# Automatic Path Migration over InfiniBand: Early Experiences

ABHINAV VISHNU, AMITH R. MAMIDALA AND DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-5/06-TR56

# Automatic Path Migration over InfiniBand: Early Experiences [⋆]

Abhinav Vishnu, Amith R. Mamidala, and Dhabaleswar K. Panda

Network Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{vishnu, mamidala, panda }@cse.ohio-state.edu

**Abstract.** High computational power of commodity PCs combined with the emergence of low latency and high bandwidth interconnects has led to the trends in cluster computing. InfiniBand architecture has been proposed as the next generation interconnect for inter-process communication and I/O communication. Clusters with InfiniBand are being deployed, as reflected in the TOP 500 Supercomputer rankings. However, increasing scale of these clusters has reduced the MTBF (Mean Time Between Failures) of components. Network component is one such component of clusters, where failures of NICs, cables or switches breaks existing paths of communication. InfiniBand provides a hardware mechanism, *Automatic Path Migration (APM)*, which allows user transparent detection and recovery from network faults, without application restart. In this paper, we design modules *alternate path specification* module, *path loading request module* and *path migration module*, which act as the workhorse for providing network fault tolerance for user level applications. We interface these modules for simple micro-benchmarks at the Verbs Layer, the user access layer for InfiniBand, and study the impact of different state transitions associated with APM. In addition, we integrate these modules with MPI layer, to provide network fault tolerance for MPI Applications. Our performance evaluation at the MPI Layer shows that APM incurs negligible overhead in the absence of faults in the system. In the presence of network faults, APM incurs negligble overhead for reasonably long running applications. For Class B, FT and LU NAS Parallel Benchmarks with 8 processes, the degradation is around 5-7% in the presence of network faults. To the best of our knowledge, this is the first design of MPI with APM and its study at the Application Level.

## 1 Introduction

Introduction of high speed RDMA-enabled interconnects like InfiniBand, Myrinet, Quadrics, RDMA-enabled Ethernet has escalated the trends in cluster computing. InfiniBand in particular is being widely accepted as the next generation interconnect due to its open standard and high performance. As a result, clusters based on InfiniBand are becoming popular, as shown by the TOP 500 Supercomputer rankings. However, increasing scale of these clusters has reduced the MTBF (Mean Time Between Failures) of components. Network component is one such component of clusters, where failures of NICs, cables or switches breaks the existing

[⋆] This research is supported in part by Department of Energy's grant #DE-FC02-01ER25506; National Science Foundation's grants #CCR-0204429 and #CCR-0311542; grants from Intel and Mellanox; and equipment donations from Intel, Mellanox, AMD, Apple, Advanced Clustering, Dell and Sun Microsystems.

paths of communication. InfiniBand provides a hardware mechanism, *Automatic Path Migration (APM)*, which allows user transparent recovery from network faults. However, the current InfiniBand literature lack the understanding of APM mechanism intricacies, associated designs and performance evaluation. In this paper, we take on these challenges. We design modules *alternate path specification module*, *path loading request module* and *path migration module*, which act as the workhorse for providing network fault tolerance for user level applications. We interface these modules for simple micro-benchmarks at the Verbs Layer, the user access layer for InfiniBand, and study the impact of different state transitions associated with APM. We also integrate these modules at MPI layer to provide network fault tolerance for MPI applications. Our performance evaluation at the MPI Layer shows that APM incurs negligible overhead in the absence of faults in the system. In the presence of network faults, APM incurs negligible overhead for reasonably long running applications. For Class B, FT and LU NAS Parallel Benchmarks with 8 processes, the degradation is around 5-7% in the presence of network faults. To the best of our knowledge, this is the first design of MPI with APM and its study at the application level.

The rest of the paper is organized as follows. In section 2, we provide background of our work. In section 3, we present our APM modules; *alternate path specification* module, *path loading request module* and *path migration module*, which constitute the core of the APM based solution. We also present the integration of these modules for verbs level tests and MPI Layer. In section 4, we present the performance evaluation at the verbs level tests followed by performance evaluation for MPI applications. In section 5, we present the related work. In section 6, we conclude and present our future directions.

