

Can Software Reliability Outperform Hardware Reliability on High Performance Interconnects? A Case Study with MPI over InfiniBand

Matthew J. Koop

Rahul Kumar

Dhableswar K. Panda

Network-Based Computing Laboratory
The Ohio State University
Columbus, OH 43210

{koop, kumarra, panda}@cse.ohio-state.edu

ABSTRACT

An important part of modern supercomputing platforms is the network interconnect. As the number of computing nodes in clusters have increased, the role of the interconnect has become more important. Modern interconnects, such as InfiniBand, Quadrics, and Myrinet have become popular due to their low latency and increased performance over traditional Ethernet. As these interconnects become more widely used and clusters continue to scale, design choices such as where data reliability should be provided are an important issue.

In this work we address the issue of network reliability design using InfiniBand as a case study. Unlike other high-performance interconnects, InfiniBand exposes both reliable and unreliable APIs. As part of our study we implement the Message Passing Interface (MPI) over the Unreliable Connection (UC) transport and compare with the Reliable Connection (RC) and Unreliable Datagram (UD) transports for MPI. We detail the costs of reliability for different message patterns and show that providing reliability in software instead of hardware can increase performance up to 25% in a molecular dynamics application (NAMD) on a 512-core InfiniBand cluster.

Categories and Subject Descriptors

D.4.4 [Operating Systems]: Communications Management;
D.4.9 [Operating Systems]: Systems Programs and Utilities

General Terms

Experimentation, Performance

Keywords

MPI, Reliability, InfiniBand

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICS'08, June 7–12, 2008, Island of Kos, Aegean Sea, Greece.
Copyright 2008 ACM 978-1-60558-158-3/08/06 ...\$5.00.

1. INTRODUCTION

As clusters continue to grow in scale, the cluster interconnect has a significant influence on application performance. Modern high-speed interconnects such as InfiniBand [9], Myrinet [4], and Quadrics [19] have delivered lower latency and higher performance than traditional Gigabit Ethernet and have become popular cluster interconnect. Of these, InfiniBand is an increasingly popular interconnect in cluster computing due to low latency (1.0-3.0 μ sec) and high bandwidth, as well as other features such as Remote Data Memory Access (RDMA). Many large clusters, such as the over 60,000-core TACC Ranger [25] and 9216-core LLNL Atlas cluster [13] use InfiniBand as their primary interconnect.

The Message Passing Interface (MPI) [15] is the dominant parallel programming model on these large clusters. Given its involvement in many applications, the MPI library has a key role in performance. Optimizations in the MPI library can seamlessly improve the performance of many applications.

Given that both MPI and the network interconnect are so critical to performance in cluster environments, it is important to examine the assumptions traditionally made in each. In many modern interconnects, such as InfiniBand and Quadrics, the most commonly used interface is a reliable one where reliability is provided in the network device hardware or firmware. With this hardware support the MPI can be designed without explicit reliability checks or protocol.

In this paper we examine this common assumption. InfiniBand hardware supports three transports, Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). On Mellanox [1] hardware, the reliability for the RC transport is handled in firmware on the network device. Given this, we use InfiniBand as a platform to evaluate the tradeoffs associated with providing reliability in the MPI layer as compared with a hardware-based design. We compare all three InfiniBand transports, RC, UC and UD in a common codebase with and without software reliability on a variety of communication patterns in this study. To the best of our knowledge, this is the first work to address this issue in such a comprehensive manner on high-speed interconnects.

There are two main questions this work seeks to address:

- Can software-based reliability outperform hardware-based reliability methods, and if so, under what conditions?
- What are the costs associated with providing reliability in software?

As part of our work we design an MPI library that allows independent evaluation of each of the three InfiniBand transports in a common codebase. Additionally, software-level reliability can be turned on and off for each of the transports. We evaluate these combinations using two molecular dynamics packages on a 512-core cluster and report up to a 25% increase in performance over the RC transport by using a software-level reliability layered over the UC transport.

The rest of the paper is organized as follows: In Section 2, we discuss related work. Following in Section 3 we give background on various high-performance interconnects and technologies. Section 4 provides additional background on InfiniBand and the provided transports. Our evaluation methodology is explained in Section 5. Reliability designs are covered in Section 6 and describe our implementation in Section 7. We evaluate the different reliability methods with several benchmarks in Section 8. We conclude and discuss future work in Section 9.

2. RELATED WORK

Other researchers have explored providing reliability at the MPI layer as part of the LA-MPI project [7, 3]. LA-MPI is focused on providing network fault-tolerance at the host to catch potential communication errors, including network and I/O bus errors. Their reliability method works on a watchdog timer with a several second timeout, focusing on providing checksum acknowledgment support in the MPI library. Saltzer et al. [23] discusses about the need for end-to-end reliability instead of a layered design.

Fast Messages (FM) [18] implements an ordering and reliability layer on top of Myrinet. This lightweight layer implements internal message buffering for performance and reliability.

Our work is different in that our focus is on evaluating the cost of reliability in hardware versus software. We believe this is the first such study where the hardware and software are fixed, with the only difference being the location of the reliability implementation. We leverage the work of other traditional reliability schemes to ensure a fair comparison.

