Designing High-Performance and Resilient Message Passing on InfiniBand

Matthew J. Koop[1] Pavel Shamis[2] Ishai Rabinovitz[2] Dhabaleswar K. (DK) Panda[3]
[1] *High Performance Technologies, Inc (HPTi), mkoop@hpti.com*
[2] *Mellanox Technologies, {pasha, ishai}@mellanox.co.il*
[3] *Dept. of Computer Science and Engineering, The Ohio State University, panda@cse.ohio-state.edu*

***Abstract*—Clusters featuring the InfiniBand interconnect are continuing to scale. As an example, the "Ranger" system at the Texas Advanced Computing Center (TACC) includes over 60,000 cores with nearly 4,000 InfiniBand ports. The latest Top500 list shows 30% of systems and over 50% of the top 100 are now using InfiniBand as the compute node interconnect. As these systems continue to scale, the Mean-Time-Between-Failure (MTBF) is reducing and additional resiliency must be provided to the important components of HPC systems, including the MPI library.**

In this paper we present a design that leverages the reliability semantics of InfiniBand, but provides a higher-level of resiliency. We are able to avoid aborting jobs in the case of network failures as well as failures on the endpoints in the InfiniBand Host Channel Adapters (HCA). We propose reliability designs for rendezvous designs using both Remote DMA (RDMA) read and write operations. We implement a prototype of our design and show that performance is near-identical to that of a non-resilient design. This shows that we can have both the performance and the network reliability needed for large-scale systems.

I. Introduction

Large-scale deployments of clusters designed from largely commodity-based components continue to be a major component of high-performance computing environments. A significant component of a high-performance cluster is the compute node interconnect. InfiniBand [1], is an interconnect of such systems that is enjoying wide success due to low latency (1.0-3.0μsec), high bandwidth and other features.

The Message Passing Interface (MPI) [2] is the dominant programming model for parallel scientific applications. As such, the MPI library design is crucial in the overall environment.

Clusters featuring the InfiniBand interconnect are continuing to scale. As an example, the "Ranger" system at the Texas Advanced Computing Center (TACC) includes over 60,000 cores with nearly 4,000 InfiniBand ports [3]. The latest list shows 30% of systems and over 50% of the top 100 are now using InfiniBand as the compute node interconnect.

Along with this increasing scale of commodity-based clusters is a concern with the Mean Time Between Failure (MTBF). As clusters continue to scale higher, it becomes inevitable that some failure will occur in the system and hardware and software need to be designed to recover from these failures. Providing resiliency within systems, middleware, and libraries is an area gaining a significant degree of interest [4], [5], [6]. One very important aspect of providing resiliency is to recover from network failures.

We seek to answer the following questions with this work:

- Can an MPI library be designed in a resilient manner for InfiniBand which can tolerate and recover from network faults including end-node Host Channel Adapter (HCA) failures?
- How much overhead will such a scheme incur? What will be the recovery cost for HCA failure?
- Can such a scheme maintain performance and scalability?

In this paper we design a light-weight reliability protocol for message passing software. As an example, we use the InfiniBand interconnect given its prominence in many HPC deployments. We explain how our design leverages network features to provide a high-performance yet resilient design. We define resiliency to mean the ability to recover from network failures including core switches and cables, but also failures on the end-node network devices themselves. We propose designs for both the eager and rendezvous protocols in MPI and support for both Remote Direct Memory Access (RDMA) read and put operations. We show that our proposed method has an insignificant overhead and is able to recover from many categories of network failures that may occur. We show that our design also uses limited memory for additional buffering of messages.

The remaining parts of the paper are organized as

This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342, #CCF-0702675, #CCF-0833169, #CCF-0916302 and #OCI-0926691; grant from Wright Center for Innovation #WCI04-010-OSU-0; and grant from Mellanox.

follows: In Section II we provide an overview of InfiniBand and then in Section III we describe the reliability semantics and error notification it provides. Our reliability design is presented in Section IV. Section V gives our experimental setup and Section VI is an evaluation and analysis of an implementation of our design. Work related to our design is presented in Section VII. Finally, conclusions and future work are presented in Section VIII.

II. InfiniBand Architecture

InfiniBand is a processor and I/O interconnect based on open standards [1]. It was conceived 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.