## 2 Background

In this section, we provide background information for our work. First, we provide a brief introduction of InfiniBand. This is followed by a detailed discussion of APM feature in InfiniBand.

### 2.1 Overview of Infi niBand and QP Transition States

The InfiniBand Architecture (IBA) [2] defines a switched network fabric for interconnecting processing nodes and I/O nodes. The InfiniBand communication stack consists of different layers. The interface presented by Host Channel Adapter (HCA) (InfiniBand interconnect) to consumers belongs to the transport layer. A Queue-Pair (QP) based model is used in this interface. A *Queue Pair* in InfiniBand Architecture consists of two queues: a *Send Queue* and a *Receive Queue*. Figure 1 shows the communication state transition sequence for a QP. At the point of QP creation, its communication state is RESET. From this state it is brought to the INIT state by modify_qp call. During RESET-INIT transition, it is specified with the HCA port to use in addition to the atomic flags. Once in the INIT state, the QP is specified with the destination LID and the destination QP from which it will receive the messages. A modify_qp call brings it to READY-TO-RCV (RTR) state. Finally, QP is brought to READY-TO-SEND (RTS) state by specifying associated parameters and making the modify_qp call. At this point, the QP is ready to send or receive data from its destination QP. Should any errors occur on the QP, the QP goes to the ERROR state automatically. At this state, the QP is broken and cannot communicate with its destination QP. In order to re-use this QP, it needs to be brought back to the RESET state and the above-mentioned transition sequence needs to be followed.

Requests for sending and receiving data are done by posting the send and receive descriptor respectively. Completion of the work requests is done by polling on the completion queue. The receive CQE and send CQE correspond to the opcode for completion entries received from the CQ for receive and send descriptors.

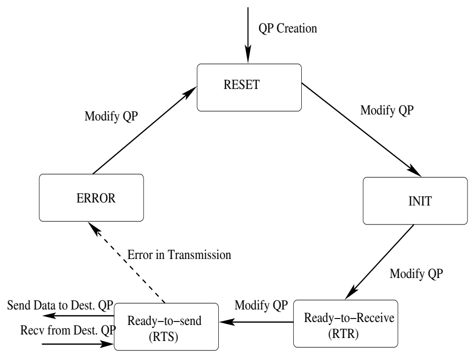

**Fig. 1.** QP Communication State Diagram

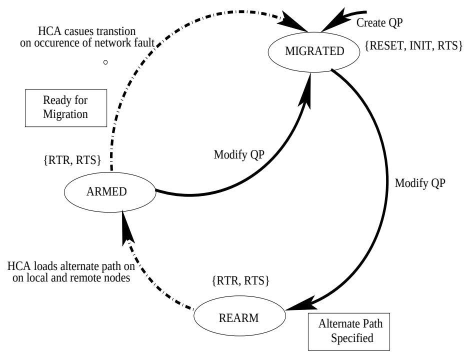

**Fig. 2.** QP Path Migration State Diagram

### 2.2 Overview of Automatic Path Migration

APM is a feature provided by InfiniBand which enables recovery from network faults by using the alternate path specified by the user. Automatic path migration is available for Reliable Connected (RC) and Unreliable Connected (UC) QP Service Type. In this paper, we have used RC QP service type. For this feature, InfiniBand specifies *Path Migration States* associated with a QP. It is to be noted that path migration state is different from the communication state of a QP. A valid combination of communication and path migration states are possible. This is shown further in Figure 2. During the creation of QP, the initial path migration state for a QP is set to MIGRATED. At this point, the QP can be in RESET, INIT or RTS communication state. Please note that once the transition sequence is complete, the QP's path migration state goes back to migration state. Hence, this state is valid for QPs, which have their communication state as RTS. APM defines a concept of alternate path, which is used as an escape route should an error occurs on the primary path of communication. The alternate path is specified by the user. This specification of the alternate path can be done at any point in communication beginning INIT-RTR communication state of the QP. Once this has been specified, the HCA can be requested to begin loading this path. This is done by specifying the QP's path migration state to be REARM. At this point, the request is initiated, however this request is non-blocking, and the notification for loading the alternate path is done with asynchronous events. Once the loading of the path is done, the path migration state of a QP is ARMED. During this state, the alternate path can be switched over to function as a primary path. This can be done by HCA automatically, should an error occur on the primary path of communication. This is shown with dotted line in Figure 2. In addition, a user can manually request the alternate path to be used as the primary path of communication. This is shown with solid line in Figure 2. This request is also non-blocking and asynchronous event(s) are generated to notify the completion of the request(s).