3. MODERN NETWORK TECHNOLOGIES AND PACKET LOSS

In this section we give background on network technologies used in modern networks and their effect on communication reliability.

Link Level Flow Control: Link level flow control is a critical feature of all networks [2, 6]. Many high-performance System Area Networks (SAN) provide link level credit flow control. These interconnects, such as ASI [14], InfiniBand, and Fibre Channel generally do not drop packets due to congestion [22]. As a result, packet loss is minimal. Ethernet, however, will drop packets due to congestion. In this case, packets do not always arrive at the receiver side.

Cyclic Redundancy Checks (CRC): Modern networks provide a link-level CRC check to provide data reliability. Networks such as InfiniBand implement an end-to-end link-level

CRC as well as per-hop CRC check. Other networks such as Myrinet [4] also provide CRC at the hardware level. Others have advocated for additional memory-to-memory CRC checks to prevent against I/O errors [10].

Reliability: Unlike many of the previous technologies, high-performance interconnects differ in their implementation of reliability. Myrinet requires software-level reliability, Quadrics [19] implements hardware-level reliability, and InfiniBand implements both modes of operation.

Packet Loss: Many modern network technologies implement link level flow control to prevent packet loss from occurring due to congestion. Additionally, many networks provide a link-level CRC check, which represents one of the few times a network will drop a packet. Empirical data confirms these very low drop rates. In our previous work we have evaluated various applications on a 4096-core system with very few network drops for InfiniBand [11].

4. INFINIBAND

In this section we give the required background information on InfiniBand. We start with the communication model and follow with the available transport services.

InfiniBand was designed as a high-speed, general-purpose I/O interconnect, and in recent years it has become a popular interconnect for high-performance computing to connect commodity machines in large clusters.

4.1 Communication Model

A Queue Pair (QP) model is used for all communication in InfiniBand. A QP consists of two queues, a Send Queue (SQ) and a Receive Queue (RQ) for initiating send and receive operations, respectively. Each QP is associated with a Completion Queue (CQ), allowing an application to poll or use an event-based interface to receive notification of operation completion.

There are two sets of communication semantics in InfiniBand: channel and memory semantics. Channel semantics include send and receive operations that are similar to those found in traditional interfaces, such as sockets, where both sender and receiver must be aware of communication. Memory semantics include one-sided operations where one host can access or modify memory on a remote node without a posted receive; such operations are referred to as Remote Direct Memory Access (RDMA).

In both sets of semantics all memory used for communication must be registered and pinned, not to be swapped out. Since the startup cost for registering memory is high, small messages are generally copied into pre-registered buffers before being sent to the receiver for optimal performance. At larger sizes, it can become advantageous to perform a zero-copy transfer by directly registering the application send and receive buffers and performing an RDMA operation.

To receive a message on a QP using channel semantics, a receive buffer must be posted to that QP. Buffers are consumed in a FIFO ordering. Using a Shared Receive Queue (SRQ) allows receive buffers to be shared across QPs for scalability.

4.2 Transport Services

There are four transport modes defined by the InfiniBand specification: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC) and Unreliable Datagram (UD). Of these, RC, UC, and UD are required to be sup-

Table 1: Feature Comparison of UD, UC, RC

	UD	UC	RC
Reliable Delivery	–	–	✓
In-Order Delivery	–	–	✓
RDMA Support	–	✓	✓
Large Messages	–	✓	✓

ported by Host Channel Adapters (HCAs) in the InfiniBand specification. A comparison of these transports is in Table 1. RD is not required and is not available with current hardware. All transports provide a checksum verification.

Reliable Connection (RC) is the most popular transport service for implementing MPI and other User Level Protocols (ULPs) over InfiniBand. As a connection-oriented service, a QP must be dedicated to communicating with only one other QP. An application that communicates with N other peers must have at least N QPs created. It provides RDMA capability, atomic operations, and reliable service.

Unreliable Connection (UC) provides the same connection-oriented service as RC, but with no guarantees of ordering or reliability. It does support RDMA write capabilities and sending messages larger than the Maximum Transfer Unit (MTU) size. Message segmentation is provided through hardware support.

Unreliable Datagram (UD) is a connection-less and unreliable transport, the most basic transport specified for InfiniBand. As a connection-less transport, a single UD QP can communicate with any number of other UD QPs. UD does have a number of limitations, however. Messages larger than an MTU size, which on current Mellanox hardware is limited to 2KB, cannot be directly sent using UD. Only channel semantics are defined for UD, so RDMA is not supported. Additionally, UD does not guarantee reliability or message ordering.

5. METHODOLOGY

We design our study to evaluate the tradeoffs associated with providing reliability within the network hardware and in the MPI library. In particular, we seek to evaluate the performance and resource requirements of the host for a variety of messaging patterns.

5.1 Native Performance

Before measuring software reliability, the native hardware performance for both reliable and unreliable communication modes must be evaluated. This baseline gives the basic communication performance differences. In the case of InfiniBand, the Reliable Connection (RC) mode has likely been significantly more optimized than the Unreliable Connection (UC) mode due to its more prominent usage in libraries and applications. Since many modern networks provide link level flow control, packet loss is minimal and real-world workloads can be run over unreliable protocols for the purposes of evaluation. The time difference between the reliable and unreliable mode application executions of a workload represent the maximum improvement that can be obtained by using a software-based reliability mechanism.