Communication in InfiniBand is accomplished using a queue based model. Sending and receiving end-points have to establish a Queue Pair (QP) which consists of Send Queue (SQ) and Receive Queue (RQ). Send and receive work requests (WR) are then placed onto these queues for processing by InfiniBand network stack. Completion of these operations is indicated by InfiniBand lower layers by placing completed requests in the Completion Queue (CQ).

There are four transport modes defined by the InfiniBand specification, and one additional transport that is available in the new Host Channel Adapters (HCAs) from Mellanox: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC), Unreliable Datagram (UD), and eXtended Reliable Connection. RD is not available with current hardware.

We will focus our efforts on the RC and XRC transports since they are traditionally the ones used for MPI and other applications over InfiniBand. Additional information on the reliability semantics of these transports is covered in the next section.

III. InfiniBand States and Reliability Semantics

In this section we first describe the Queue Pair (QP) states followed by a discussion on the reliability semantics of InfiniBand. Lastly, we describe how error states are communicated from the hardware.

A. Queue Pair States

Queue Pairs (QPs) are communication endpoints. In the case of RC, a single QP is a connection to another QP in another process. Each QP has its own state, each with different characteristics. Figure 1 shows the state diagram for each of these states:

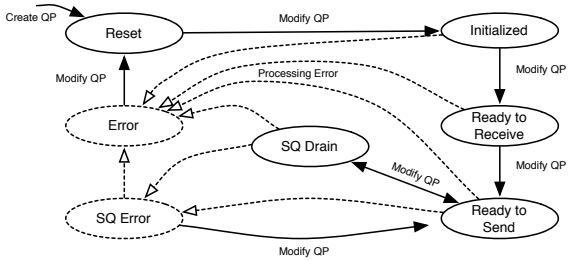


Figure 1. Queue Pair State Diagram

- *Reset*: This is the base state of a QP after creation. No work requests can be processed.
- *Init*: After moving from Reset, new receive requests can be issued, but no incoming or outgoing requests are satisfied.
- *Ready to Receive (RTR)*: In this mode a QP can process receive requests, but will not process any outgoing send requests.
- *Ready to Send (RTS)*: This is the working state of the QP. Incoming and outgoing requests are serviced.
- *Send Queue Drain (SQD)*: In this mode send operations will halt except for those that were already underway.
- *Send Queue Error (SQE)*: This state is entered if a completion error occurs when processing an outgoing send request. For Reliable Connection (RC) this state is not used and the QP will directly enter the "Error State" instead.
- *Error*: In this state the QP is "broken" and all requests that have errored out and not completed will be placed onto the CQ.

B. Reliability Semantics

The reliable transports of InfiniBand: Reliable Connection (RC), eXtended Reliable Connection (XRC) and Reliable Datagram (RD) provide certain reliability guarantees. As with other transports, there is both a variable CRC and end-to-end CRC that makes sure that data is not corrupted in transit across the network.

Further, the reliable transports of InfiniBand guarantee ordering and arrival of messages sent. When a message is placed into the Send Queue (SQ), it will not be placed into the Completion Queue (CQ) until the complete data has arrived at the remote HCA. Therefore, upon send completion the sender knows that the data has reached the remote node, but not that the data has arrived in the memory of the other node. This is an important aspect to note. It is not sufficient simply to wait for a send completion if the sender wants to make

sure that data has been placed into the end location. Errors on the HCA or over the PCIe bus may prevent data completion.

Further, the lack of a send completion on the sender does not imply that the receiver has not received the data. For example, the data may be received, however, before the hardware-level ACK message is sent there may become a network partition. In this case the sender QP will move to an error state with that request marked as incomplete.

Note that InfiniBand does provide an Automatic Path Migration (APM) feature that can automatically failover to another network path. In high availability system deployments the alternate path will typically go through separate FRUs (Field Replacable Units) such as cables, switch chips, switch enclosures in the Fabric. Typically an HCA has two ports and both ports are connected to separate fabrics or separate parts of the fabric. Using this approach, however, does not allow trying more than one alternate path, so if this one also fails then the QP will fall into an error state.

C. Error Notification

Errors in InfiniBand are provided both via asynchronous events as well as through the CQ. In many cases the errors will eventually be provided to the CQ, where they can be pulled out and reprocessed if needed. When an error occurs the QP moves into the Error state and must be moved to the Init stage to become usable again.