## 3 Network Fault Tolerance Modules

In this section, we present the modules which form the core in providing network fault tolerance for our design. There are three modules which work in conjunction namely *Alternate Path Specification*, *Path Loading Request Module* and *Path Migration Module*. The alternate path specification module is responsible for deciding the alternate path to be used in presence of network fault. The path loading request module is responsible for requesting the alternate path to be loaded in the path migration state machine. The path migration module is responsible for transition of alternate path to the primary path of communication. We decoupled these modules with each other, since each of the modules can be requested separately by MPI Library. We begin with the design of alternate path specification module.

### 3.1 Alternate Path Specification Module

This module is responsible for specifying an alternate path to a queue pair. There are various parameters associated with specification of an alternate path. *alt_dlid* specifies the destination LID of the alternate path. *alt_port* is used for specifying the alternate port. The destination queue pair number remains the same for the alternate path specification, as per the IBA specification. In this paper, we have used dual-port InfiniBand HCAs for evaluation. The first port is used for primary path of communication and the second port is specified in the alternate path of the communication.

### 3.2 Path Loading Request Module

This module is responsible for initiating the loading of alternate path in a QP. It takes the list of processe(s) for which the initiating needs to be done. This module can be invoked during anytime of the program execution after the INIT communication state of the corresponding QP. The completion of the request(s) is notified by asynchronous event(s), which we refer to as the ARMED event(s) in this paper. In the performance evaluation section, the invoking of this module simultaneously with alternate oath specification module is referred by migrated-armed legend.

### 3.3 Path Migration Module

This module is invoked when a user wants to manually move the alternate path to be used as a primary path of communication. This functionality is useful in the presence of congestion/performance drop on the available bandwidth on the primary path. Alternatively, if an error occurs during transmission, the HCA requests the alternate path to be loaded as the primary path of communication, without intervention from the user application. This module assumes that the path loading request module has successfully loaded the alternate path, and the alternate path is in a healthy state. The completion of this sequence is done with the help of asynchronous events, which are referred as MIGRATED events in this paper. In the performance evaluation section, the invoking of this module is referred by armed-migrated legend.

### 3.4 Interfacing with the Verbs and the MPI Layer

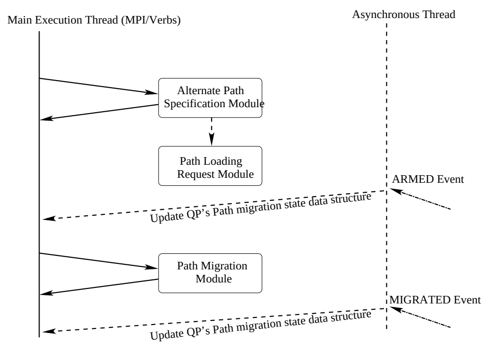
Main Execution Thread (MPI/Verbs)
Asynchronous Thread
Alternate Path Specification Module
Path Loading Request Module
ARMED Event
Update QP's Path migration state data structure
Path Migration Module
MIGRATED Event
Update QP's Path migration state data structure

**Fig. 3.** Interaction of Network Fault Tolerance Modules with Main Execution Thread and Asynchronous Thread

In this section, we discuss the interfacing of our network modules with the Verbs and the MPI Layer. In order to handle the MIGRATED and ARMED events, we create an asynchronous thread. This thread is in the sleep state, and wakes up only on the occurrence of events generated by the HCA. The interaction of the main execution thread and the asynchronous thread is shown further in Figure 3. Since the path migration module assumes that the alternate path has been loaded, we maintain data structures shared by the asynchronous thread and the main execution thread. The asynchronous thread updates these data structures on the occurrence of ARMED events. Using these data structures, the main thread of execution can ascertain the path migration status of its QPs. For MPI evaluation, we use MVAPICH [4], which supports multiple ports, multiple HCAs and combinations. To understand the impact on large scale clusters, we create multiple QPs between each pair process. Each of these QPs are used in a round robin fashion.