5.2 Reliability Performance

The main evaluation criteria is the performance that can be obtained through providing reliability in the MPI library instead of hardware. To perform this evaluation, the same messaging workloads must be used on both a software-based implementation of reliability and a hardware implementation of reliability. To isolate this difference, the rest of the hardware and software stack should be identical.

Since InfiniBand provides both reliable and unreliable semantics on the same hardware it offers an ideal platform for our study. Current generation Mellanox InfiniBand adapters allow us to study three transports and isolate the reasons for performance differences:

- Reliable Connection (RC): Reliability and message segmentation are provided in hardware
- Unreliable Connection (UC): Message segmentation is provided in hardware
- Unreliable Datagram (UD): Neither reliability or segmentation is provided in hardware.

These three transports allow us to isolate the tradeoffs of providing hardware reliability and segmentation. In particular, comparing RC and UC gives the difference of hardware-level reliability and software-level reliability. The difference between RC and UC gives insight into the differences between connection-oriented and connection-less transports as well as hardware and software level segmentation.

5.3 Resource Usage

Another important metric to be measured is the additional memory usage incurred by the software to provide reliability support. To provide high performance message passing over an unreliable transport, significant buffering of unacknowledged messages may be required. The maximum memory usage for message buffering will be tracked. Additionally, the number of ACKs issued by the MPI library to provide reliability will also be tracked.

6. PROPOSED DESIGN

The reliability protocol used has a significant influence on two of our evaluation metrics from Section 5 – performance and resource usage. In this section we discuss the suitable reliability protocols.

6.1 Reliability Engine

In our previous work [11] we have evaluated three possible reliability protocols for MPI over the UD transport of InfiniBand. In this work we select the “progress engine” style of reliability, which provided the best performance in the previous evaluation. More details on this can be found in our prior work.

In this method, shown in Figure 1, message segments are acknowledged lazily when the application makes calls into the MPI library. If there is a large skew between processes, there is the possibility of unneeded message retries. Our previous work has showed this level to be acceptable. The next subsection gives the protocol details.

Note that additional reliability modes, particularly those based on kernel involvement, are not evaluated here. This work specifically targets providing reliability within the MPI library. It is anticipated that a kernel-level interrupt scheme

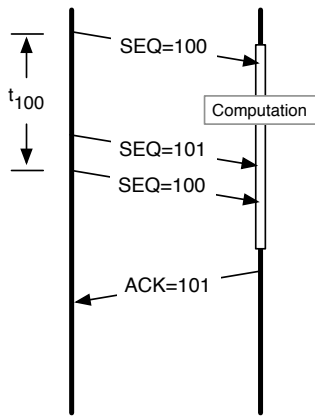


Figure 1: Progress-Based Acknowledgments

may in some cases be able to provide even higher performance than the strictly user-level method that we are evaluating.

6.2 Reliable Message Passing

For small messages, those less than 8KB, our design uses the traditional sliding window protocol. The sender issues packets in order as there is available space in the window. In this manner the window represents the maximum number of unacknowledged message segments outstanding. Additional send operations occurring when the send window is already full are queued until outstanding operations have been acknowledged.

To enable reliability, after each message segment is sent it is tagged with a timestamp and added to an unacknowledged queue. This timestamp is provided through the Read Time Stamp Counter (RDTSC) assembly instruction. If the MPI library detects that the timestamp has aged past a pre-configured timeout value ($t_{timeout}$) without an acknowledgment, the message segment is retransmitted. The receiver will discard all duplicate messages and send an explicit ACK to prevent additional retransmissions.

ACKs are also “piggybacked” onto all other messages being sent to the sender since many applications have bi-directional communication patterns. This significantly reduces the number of explicit ACKs that are required. If an ACK has not been piggybacked after $t_{timeout}/2$ usec, the receiver issues an explicit ACK to the sender.

Our design also uses ACKs for flow control in both the reliable and unreliable modes. This signaling is used to relay the number of free receive buffers available to the sender. Our reliability method also makes use of these control messages that are already being issued in the background.

6.3 Preserving Zero-Copy over Unreliable Connection

On interconnects with RDMA capabilities, a zero-copy protocol is often used to transmit large messages. In zero copy protocols the sender and receiver use small control messages to match the message and then the message data is placed directly in user memory. A zero-copy protocol can significantly increase bandwidth and reduce cache pollution. Instead of performing data copies in the send and receive paths, within user-space or the kernel, a zero-copy approach directly sends the data from the source application buffer to the final destination buffer.

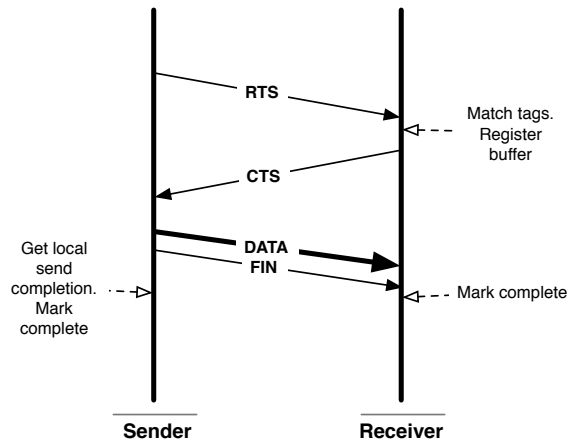


Figure 2: Traditional Zero-Copy Rendezvous over RC

On InfiniBand, a handshake protocol (rendezvous) is used. As shown in Figure 2, the sending process sends a Request to Send (RTS) message with message tag information. Upon discovery of the message by the MPI layer at the receiver end, the receiver will send a Clear to Send (CTS) to the sender. If the sender receives a CTS, then it can use RDMA Write to send the message data directly to user application buffer at remote side. Thus, RDMA Write provides a convenient method to perform zero-copy protocols when MPI message tags are matched by the MPI library.