IV. Design

The goal of our design is to provide a *resilient* MPI for InfiniBand that can survive both general network errors as well as failure of even the HCA. Further, our goal is to keep the overhead of providing this resiliency at a minimum.

As noted in Section III, the InfiniBand network provides certain guarantees and error notifications when using reliable transports. These guarantees, however, are not enough for an MPI library since send completion does not guarantee the data has been received by the MPI library of the receiver.

A. Message Completion Acknowledgment

Since InfiniBand completions are not a strong guarantee of data transmission, we must provide message acknowledgments from within the MPI library. Unlike previous work that showed reliability for unreliable transports [7], there is no need for the MPI library to record timestamps and retransmit after a timeout. InfiniBand will signal an error to the upper-level if there has been a failure. With this level of hardware support, the acknowledgment design can have lower overhead. Thus, we only need to maintain the messages in memory until a software-level acknowledgment is received from the receiver.

MPI is generally implemented with two protocols: Eager and Rendezvous. Eager is generally used for smaller messages and can be sent to the receiver without checking if the corresponding receive has already been posted. Rendezvous is used when the communication must be synchronized or there are large messages. In the following paragraphs we describe our designs for these two protocols:

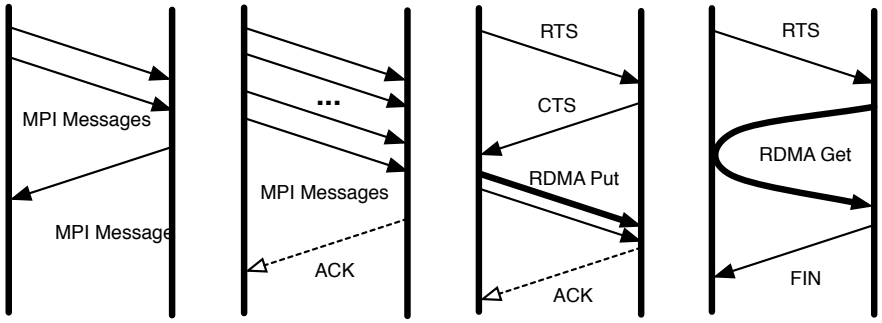
1) Eager Protocol: For the eager protocol we follow a traditional mechanism to signal message completion that is common in flow control protocols [8]. The acknowledgment can be piggybacked on the next message back to that process. As shown in Figure 2(a), this requires no additional messages. If the communication is not bi-directional, then after a preconfigured number of messages or data size an explicit acknowledgment, as shown in in Figure 2(b). Thus, very few additional messages are required.

It is important to note that traditional eager protocol implementations, including the one used in MVAPICH, make a copy of the data to be sent into a registered buffer. Thus, the send buffer is available immediately after the copy and the send operation can be marked complete according to the MPI specification.

In the case of our resilient design, since there are no timeouts the only additional cost in providing this support is to hold onto this temporary message buffer longer. As noted above, since the send operation is already marked as complete there is no additional time required for an eager send operation to complete from the MPI application point of view.

2) Rendezvous: When transferring long messages, the usual process of data transfer is to use a zero-copy protocol. These are generally classified into two categories: RPut and RGet. We examine reliability designs for each of these methods.

RPut: In this mode the sender notifies the receiver with a Request to Send (RTS) message and the receiver will respond with a Clear to Send (CTS) message that includes the address where the message should be placed. The sender can then perform an RDMA write operation to directly put the message into the memory of the receiver. Normally the sender can mark the send complete as soon as the RDMA write completion is



(a) Eager Piggyback (b) Eager ACK (c) RPUT (d) RGET

Figure 2. Reliability Protocols

placed in the CQ. With our strengthened semantics though, it cannot be freed until the receiver sends an explicit acknowledgment, as shown in Figure 2(c).

RGet: With this protocol the sender sends the source address in the RTS message. The receiver can then do an RDMA read to directly place the data from the source into the destination buffer. Then the receiver sends a finish message (FIN) to the sender. There is no need for an additional ACK in this case. Thus, there should be no additional overhead. In other words, our approach does not modify the original RGet protocol. (Figure 2(d))

B. Error Recovery

There are two main forms of errors that can occur to the MPI library. These can be classified as Network Failures or Fatal Events. Figure 3 shows the basic recovery flow that is attempted in our design.