## 4 Performance Evaluation

In this section, we evaluate the performance of our Network Fault Tolerant design modules over InfiniBand. At the Verbs layer, we design a ping-pong latency test and a computation test. We study the impact of different transition states in APM, when they are requested at different points in the execution of the test. This is followed by the study at the MPI applications and the impact of these states on execution time, in the absence and presence of faults. We begin with a brief overview of our experimental testbed.

### 4.1 Experimental Testbed

Our Experimental Testbed consists of four IA32 nodes each having 133 MHz PCI-X slot. Each node has two Intel Xeon CPUs running at 2.4 GHz processors, 512 KB L2 cache and 1 GB of main memory. This cluster uses II Generation MT23218 4X Dual Port HCAs from Mellanox. The kernel version we used is Linux 2.6.9-15.EL. The IBGD version is 1.8.2 and HCA firmware version is 3.3.2.

### 4.2 Evaluation of the Network Fault Tolerance Modules at the Verbs Layer

In this section, we evaluate the performance of the network fault tolerance modules at the verbs layer. To study this performance, we design a ping-pong latency test. The test is built over the concept of two processes: sender and receiver. The sender sends posts a send descriptor and polls on the completion queue for receiving a receive and a send CQE each. The receiver polls on completion queue for recv CQE. Once a recv CQE is obtained, the receiver sends a reply back to the sender. This step is reported for a large number of iterations. The sender reports the latency as the half of the total time for above operation. To understand the impact on a large scale cluster, we create multiple QP connections between these processes. These QPs are used in a round-robin fashion for communication. The legend corresponding to original is the case when none of the network fault tolerance modules are invoked at all.

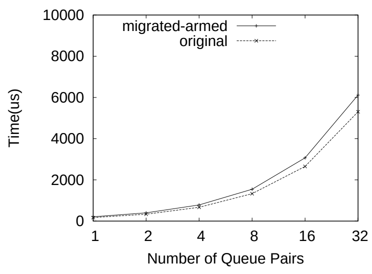

**Fig. 4.** Transition From INIT-RTS, Small Number of QPs

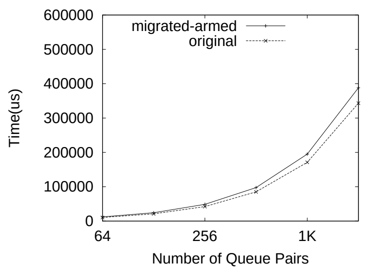

**Fig. 5.** Transition From INIT-RTS, Large Number of QPs

In Figure 4 and Figure 5, we present the time consumed in INIT-RTS stage, when alternate path specification module and Path loading request module are invoked during RTR-RTS phase. We notice that the total time taken by each of the lines is linear with increasing number of queue pairs. An increased time in execution by around 15% is noticed. For all the remaining tests, we assume that port 1 is the primary path of communication and port 2 is the alternate path of communication for all QPs.

In Figure 6 and Figure 7, we present the time consumed in Migrated-Armed and Armed-Migrated stages with increasing number of QP connections between the processes. To calculate the timings for Migrated-Armed transition, the alternate path specification module is invoked