To maintain these zero-copy semantics for an unordered and unreliable transport, this protocol must be adjusted. The traditional RDMA Write rendezvous protocol over RC makes use of both of the assumptions of ordering and reliability. A Finish (FIN) message is issued directly after the RDMA write data message and upon receipt the receiver knows the data has already arrived and can mark the receive as complete. Similarly, upon getting a local send completion from the CQ for the FIN message, the sender can mark the send complete.

Unfortunately, with an unreliable transport, neither of these mechanisms can be used. Since UC is unordered, the FIN message may arrive before the data is placed by the RDMA Write operation. Additionally, the local send completion on the sender no longer indicates successful data placement on the receiver.

To adapt the protocol for UC, the RDMA write operation must signal the receiver as well. InfiniBand has the option to send a 32-bit *immediate* data field with an RDMA write operation that generates a CQ entry on the target. We leverage this capability to allow the receiver to match the RDMA Write and FIN data messages. The FIN message is sent through the usual reliability channels with a sender-side retry – thus guaranteeing it will be received by the receiver. As seen in Figure 3(b) If the RDMA Write immediate data is not received within a timeout period, a NACK message is sent to the sender, which will trigger a resend of the RDMA Write data message. However, if the immediate data is received within the timeout a ACK message is immediately returned, allowing the sender to mark the send operation complete (Figure 3(a)).

It is important to note that the additional ACK requirement may decrease application performance in some cases.

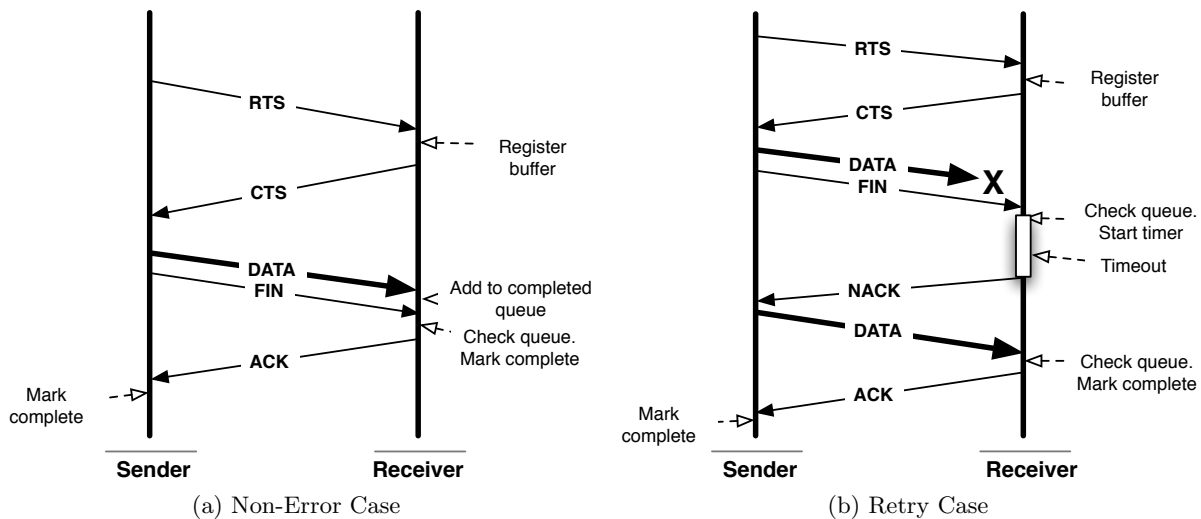


Figure 3: Zero-Copy over Unreliable Connection

In particular for blocking MPI_Send operations, the sender cannot mark the send complete and proceed with application execution until the ACK is received.

7. IMPLEMENTATION

We have implemented our design over the verbs interface of the OpenFabrics/Gen2 stack [17]. Our prototype design is based on MPICH [8] from Argonne National Laboratory and MVAPICH. MPICH defines an Abstract Device Interface (ADI), that allows the device and transport specific information to be encapsulated. We develop an ADI component based on the MVAPICH ADI components. MVAPICH [16] is a derivative of MVICH [12] (an MPI over VIA [5]) from the Ohio State University that has been significantly redesigned and optimized for InfiniBand. Both MVICH and MVAPICH were originally created for reliable connection-oriented transports. Recently, based on our previous work [11], a UD-based ADI device has been included in the MVAPICH stack.

We extend the UD-based ADI device to include support for both RC and UC transports, as well as the zero-copy reliability design described earlier. All transports are incorporated into a single codebase to allow isolation of hardware performance characteristics and avoid software differences. UC and RC message handling code paths are identical aside from setup code. Since UC does not currently support Shared Receive Queue (SRQ), both UC and RC messaging layers are implemented without it.