1) Network Failure: In this case the error is internal to the network. It may come in the form of a "Retry Exceeded" error, which can denote either a cable or switch failure or even severe congestion. In this case we should be able to retry the same network path again since it may be a transient issue.

To attempt to reconnect, each side will attempt to send another out-of-band message to the other side. In our prototype, this is done using the Unreliable Datagram (UD) transport where QPs do not fall into the error state due to network errors. If the remote peer replies then the QPs can be repaired and the messages that have not arrived can be re-transmitted. If the remote peer does not respond it could be an internal network failure so we can attempt different paths within the network, with timeouts for each. After a pre-configured length of time the job could be configured to fail.

2) Fatal Event: If we receive an asynchronous event from the InfiniBand library of a "Fatal Event" from the HCA meaning it is not able to continue, we attempt to reload the HCA driver. In this case we must unload all InfiniBand resources (QPs, unpin memory). The resources must be freed in order to allow the driver to reset the device and restart itself. After reloading we must recreate most of these resources. Some resources, such as QPs, will be re-created once communication restarts by the connection manager. If the same HCA is not available after a driver reset the library should move connections over to another HCA, if available.

We will then attempt to re-connect with the remote peer. The remote peer will have to reload the connections and memory, but this can be conveyed with control messages.

Note that the sender will interpret a fatal event on the receiver as a Network Failure, so the receiver must reconnect to the sender. In the case of fatal events on both the sender and receiver, reconnect requests must be sent through the administrative network since no InfiniBand resources will be available for either side to reconnect.

V. Experimental Setup

Our experimental test bed is a 576-core InfiniBand Linux cluster. Each of the 72 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 [9], OFED 1.3 release.

We have implemented a prototype of our design over the verbs interface of the OpenFabrics stack [9]. Our prototype design is designed within the MVAPICH MPI library [10]. Our design uses only InfiniBand primitives

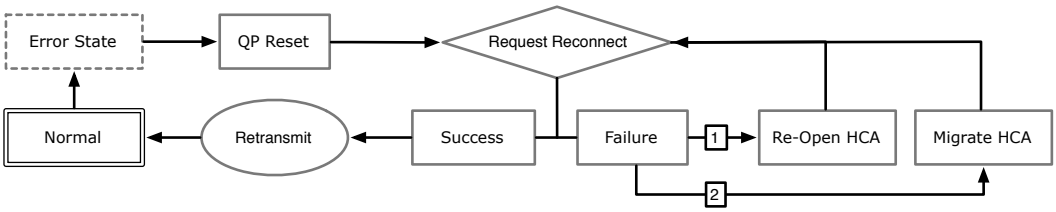


Figure 3. Recovery Flow Chart: Upon failure, a reconnect request will take place out-of-band. If this fails or if a fatal HCA event was received then attempt to reopen the HCA or switch to another HCA.

for re-connection and can recover from network failures and fatal events.

We have evaluated our prototype by using both specialized firmware to introduce errors, defective switches and directly moving QPs to error states. We observed that the prototype was able to reconnect and withstand these errors.

VI. Experimental Evaluation

In this section we describe the evaluation of our prototype design. We first present an evaluation on both microbenchmarks and application kernels. Then we examine the cost of recovering from an error using this design.

A. Methodology

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

- **RPut-Existing** (RPut): This configuration is the base MVAPICH 1.1 version with the RPut rendezvous protocol.
- **RGet-Existing** (RGet): Same as RPut-Existing, but using the RGet protocol.
- **RPut-Resilient** (RPut-Rel): In this configuration we use the modified MVAPICH version that uses the design from Section IV with the RPut protocol.
- **RGet-Resilient** (RGet-Rel): This configuration is the same as Resilient-RPut, but uses the RGet protocol instead.

B. Microbenchmark Performance

We performed an evaluation of base-level microbenchmarks to determine if our resiliency design incurs any overhead. We found that for all standard microbenchmarks including latency, bandwidth and bi-directional bandwidth there is no overhead. Results for latency and bandwidth are shown in Figure 4.

C. Application Benchmarks

In this section we evaluate the performance of all of our configurations on the NAS Parallel Benchmark suite to expose any additional overhead.