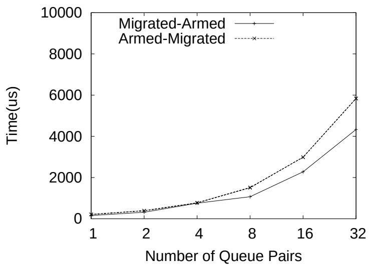

**Fig. 6.** Transition for different transition states in APM, Small Number of QPs

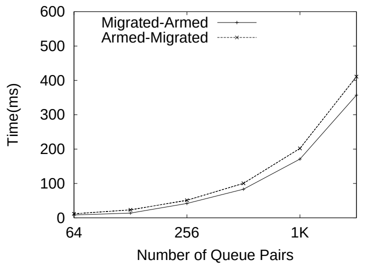

**Fig. 7.** Timings for different transition states in APM, Large Number of QPs

during INIT-RTR phase and time is calculated till ARMED events for all QPs are received by the asynchronous thread. For calculating the Armed-Migrated transition timings, path migration module is invoked for all QPs. Once the asynchronous thread receives MIGRATED event for all QPs, shared data structure between main thread and asynchronous thread is updated along-with the time-stamp of updation. A linear trend is observed with the increasing number of QPs in these transitions as shown by the figures. For small number of QPs, armed-migrated transition takes around 30% more time than migrated-armed transition. For larger number of QPs, the time reduces to around 16%. However, since these requests are asynchronous in nature, it remains to be seen, how these transitions impact the ongoing communication.

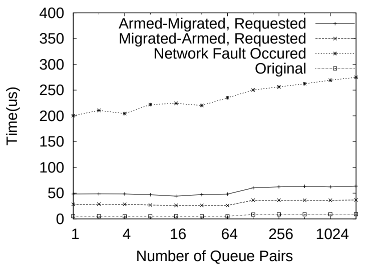

**Fig. 8.** Impact on Latency for 128 Byte Message with Increasing Number of QPs

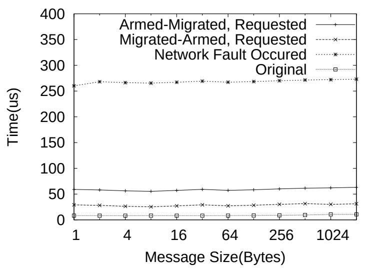

**Fig. 9.** Impact on Latency for Small Messages using 512 QPs

Figure 8 compares the performance of the original case with different transition sequences. To see the impact on latency, we initiate the transition sequence for the QP which is used in current iteration. Since QPs are used in a round robin fashion, this step is performed till request has been done for all QPs. We measure the latency till all events for the corresponding transition sequence have been generated. The graph shows the average latency observed for those iterations. The results show that both Migrated-Armed and Armed-Migrated requests add significant overhead to ongoing communication. We now show the results for acid test, the impact of performance on latency, when network fault occurs. After the alternate path is loaded, we disable the primary path of communication by manually plugging out the cable corresponding to primary path of communication. The HCA automatically moves the alternate path of the current QP to be used as a primary path of communication. Since QPs are used in a round robin fashion, the HCA does it for every QP. We measure the average latency observed till all MIGRATED events for all QPs has been generated. This test helps us understand that for large scale clusters, the impact on latency for small message, each process pair using one QP for communication. Figure 9 shows the impact on latency for small messages, when 512 QPs are used for communication. We notice that the amount of overhead remains almost same with increasing message size. Hence, the overhead paid per QP remains same independent of the message size.

Figures 10 and 11 show the impact on computation for different APM transition sequences. Since the ARMED and MIGRATED events are handled by asynchronous thread, we run both the processes on the same node. The test initiates APM transition request (migrate-armed/armed-migrated) for all QPs and performs computation. We have chosen a minimum computation loop, such that all interrupts are generated before the computation loop finishes. We subtract the time spent in requesting the transition and report the performance. We notice that with increasing number of QPs, the amount of computation time increases significantly. However, with the increasing computation loop, the overhead incurred remains same for the same number of QPs. For 2048 QPs, the overhead incurred is almost 60 ms.