Additionally, the library is instrumented to give insight into performance differences by tracking message statistics. Information such as the memory usage of the reliability protocol as well as message rates and sizes are tracked.

8. EVALUATION AND ANALYSIS

In this section we evaluate each of the following combinations to determine the cost of reliability:

- **RC - Native:** RC transport with no software reliability
- **RC - Reliable:** RC transport with the software reliability layer enabled. Although in production such a

combination would not be used, it allows us to observe the cost of reliability above the RC transport.

- **UC - Native:** UC transport with no software reliability. This is not a ‘safe’ configuration for production since message drops can occur. For research purposes, however, it gives an upper bound on the performance improvement that can be provided by a software reliability method.
- **UC - Reliable:** UC transport with software reliability.
- **UD - Native:** UD transport with no reliability. Same caveats and rationale as ‘UC - Native’ apply here.
- **UD - Reliable:** UD transport with software reliability.

We evaluate first with microbenchmarks to observe the basic performance of each of the transports. Next we evaluate each of the combinations on two molecular dynamics applications, NAMD and LAMMPS.

8.1 Experimental Setup

Our experimental test bed is a 560-core InfiniBand Linux cluster. Each of the 70 compute nodes have dual 2.33 GHz Intel Xeon “Clovertown” quad-core processors for a total of 8 cores per node. Each node has a Mellanox MT25208 dual-port Memfree HCA. InfiniBand software support is provided through the OpenFabrics/Gen2 stack [17], OFED 1.2 release.

8.2 Microbenchmarks

In this section we evaluate each of the combinations on three microbenchmarks: ping-pong latency, uni-directional bandwidth, and uni-directional message rate.

Ping-Pong Latency: Figure 4(a) shows the latency of each of the transports with the software reliability toggled on and off. The latency for all transports is nearly identical. For messages under 64 bytes, UD-Native provides the lowest latency.

Uni-Directional Bandwidth: To measure the uni-directional bandwidth, we use the `osu_bw` benchmark [16], which sends a

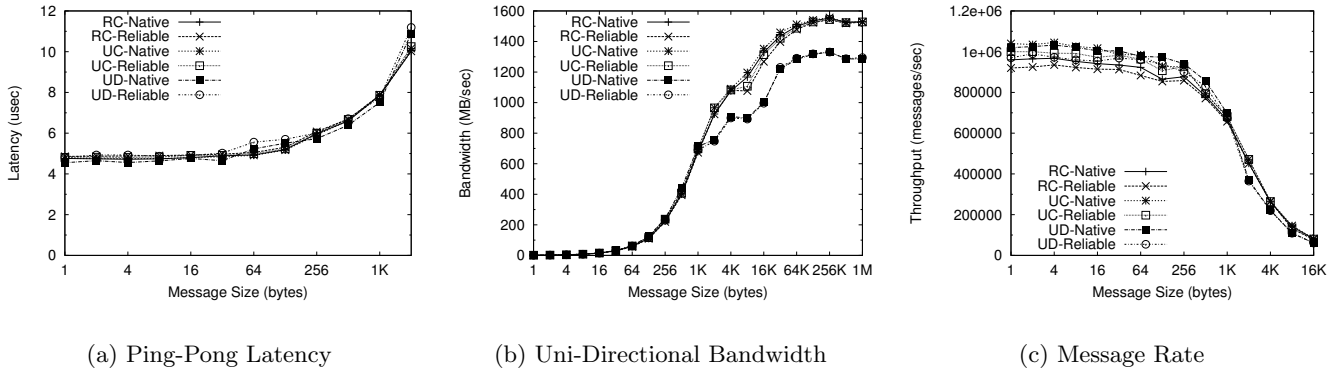


Figure 4: Microbenchmark Transport Performance Comparison

window of sends from one task to another and measures the time until an acknowledgment from the receiver is received. Figure 4(b) shows the results for each of the evaluation configurations. There is little difference between the UC and RC transports when natively used. The software reliability layer adds a noticeable but minimal decrease of 50 MB/sec in throughput between 8KB and 32KB. The UD transport shows lower performance for all message sizes above the MTU size since all messages above that level must be segmented by the MPI library instead of the HCA.

Message Rate: The message rate is measured in a similar manner as the uni-directional bandwidth. In this case, however, we report the number of messages sent per second instead of the bandwidth achieved. This shows the ability of the HCA to inject messages into the network. The results are shown in Figure 4(c). The results show that the unreliable transports, UC and UD, natively achieve nearly 10% higher throughput than RC. Even when layered with reliability the unreliable transports are able to outperform RC.

8.3 Application Benchmarks

In this section we evaluate each of the reliability combinations with two molecular dynamics applications. We evaluate with both NAMD and LAMMPS applications.

8.3.1 NAMD

NAMD is a fully-featured, production molecular dynamics program for high performance simulation of large bimolecular systems [20]. NAMD is based on Charm++ parallel objects, which is a machine independent parallel programming system. Of the standard data sets available for use with NAMD, we use the `apoa1`, `flatpase`, `er-gre`, and `jac2000` datasets. We evaluate all data sets with 512 tasks.