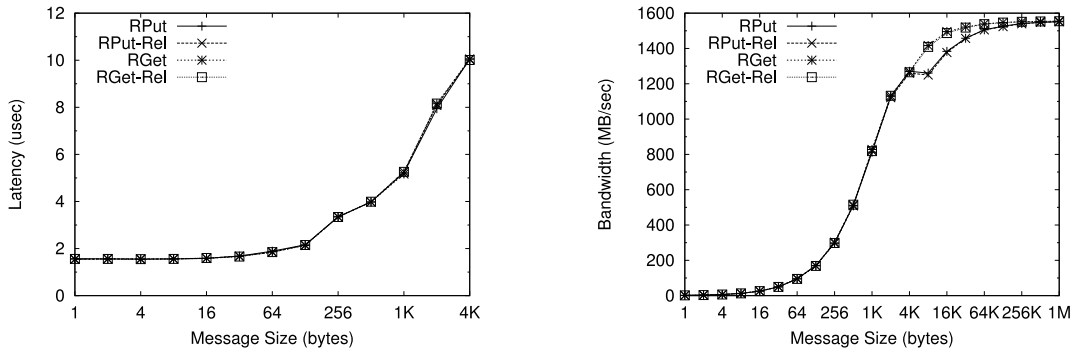
Figure 5 shows the results of our evaluation of each of the NAS benchmarks. All of these results are with 256 processes on classes B and C, as indicated on the graph. The results show very low overhead, with nearly unnoticeable overhead in the RGet-Rel cases. The RPut-Rel configuration, however, has higher overhead. Recall from Section IV, that a reliable form of RPut requires an additional acknowledgment message. For SP.C, which has many large messages, this overhead is most apparent. The overhead reaches 4% for RPut-Rel with only a 1.5% overhead for RGet-Rel. From these results we can determine that the RGet-Rel configuration should be the default configuration.

A different potential overhead of our design is the requirement to buffer messages until they are acknowledged. Large messages will take the rendezvous path and will not be buffered, so only smaller messages will take this path. Table I shows the memory per process on average required for buffering in both the normal case as well as our design. From the table we can observe that no process requires more than 0.45 MB of memory for the additional reliability semantics.

D. Recovery Cost

In this initial evaluation we determine the overhead to detect failures and reconnect required connections. For this evaluation we add a series of transitions to the *Error* state within a microbenchmark between two nodes and track the amount of time from setting the failure to response and reconnection.

Our evaluations show that a single reconnection due to an error takes on average 0.335 seconds. This is faster than could be achieved using a standard multi-second timeout within the MPI library. Our approach uses the hardware based timeouts from the InfiniBand device to determine if there is a connection failure.



(a) Latency

(b) Bandwidth

Figure 4. Microbenchmark Results: No added overhead for reliability

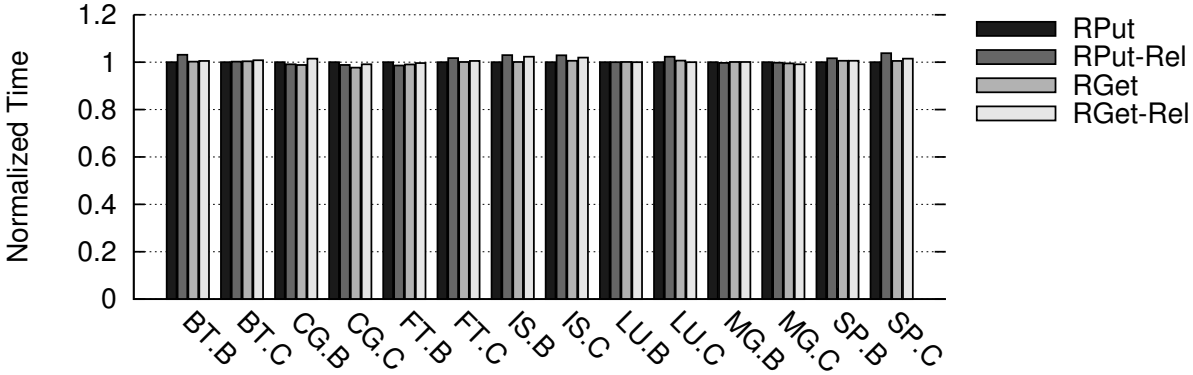


Figure 5. NAS Parallel Benchmarks Performance (256 Processes)