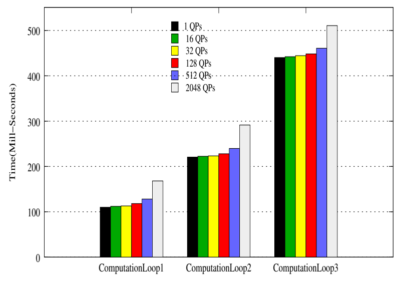

**Fig. 10.** Impact on Computation, Migrated-Armed Requested During Computation

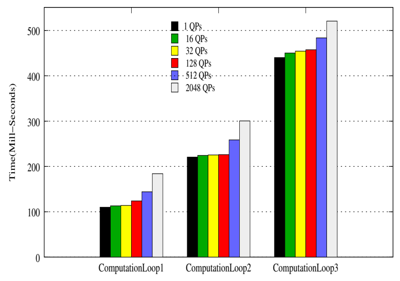

**Fig. 11.** Impact on Computation, Armed-Migrated Requested During Computation

### 4.3 Evaluation of the Network Fault Tolerance Modules at the MPI Layer

In this section, we present the performance for NAS parallel benchmarks, when APM sequence transitions are requested. We also study the impact when network fault occurs in the system. We use a 4x2 configuration (4 nodes and 2 processes per node). Before taking these results, we profiled applications to make sure that network fault tolerance modules are invoked during the critical execution of the application. We break the primary communication path by unplugging the cable at different points in the application execution for sixteen runs and present the average performance we have observed. At this point, we do not have a very systematic methodology for network fault injection. In future, we plan to design software based error injection mechanism.

Figure 12 and Figure 13 show the results with different transition states in the absence of faults, and compare its performance to the case when faults occur for IS, Class A and Class B respectively. Without the occurence of the fault, there is some increase in timing for IS Class A. However, the execution time of IS Class B is higher than of Class A, and as a result, the impact is unnoticeable for IS Class B benchmark. In the presence of network faults, with the increasing number of QPs and , the execution time increases sharply, since more time is spent in moving the alternate path to primary path. This step needs to be done with all QPs, since they are used in a round robin fashion for communication. Clearly, for application running for fraction of a minute, APM does not provide any benefit.

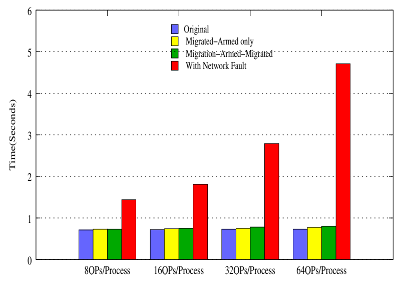

**Fig. 12.** Performance Evaluation on IS, Class A, 4x2 Configuration

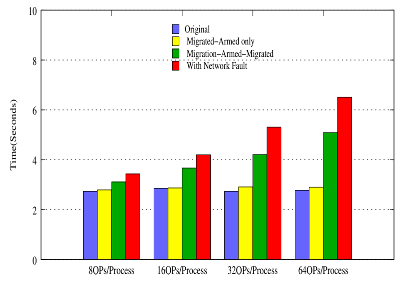

**Fig. 13.** Performance Evaluation on IS, Class B, 4x2 Configuration

Figure 14 and Figure 15 show the results for NAS FT Class B and LU Class B respectively. Since the overhead paid per QP almost remains same, when a network fault occurs, we notice that the percentage of performance degradation is much lesser in these cases. We notice that even with increasing the number of QPs/process to 64, we only notice around 5-6% degradation in performance. For LU class B in particular, the execution time is around 256 seconds, and hence the overhead of state transitions is amortized with the long running application. Hence for applications running for reasonably long time, APM incurs almost negligible overhead in the overall execution time.