Figure 5(a) shows the results for each of the evaluation combinations and datasets. Characteristics of the application execution are provided in Table 2. In most of the cases UC-Reliable performs within 1% of RC-Native. Our software-based reliability adds 2%, 1%, and 1% of overhead above UC-Native for `apoa1`, `er-gre`, `flatpase`, respectively – matching that of the hardware.

One particular dataset, the Joint Amber-Charm benchmark (`jac2000`), shows a large performance difference between each of the transports. Table 2 shows that there is a very high message rate used by this application. Per process there are nearly 7000 messages per second transmitted.

Given this higher message rate the overhead caused by the reliability protocol is increased to 7% in the case of UC-Reliable. Even with the software reliability layer, however, the execution time is 25% reduced from RC-Native. The UD transport shows even higher performance, likely due to the lack of connection cache thrashing [24] in addition to the software reliability.

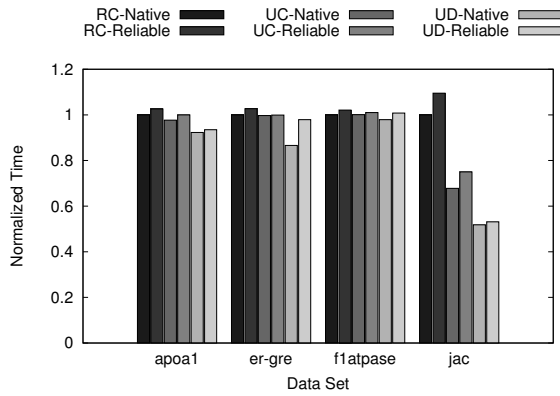
Additionally, as shown in Table 2, there is very little memory required to provide the software-layer reliability. Less than 1MB of memory is used per process at the high-watermark of memory usage for all datasets. The average maximum number of entries per process waiting for acknowledgments at any time was between 50 and 200. This value will depend on the application behavior and synchronization of the application. This number could be limited if needed, although it was not in this evaluation.

8.3.2 LAMMPS

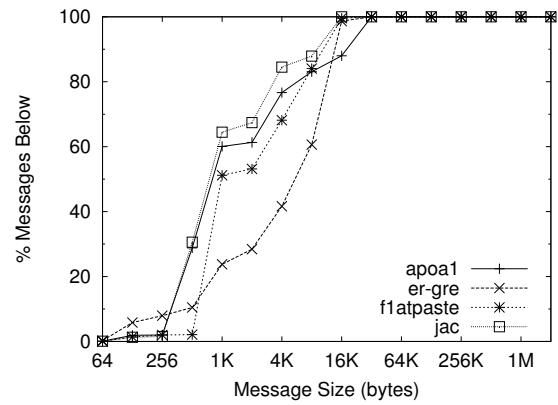
Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [21] is a classical molecular dynamics simulator from Sandia National Laboratories. Several benchmark datasets are available from the LAMMPS website. They are meant to represent a range of simulation styles and computational expense for molecular-level interaction forces. In our experimental analysis, we used the Rhodospin protein (`in.rhodo`), Polymer chain melt (`in.chain`) and EAM Metal (`in.eam`) datasets available from the LAMMPS website. LAMMPS reports the “Loop Time” for a particular benchmark as a measure of time required to simulate a set of interactions.

The results of our evaluation can be found in Figure 6(a). Characteristics of the communication are supplied in Table 3. As with NAMD, the performance between RC-Native and UC-Reliable is nearly identical in most cases. The software-based reliability layer for UC-Reliable adds a 1% and 2% overhead from UC-Native for `in.chain` and `in.eam`, respectively. The combinations with the UD transport perform significantly worse with these datasets, 17% and 6% worse than RC-Native. This is due to the segmentation costs incurred for the large messages in these datasets. As shown in Figure 6(b) 40% of the messages for `in.chain` and 80% of those for `in.eam` are over 8KB.

The `in.rhodo` dataset shows higher performance for both the unreliable transports, even after adding the software reliability layer. The MPI message rate of this dataset is nearly



(a) NAMD Normalized Time

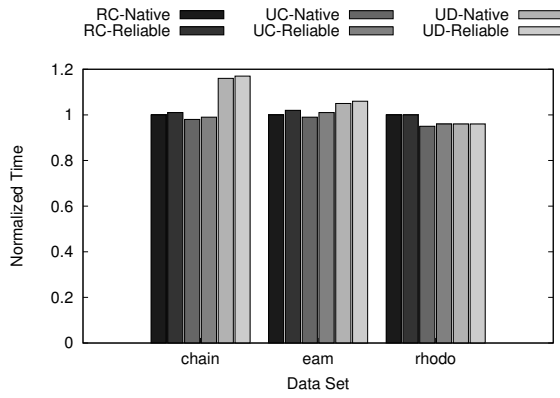


(b) MPI Message Distribution

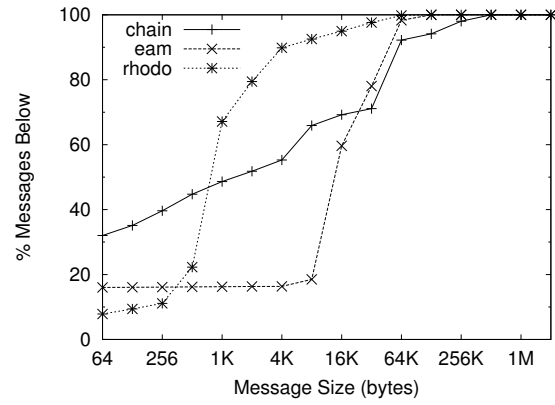
Figure 5: NAMD Evaluation Results

Table 2: NAMD Characteristics Summary (Per Process)

