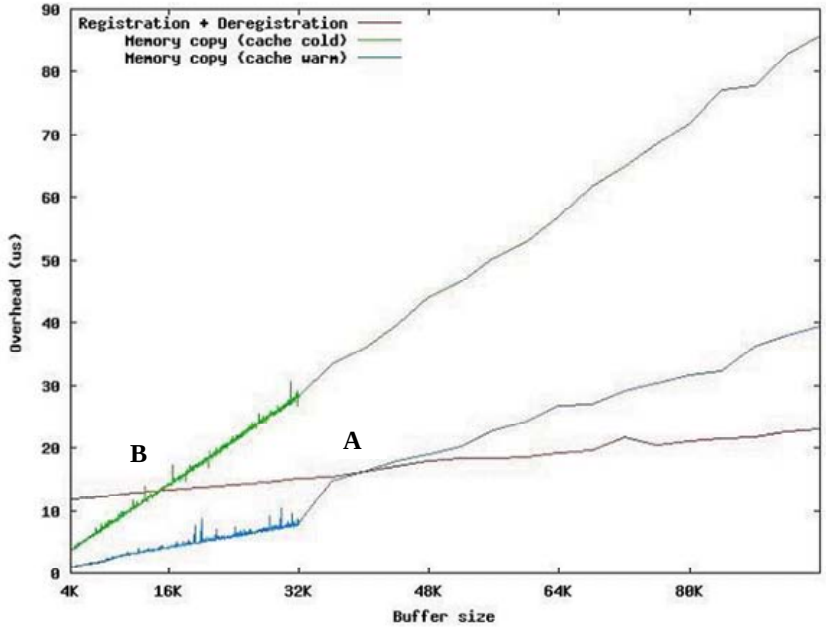


memory semantics which allow MPI Collective operations to be efficiently implemented to achieve lower latency (37% improvement as shown in the paper) and greater scalability.

Douglas Doerfler and Ron Brightwell from Sandia National Laboratories have created a new MPI benchmark for measuring the application availability as an indication for the Send/Receive overlap capabilities. The paper named "Measuring MPI Send and Receive Overhead and Application Availability in High Performance Network Interfaces" will be published at the EuroPVM/MPI, September 2006. The results in the paper do not represent the adapter's behavior but rather the software and the MPI driver implementation of the MPI tag matching (which has nothing to do with the adapter) and partitioning between different threads. The main overhead for MPI is the tag matching and InfiniBand MPIs, such as MVAPICH, do the matching with a user space process. As a result, the overhead will increase once the MPI shift from Eager mode to Rendezvous. Doing the tag matching in a kernel process, like other adapters, will still require the same resources, but will be hidden from the user space. There are several approaches that can be used, instead of the current software implementation, such as a separate thread or kernel module that can deal with tag matching while the main thread is doing computation, etc.

Memory Registration

From a certain message size, Eager model is too expensive and the zero-copy approach provides superior performance results. Zero-copy Rendezvous requires the destination buffers to be registered prior to the data transfer. There is no debate on the fact that memory registration and deregistration has some overhead in terms of CPU overhead which is determined by the driver implementation. Dr. Loïc Prylli from Myricom measured the tradeoff of doing registration and deregistration for each buffer transfer, versus memory copies.



Without using optimization for reducing the memory registration and deregistration overhead, discussed later in the paper, it is clear to see that zero copy offsets the cost of registration and deregistration as expected from ~32KB message size for cache hot mode (marked as A point) and ~16KB for cache cold mode (marked as B point) . Zero-copy is critical for preserving memory bandwidth and CPU utilization, as you don't want the CPU to copy those large messages.

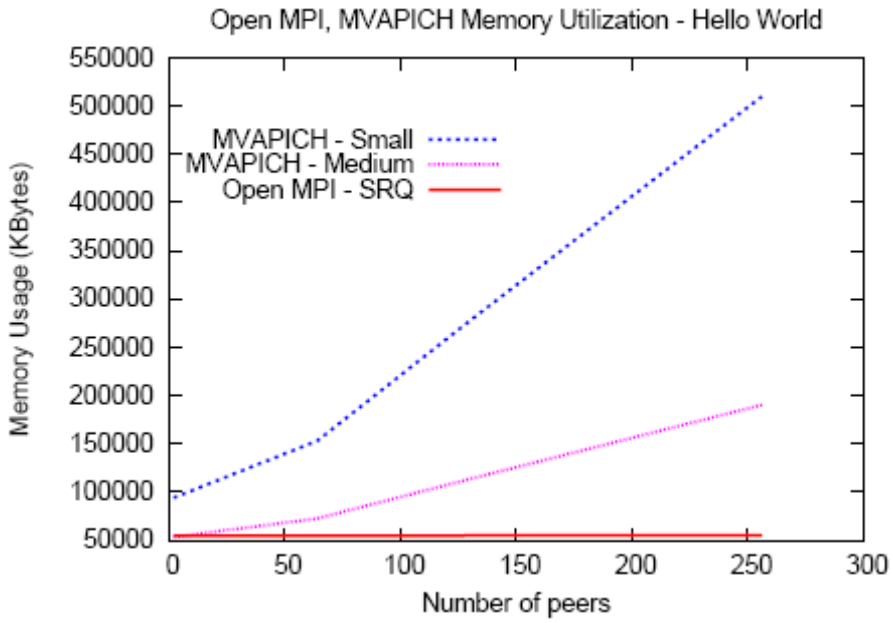
Registration cache is one of the common methods used to dramatically reduce the registration and deregistration cache. Winsocks Direct (WSD) cost to register memory per operations versus zero-copy threshold is around 9KB, meaning above 9KB, memory copy operations become more expensive than zero-copy. Mellanox and Myricom have ways to optimize the registration cost with specific adapter support so that the overhead is cheaper than the memory copy for smaller messages. It's no surprise that Mellanox named this feature FMR - fast memory registration.