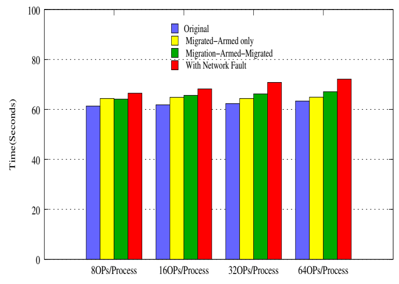

**Fig. 14.** Performance Evaluation on FT, Class B, 4x2 Configuration

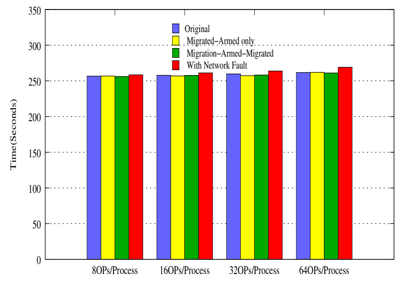

**Fig. 15.** Performance Evaluation on LU, Class B, 4x2 Configuration

## 5 Related Work

A couple of researchers have focused on designing MPI for providing network fault tolerance. In our previous work [3, 7], we have focused on combining multiple IBA HCAs. However, the design does not support network fault tolerance. LA-MPI [5] is an MPI implementation developed at Los Alamos National Labs. LA-MPI was designed with the ability to stripe message across several network paths. OpenMPI [1], also provides striping across multiple interconnects. It is capable of striping messages across a combination of interconnects, IBA, Myrinet and Ethernet. It also supports network failover. Pakin et. al. has also proposed VMI [6], which provides support for multiple interconnects and failover. However, above works are software based solutions and none of the above works have focused on providing network fault tolerance at MPI layer using InfiniBand's APM hardware mechanism.

## 6 Conclusions and Future Work

Increasing scale of clusters has reduced the MTBF (Mean Time Between Failures) of associated components. Network component is one such component of clusters, where failures of NICs, cables and switches breaks existing paths of communication. InfiniBand provides a hardware mechanism, *Automatic Path Migration* (APM), which allows user transparent recovery from network faults. However, the current InfiniBand literature lacks the understanding of APM mechanism intricacies, associated designs and performance evaluation. In this paper, we have designed modules *alternate path specification* module, *path loading request module* and *path migration module*, which act as the workhorse for providing network fault tolerance with APM for user level applications. We have interfaced these modules for simple micro-benchmarks at the Verbs Layer, the user access layer for InfiniBand, and studied the impact of different state transitions associated with APM. We integrate these modules with the MPI layer to provide network fault tolerance for MPI Applications. We have also discussed the need of interfacing alternate path specification module with the *subnet manager*, to provide the path with least errors. Our performance evaluation has shown that APM incurs negligible overhead in the absence of faults in the system. For MPI applications executing for reasonably long time, APM causes negligible overhead in the presence of network faults. For Class B, FT and LU NAS Parallel Benchmarks with 8 processes, the degradation is around 5-7% in the presence of network faults.

In future, we plan to study the impact of our design for large scale clusters at the application level. We plan to design software based error injection mechanism and study the impact. One of the limitations of APM is the requirement of the alternate path to be in healthy state. We plan to design solutions which overcome this limitation. We also plan to use traces for fault generation and compare the performance of hardware and software based solutions.

## References

1. Edgar Gabriel and Graham E. Fagg and George Bosilca and Thara Angskun and Jack Dongarra and Jeffrey M. Squyres and Vishal Sahay and Prabhanjan Kambadur and Brian Barrett and Andrew Lumsdaine and Ralph H. Castain and David J. Daniel and Richard L. Graham and Timothy S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *EuroPVM/MPI*, pages 97–104, 2004.
2. InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2, October 2004.
3. J. Liu, A. Vishnu, and D. K. Panda. Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation. In *SuperComputing Conference*, 2004.
4. Network-Based Computing Laboratory. MVAPICH/MVAPICH2: MPI-1/MPI-2 for InfiniBand on VAPI/Gen2 Layer. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.
5. Richard L. Graham and Sung-Eun Choi and David J. Daniel and Nehal N. Desai and Ronald G. Minnich and Craig E. Rasmussen and L. Dean Risinger and Mitchel W. Sukalski. A Network-Failure-Tolerant Message-Passing System for Terascale Clusters. volume 31, pages 285–303, Norwell, MA, USA, 2003. Kluwer Academic Publishers.
6. Scott Pakin and Avneesh Pant. VMI 2.0: A Dynamically Reconfigurable Messaging Layer for Availability, Usability, and Management. In *The 8th International Symposium on High Performance Computer Architecture (HPCA-8), Workshop on Novel Uses of System Area Networks (SAN-1)*, Cambridge, Massachusetts, February 2, 2002. Available from http://www.csl.cornell.edu/SAN-1/san1.pdf.

7. A. Vishnu, G. Santhanaraman, W. Huang, H.-W. Jin, and D. K. Panda. Supporting MPI-2 One Sided Communication on Multi-Rail InfiniBand Clusters: Design Challenges and Performance Benefits. In *International Conference on High Performance Computing, HiPC*, 2005.