Characteristics			Data Sets				
			apoa1	er-gre	flatpase	jac2000	
General	MPI Message Rate (msg/sec/process)		1579.14	1958.28	2338.16	6979.33	
	MPI Volume Rate (MB/sec/process)		6.27	13.43	9.16	15.89	
	Communicating Peers (peers/process)		504	504	504	504	
Transport Specific	RC	Native	Flow ACKs	234.60	216.02	565.85	193.63
		Reliable	Flow Control ACKs	257.79	50.38	1717.47	26.06
			Reliability ACKs	43106.37	16786.32	118445.53	56299.19
			Avg. Max Reliability Memory	0.68 MB	0.44 MB	0.82 MB	0.33 MB
			Avg. Max Queue Length	124.01	71.34	170.45	65.08
	UC	Native	Flow Control ACKs	234.65	216.07	560.28	194.28
		Reliable	Flow Control ACKs	210.32	41.79	1301.50	37.09
			Reliability ACKs	43149.78	16830.40	118914.76	57981.19
			Avg. Max Reliability Memory	0.64 MB	0.47 MB	0.79 MB	0.35 MB
			Avg. Max Queue Length	144.06	69.00	158.22	61.85
	UD	Native	Flow Control ACKs	287.88	103.82	801.51	301.94
		Reliable	Flow Control ACKs	0.65	0.06	3.56	0.04
			Reliability ACKs	41897.84	16649.10	114419.89	57428.87
			Avg. Max Reliability Memory	0.22 MB	0.11 MB	0.41 MB	0.12 MB
Avg. Max Queue Length			33.19	14.82	38.42	20.20	



(a) LAMMPS Normalized Time



(b) MPI Message Distribution

Figure 6: LAMMPS Evaluation Results

Table 3: LAMMPS Characteristics Summary (Per Process)

Characteristics				Data Sets		
				chain	eam	rhodo
General	MPI Message Rate (msg/sec/process)			372.34	332.73	4849.74
	MPI Volume Rate (MB/sec/process)			9.33	7.14	13.2
	Communicating Peers (peers/process)			15	8	90
Transport Specific	RC	Native	Flow ACKs	37.39	0.51	428.86
			Flow Control ACKs	11.34	0.12	54.53
		Reliable	Reliability ACKs	1817.20	9310.43	28696.25
			Avg. Max Reliability Memory	0.08 MB	0.04 MB	0.13 MB
			Avg. Max Queue Length	16.83	7.67	65.09
	UC	Native	Flow Control ACKs	37.26	0.53	428.86
			Flow Control ACKs	11.39	0.11	55.31
		Reliable	Reliability ACKs	1814.69	8992.67	28670.16
			Avg. Max Reliability Memory	0.08 MB	0.04 MB	0.13 MB
			Avg. Max Queue Length	16.73	7.84	65.30
	UD	Native	Flow Control ACKs	21.24	0	164.70
			Flow Control ACKs	0	0	0.19
		Reliable	Reliability ACKs	1983.67	10443.20	29322.30
			Avg. Max Reliability Memory	0.07 MB	0.07 MB	0.12 MB
			Avg. Max Queue Length	8.82	7.61	8.47

5000 messages/sec, an order of magnitude higher than the other datasets. As seen in the NAMD `jac2000` dataset, the higher message rate seems to favor the unreliable transports.

As with NAMD, the memory required to support reliability at the MPI layer is minimal. Less than 128KB of memory was required at maximum for the evaluated datasets. A large number of explicit ACK messages were sent by the reliability layer, but this had minimal impact on performance.

8.4 Analysis

In both of the application runs it was observed that datasets where message rates were relatively low – less than 3000 messages/sec – the performance between UC and RC was similar and the software-based reliability protocols added little overhead.

When the message rate is increased the performance of the unreliable transports improves dramatically over RC. It is likely that HCA resources are becoming exhausted and the software-based approach that makes use of the host CPU and delayed ACKs leads to the improvement. This seems to be particularly prevalent when there are a large number of communicating peers.

In the case of the `jac2000` benchmark for NAMD there was a significant overhead to provide reliability in software. Upon further examination, it appears that the communication pattern is not as bi-directional as others. As a result, after controlling for execution time, the number of explicit (non-piggybacked) ACKs is nearly double that of other datasets (including that of `in.rhodo` for LAMMPS).

Our work shows that providing reliability at the MPI layer is not only feasible, but in some cases may provide higher performance. Furthermore, it can be done with little memory usage on the host.

9. CONCLUSIONS AND FUTURE WORK

Interconnects continue to be a significant factor in the performance of clusters. Modern high-speed interconnects such as InfiniBand, Myrinet, and Quadrics have delivered lower latency and higher performance than traditional Gigabit Ethernet. In many high-speed interconnects reliability is performed within the network hardware. In this work we seek to examine this assumption that reliability should be implemented in hardware for highest performance.