Furthermore, the new verbs developed for the InfiniBand specification 1.2 (for example Fast Registration Memory Request) and iWARP include optimizations for registration and deregistration, and target to reduce the threshold to 1KB of message.

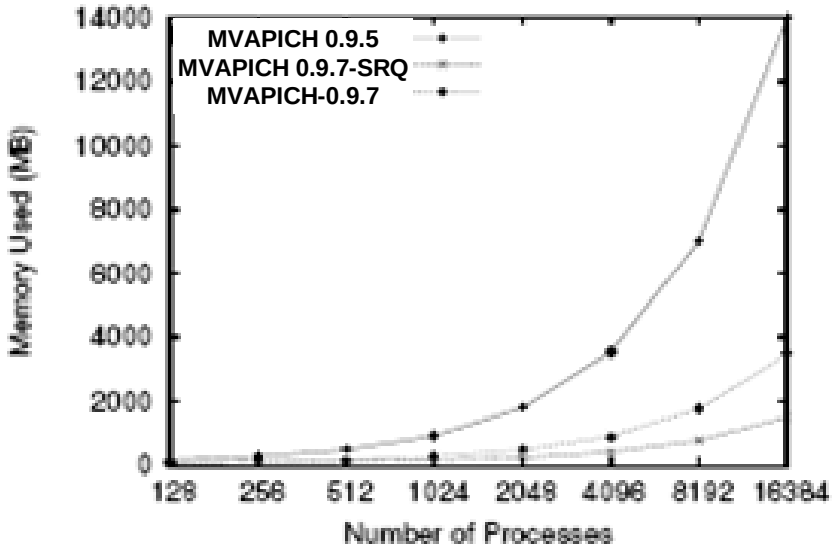
Send/Receive optimizations

The InfiniBand specification was developed for creating a general I/O technology allowing a single I/O fabric to replace multiple existing fabrics. Therefore, it was designed to provide Send/Receive, as well as RDMA capabilities. To enable OS bypass, InfiniBand defines the concept a Queue Pair (QP) as the interface between the host and the adapter. Two-sided Send/Receive operations are initiated by posting a send WQE on a QP's send queue, which specifies the sender local buffer. The remote process post a receive WQE on the corresponding receive queue which specifies a local buffer address to be used as the destination.

When operating in large clusters, there is a need to reduce the memory footprint, and to keep it constant regardless of the number of processes. InfiniBand defines the concept of Shared Receive Queue (SRQ), so that receive resources can be shared among multiple endpoints. The following results from "InfiniBand Scalability in Open MPI", Shipman et al, IPDPS, May 2006, demonstrate the expected results of SRQ implementation in Open MPI and InfiniBand's great scalability.



Dhabaleswar K. Panda has announced MVAPICH (MVAPICH 0.9.7) support of SRQ on March 14th, 2006, and has presented the testing results at the IPDPS 2006 conference, as shown below. The conclusion is exactly the same.