Class	BT.C	CG.C	FT.C	IS.C	LU.C	MG.C	SP.C
Reliable	3.46 MB	6.92 MB	0.07 MB	0.01 MB	1.65 MB	0.16 MB	5.33 MB
Normal	3.20 MB	6.48 MB	0.07 MB	0.01 MB	1.53 MB	0.13 MB	4.95 MB
Difference	0.26 MB	0.44 MB	0.00 MB	0.00 MB	0.12 MB	0.03 MB	0.38 MB

Table I
COMPARISON OF AVERAGE MAXIMUM MEMORY USAGE PER PROCESS FOR MESSAGE BUFFERING

VII. RELATED WORK

Other researchers have explored providing reliability at the MPI layer as part of the LA-MPI project [11], [12]. 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 within the MPI library with a several second timeout, focusing on providing checksum acknowledgment support in the MPI library. Saltzer et al. [13] discusses the need for end-to-end reliability instead of a layered design.

The authors of this work have previous work [7], [14] providing reliability over the unreliable transports of InfiniBand, including Unreliable Connection and Unreliable Datagram. These works have not ever considered HCA failure or attempting to leverage the reliability from InfiniBand.

Other MPI designs have included support for failover

in the past. Vishnu, et. al., showed how to provide failover for uDAPL to other networks if there was a network partition [15]. Again, this work did not address the issue of HCA failure and relied only on the network guarantees of the underlying InfiniBand hardware. Open MPI [16] also provides support for network failover using their Data Reliability component, but this component can occur a very large overhead on performance since it does not take into account any reliability provided by the hardware and is similar to the support in LA-MPI.

VIII. Conclusion

InfiniBand is becoming increasingly common in cluster environments. In existing MPI designs for InfiniBand, some network-level errors will result in a crash of the end application.

In this work we have designed a resilient MPI for InfiniBand that can withstand network failures as well as HCA failures. To the best of our knowledge, this is the first MPI design that contains all of these features for InfiniBand. We have evaluated our design on 256 cores and the performance between the non-resilient design and resilient designs is near identical. We have also shown that the amount of memory required for buffering messages is also very low (less than 1 MB). Our design allows for increased network resiliency with very low overhead.

In the future we plan to look into additional failure resiliency techniques through querying of the network subnet manager to determine the causes of network failures and to pick different paths.

Software Distribution: The software described in this paper has been incorporated into the MVAPICH 1.2 release and is available for download from MVAPICH web page [10].

References

[1] InfiniBand Trade Association, "InfiniBand Architecture Specification," http://www.infinibandta.com.

[2] *MPI: A Message-Passing Interface Standard*, Message Passing Interface Forum, Mar 1994.

[3] Texas Advanced Computing Center, "HPC Systems," http://www.tacc.utexas.edu/resources/hpcsystems/.

[4] Los Alamos, "National HPC Workshop on Resilience," http://institute.lanl.gov/resilience/conferences/2009/.

[5] Stephen L. Scott and Box Leangsuksun, "Workshop on HPC Resiliency," http://www.lanl.gov/conferences/lacss/2009/.

[6] "Resilience '09: Proceedings of the 2009 workshop on resiliency in high performance," 2009, stephen L. Scott and Chokchai (Box) Leangsuksun and Christian Engelmann.

[7] 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.

[8] J. Liu and D. K. Panda, "Implementing Efficient and Scalable Flow Control Schemes in MPI over InfiniBand," in *Workshop on Communication Architecture for Clusters (CAC 04)*, April 2004.

[9] OpenFabrics Alliance, "OpenFabrics," http://www.openfabrics.org/.

[10] Network-Based Computing Laboratory, "MVAPICH: MPI over InfiniBand and iWARP," http://mvapich.cse.ohio-state.edu.

[11] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi, "A Network-Failure-Tolerant Message-Passing System for Terascale Clusters," *International Journal of Parallel Programming*, vol. 31(4), August 2003.

[12] 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.

[13] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, 1984.

[14] M. Koop, R. Kumar, and D. K. Panda, "Can Software Reliability Outperform Hardware Reliability on High Performance Interconnects? A Case Study with MPI over InfiniBand," in *22nd ACM International Conference on Supercomputing (ICS08)*, June 2008.

[15] A. Vishnu, P. Gupta, A. Mamidala, and D. K. Panda, "A Software Based Approach for Providing Network Fault Tolerance in Clusters Using the uDAPL Interface: MPI Level Design and Performance Evaluation," in *Proceedings of SuperComputing*, November 2006.

[16] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.