In this paper we have examined the performance of software and hardware reliability by taking InfiniBand as a case study. Since InfiniBand offers unreliable and reliable transports, with the latter being implemented in hardware on Mellanox HCAs, it provides an ideal platform for this evaluation. We have designed and implemented an MPI library which can run over each of the InfiniBand transports, each of them with and without a software reliability layer enabled. This integrated evaluation allows a more detailed and precise analysis than otherwise could be achieved using separate adapters or MPI libraries. To the best of our knowledge, this is the first work to address this issue in such a comprehensive manner on high-speed interconnects.

We have evaluated the various configurations on two molecular dynamics MPI applications with different data sets on a 512-core InfiniBand cluster. We give detailed analysis of the application and data set communication characteristics and show how these influence the performance. For NAMD we observe up to 25% improvement using a software-based reliability protocol instead of the hardware-based reliability.

In many other cases we see little to no difference in providing reliability in software instead of hardware. We further show that the amount of memory required to provide software-based reliability is minimal, a maximum of 800KB, for each process.

In the future we plan to evaluate providing other forms of reliability in software as well. In particular, we are interested in looking at performing CRC checks within the host software stack to avoid errors that may occur in I/O bus transactions. We plan to evaluate additional applications and see how protocols built over unreliable transports influence performance in both positive and negative ways.

10. ACKNOWLEDGMENTS

This research is supported in part by Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755, National Science Foundation grants #CNS-0403342 and #CNS-0702675; grants from Wright Center for Innovation and State of Ohio #WCI04-010-OSU-0; grants from Cisco Systems, Intel, Mellanox, QLogic, and Sun Microsystems; and equipment donations from Advanced Clustering, AMD, Appro, Intel, IBM, Mellanox, Microway, QLogic and Sun Microsystems.

11. REFERENCES

- [1] Mellanox Technologies. <http://www.mellanox.com>.
- [2] A. S. Tanenbaum. Computer networks. Prentice-Hall 2nd ed., 1989, 1981.
- [3] R. T. Aulwes, D. J. Daniel, N. N. Desai, R. L. Graham, L. Risinger, M. W. Sukalski, and M. A. Taylor. Network Fault Tolerance in LA-MPI. In *Proceedings of EuroPVM/MPI '03*, September 2003.
- [4] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, and J. S. S. Wen-King. Myrinet: a gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Feb 1995.
- [5] Compaq, Intel, and Microsoft. VI Architecture Specification V1.0, December 1997.
- [6] M. Gerla and L. Kleinrock. Flow control: A comparative survey. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 28(4):553–574, Apr 1980.
- [7] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalski. A Network-Failure-Tolerant Message-Passing System for Terascale Clusters. *International Journal of Parallel Programming*, 31(4), August 2003.
- [8] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard. Technical report, Argonne National Laboratory and Mississippi State University.
- [9] InfiniBand Trade Association. InfiniBand Architecture Specification. <http://www.infinibandta.com>.
- [10] Jonathan Stone and Craig Partridge. When the CRC and TCP checksum disagree. *SIGCOMM Comput. Commun. Rev.*, 30(4):309–319, 2000.
- [11] M. Koop, S. Sur, Q. Gao, and D. K. Panda. High Performance MPI Design using Unreliable Datagram for Ultra-Scale InfiniBand Clusters. In *21st ACM*

- International Conference on Supercomputing (ICS07)*, Seattle, WA, June 2007.
- [12] Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture. <http://www.nersc.gov/research/FTG/mvich/index.html>, August 2001.
- [13] Lawrence Livermore National Laboratory. MI&E Atlas (Peloton). <http://www.llnl.gov/computing/hpc/resources/>.
- [14] D. Mayhew and V. Krishnan. PCI Express and Advanced Switching: Evolutionary path to building next generation interconnects. *High Performance Interconnects, 2003. Proceedings. 11th Symposium on*, pages 21–29, 20–22 Aug. 2003.
- [15] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.
- [16] Network-Based Computing Laboratory. MVAPICH: MPI over InfiniBand and iWARP. <http://mvapich.cse.ohio-state.edu>.
- [17] OpenFabrics Alliance. OpenFabrics. <http://www.openfabrics.org/>.
- [18] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 55, New York, NY, USA, 1995. ACM.
- [19] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics network: high-performance clustering technology. *IEEE Micro*, 22(1):46–57, Jan/Feb 2002.
- [20] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale. NAMD: Biomolecular Simulation on Thousands of Processors. In *Supercomputing*, 2002.
- [21] S. J. Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117:1–19, 1995.
- [22] S. Reinemo, T. Skeie, T. Sodrings, O. Lysne, and O. Trudbakken. An overview of QoS capabilities in InfiniBand, Advanced Switching Interconnect, and ethernet. *Communications Magazine, IEEE*, 44(7):32–38, July 2006.
- [23] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.
- [24] S. Sur, A. Vishnu, H. W. Jin, W. Huang, and D. K. Panda. Can Memory-Less Network Adapters Benefit Next-Generation InfiniBand Systems? In *Hot Interconnect (HOTI 05)*, 2005.
- [25] Texas Advanced Computing Center. HPC Systems. <http://www.tacc.utexas.edu/resources/hpcsystems/>.