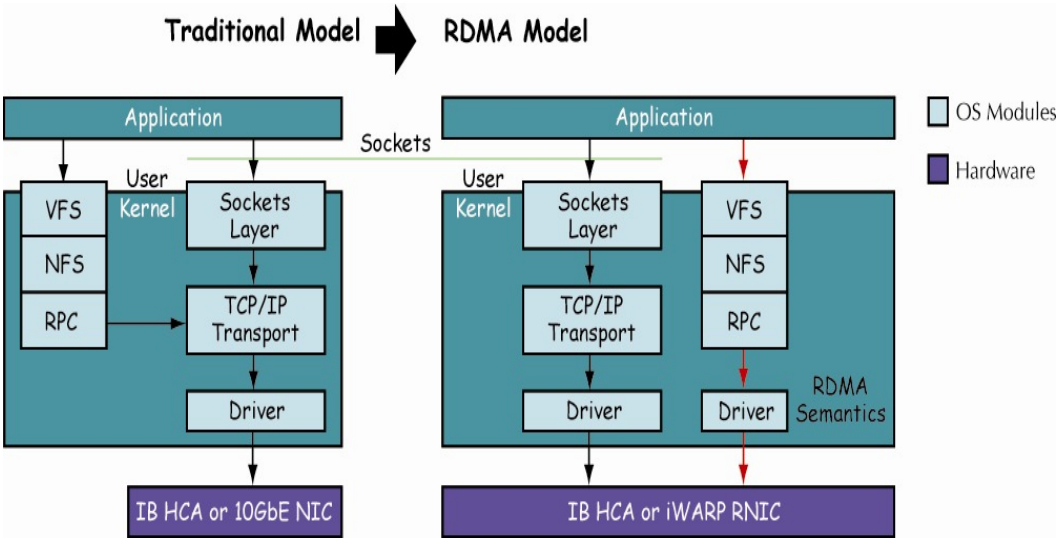
Multiple applications support

InfiniBand differs from Myrinet, Quadrics and QLogic InfiniPath, as it is designed as a general high performance I/O fabric with support for multiple applications in a single wire. InfiniBand drivers provide interfaces not just for MPI applications but also for TCP, socket and storage applications. The storage interfaces include block storage such as SRP, iSER and file systems such as Lustre, GPFS, CFS and NFS.

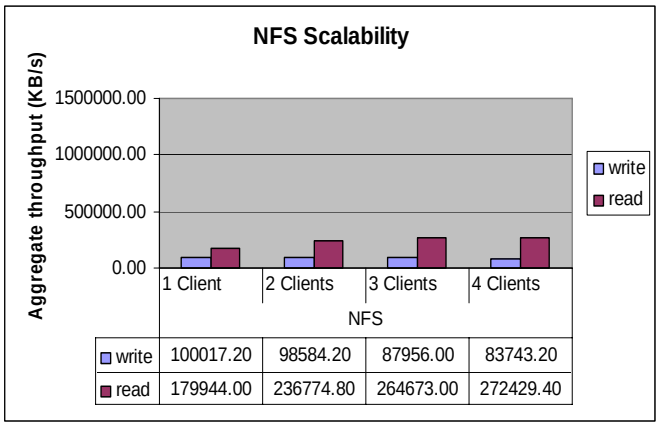
NFS, Network File System, allows a system to share directories and files with others over a network. By using NFS, users and applications can access files on remote systems almost as if they were local files. A common NFS storage configuration is a pool of NFS filers that keep files for a large array of “stateless” application servers. The application servers do not have any dedicated storage and are not responsible for providing access to any storage, therefore a failure of an application server does not block access to files. Furthermore, application processing capacity can be increased simply by adding new servers.

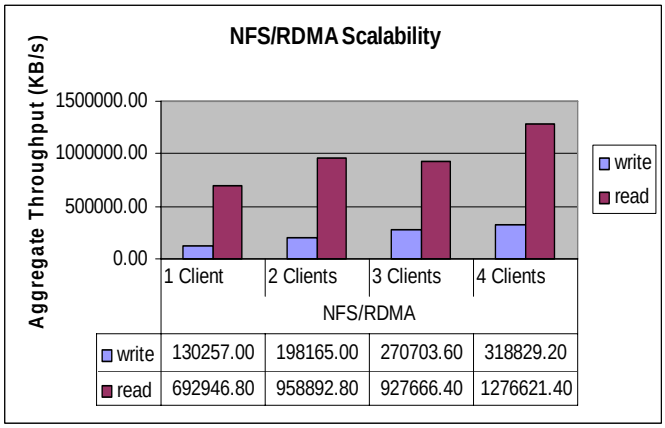
NFS-over-RDMA

The benefits of NFS-over-RDMA are not simply “faster” NFS. Applications that already use NFS will benefit from the increased data bandwidth, reduced CPU overhead, direct I/O (zero copy) and lower latency. If NFS-over-RDMA can match the performance “direct attach” or SAN-connected file systems, than 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.



Helen Y. Chen, Sandia National Laboratories at. el. compared NFS to NFS-over-RDMA in her presentation “Early Experiences with NFS-over-RDMA”, The Commodity Cluster Computing Symposium Baltimore MD, July 25-27, 2006.





The client and the server CPU efficiency were compared between the traditional mode and the RDMA mode. The CPU per MB of transfer is being calculated for the server and the client with by $(\Delta t)*\Sigma(\%CPU/100/file\text{-}size)$. For the client side, NFS-over-RDMA shows 61.86% higher efficiency for writes and 75.47% more efficiency for reads. The server side shows 68.10% higher efficiency for writes and 84.70% higher efficiency for reads. On the scalability side, NFS/RDMA incurred ~½ the CPU overhead and for ~½ of the duration, but delivered 4 times the aggregate throughput comparing to NFS.

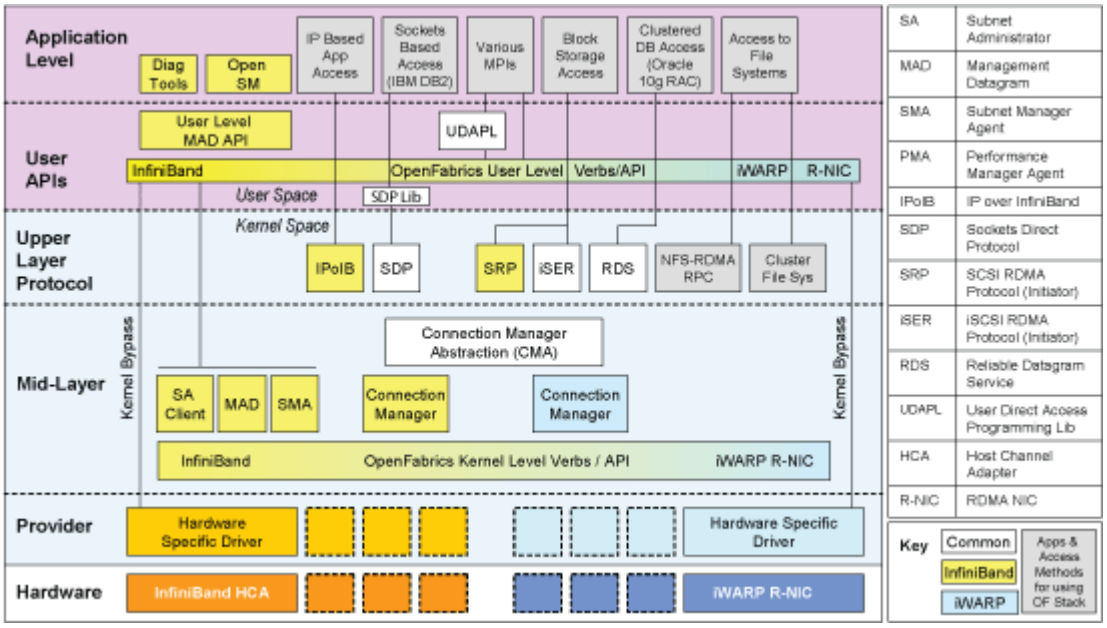
No need to compromise

The choice between Send/Receive and RDMA is driven by the applications. There are cases where Send/Receive is the preferred option and other cases where RDMA is the natural choice. Zero-copy is one of those cases. Indeed, there is some overhead for registration and deregistration memory, but the message size point where is it much more beneficial to use the zero-copy approach is decreasing to hundreds of bytes, with the new IBTA and IETF definitions for InfiniBand and iWARP. Furthermore, RDMA was proven to enhance performance in other cases, such as MPI collective operations, overlapping, checkpointing, atomic access to shared memory data structures, storage applications etc.

The Send/Receive and RDMA application interfaces to the adapters, for InfiniBand and iWARP are open sourced and are constantly optimized under the auspices of the OpenFabrics Alliance. Helen Chen had provided a descriptive diagram of the driver for InfiniBand and iWARP in her paper, showing the variety of the common application program interfaces. Moreover, since the drivers are open sourced, it is simple to modify the code for other propriety applications or to enhance the usage of RDMA or Send/Receive.

The OpenFabrics consortium includes all the major InfiniBand and iWARP companies, includes AMD, Cisco, Dell , IBM, Intel, LSI Logic, Oracle, Sun, the

major USA labs and others, showing the wide-ranging adoption for RDMA technology.



For storage interconnect applications, the situation is different when compared to the MPI compute applications where RDMA and Send/Receive are used together. When the application requires large blocks of data to be moved, RDMA is the only option that provides the required performance, scalability and CPU overhead. It is common to demand optimal storage I/O and high compute I/O in for the same application . One example is when file reads and writes happen before and after the computational periods for the purpose of check pointing and restart mechanisms, etc. In this example, the ability to read and write large quantities of data without interrupting the CPUs is essential, especially when cluster size increases.

RDMA and Send/Receive in the same network provide the user with a variety of tools that are essential for achieving the best application performance and to be able to utilize the same network for multiple tasks, such as compute, storage and management. In the last decade, the industry had made huge progress, both in the network specification, and in the programming interface. With a wide variety of APIs and market adoption, RDMA has completed the missing parts that Send/Receive could not provide, and when combined together, they become the best, flexible, high-performance solution without compromise.

