Automatic Path Migration over InfiniBand: Early Experiences *

Abhinav Vishnu Amith R. Mamidala Sundeep Narravula Dhabaleswar K. Panda
Network Based Computing Lab
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{vishnu, mamidala, narravul, panda}@cse.ohio-state.edu

Abstract

High computational power of commodity PCs combined with the emergence of low latency and high bandwidth interconnects has escalated the trends of cluster computing. Clusters with InfiniBand are being deployed, as reflected in the TOP 500 Supercomputer rankings. However, increasing scale of these clusters has reduced the Mean Time Between Failures (MTBF) of components. Network component is one such component of clusters, where failure of Network Interface Cards (NICs), cables and/or switches breaks existing path(s) of communication. InfiniBand provides a hardware mechanism, Automatic Path Migration (APM)*, which allows user transparent detection and recovery from network fault(s), without application restart. In this paper, we design a set of modules; which work together for providing network fault tolerance for user level applications leveraging the APM feature. 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.*

Keywords: InfiniBand, APM, MTBF, MPI, Verbs

1 Introduction

Introduction of high speed RDMA-enabled interconnects like InfiniBand [9], Myrinet, Quadrics, RDMA-enabled Ethernet has escalated the trends of 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 increasingly popular, as shown by the TOP 500 [3] Supercomputer rankings. However, increasing scale of these clusters has reduced the Mean Time Between Failures (MTBF) of components. Network component is one such component of clusters, where failures of network interface cards (NICs), cables or switches breaks the existing path(s) of communication. InfiniBand provides a hardware mechanism, *Automatic Path Migration (APM)*, which allows user transparent detection and recovery from network fault(s). However, the current InfiniBand literature lacks the understanding of APM intricacies, associated design tradeoffs and performance analysis.

In this paper, we address these challenges. We design a set of modules; *alternate path specification module*, *path loading request module* and *path migration module*, which work together for providing network fault tolerance for user level applications. We evaluate these modules with 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 the MPI (Message Passing Interface) 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 [5] 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 study of APM over InfiniBand and its performance analysis at the MPI application level.

*This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco systems, Linux Networx and Sun Microsystems; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, Dell, Microway, PathScale, IBM, SilverStorm and Sun Microsystems.

1-4244-0910-1/07/$20.00

The rest of the paper is organized as follows. In section 2, we provide the background of our work. In section 3, we present the design of the network fault tolerance modules; *alternate path specification module*, *path loading request module* and *path migration module*, which constitute the core of our APM based solution. We also present the integration of these modules at the Verbs layer and the MPI Layer. In section 4, we present the performance evaluation at the Verbs layer 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 the background information for our work. First, we provide a brief introduction of InfiniBand. This is followed by a detailed discussion on the APM feature in InfiniBand. We begin with the discussion on InfiniBand.

2.1 Overview of InfiniBand and QP Transition States

The InfiniBand Architecture (IBA) [9] defines a switched network fabric for interconnecting processing nodes and I/O nodes. An InfiniBand network consists of switches, adapters (called Host Channel Adapters or HCAs) and links for communication. For communication, InfiniBand supports different classes of transport services (Reliable Connection, Unreliable Connection, Reliable Datagram and Unreliable Datagram). In this paper, we focus on the reliable connection model. In this model, each process-pair creates a unique entity for communication, called *queue pair*. Each queue pair consists of two queues; *send queue* and *receive queue*. Figure 1 shows the communication state transition sequence for a QP. Each QP has a combination of *communication state* and *path migration state*. Figure 1 shows the communication state of the QP. Figure 2 shows a combination of communication and path migration state for the QP. In this section, we focus only on the communication state. In the next section, we discuss the combinations of these states.

At the point of QP creation, its communication state is RESET. At this point, it is assigned a unique number called qp_{num}. From this state it is brought to the INIT state by invoking modify_qp function. The modify_qp function is provided by the access layer of InfiniBand for different transition states provided by InfiniBand specification [9]. During the RESET-INIT transition, the QP 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 $DLID$ and

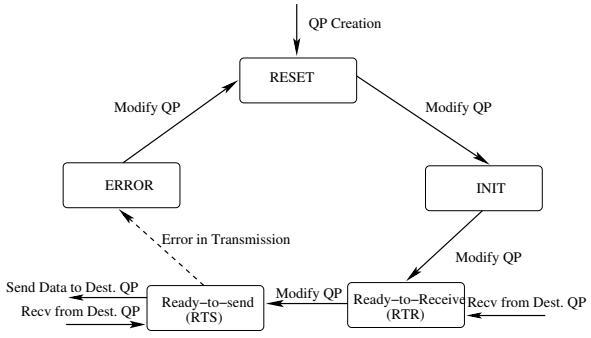

Figure 1. QP Communication State Diagram

the destination QP from which it will receive the messages. A modify_qp call brings it to READY-TO-RCV (RTR) state. At this point, the QP is ready to receive the data from the destination QP. 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 and receive data from its destination QP. Should any error(s) occur on the QP, the QP goes to the ERROR state automatically by the hardware. 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 (RESET-RTS) needs to be executed.

2.2 Overview of Automatic Path Migration

Automatic Path Migration (APM) is a feature provided by InfiniBand which enables transparent recovery from network fault(s) 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 the RC QP service type. For this feature, InfiniBand specifies *Path Migration States* associated with a QP. A valid combination of communication and path migration states are possible. This is shown further in Figure 2. In the figure, the path migration state of the QP is shown using oval shape. The possible communication states of the QP are shown using curly brackets. At any point of time, only one of the communication states is applicable to a QP. Once the QP is created, 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. Once the transition sequence (RESET-RTS) is complete, the QP's path migration state goes back to MIGRATED. Hence, this state is valid for QPs having their communication state as RTS. APM defines a concept of alternate path, which is used as an escape route should an error occur on the

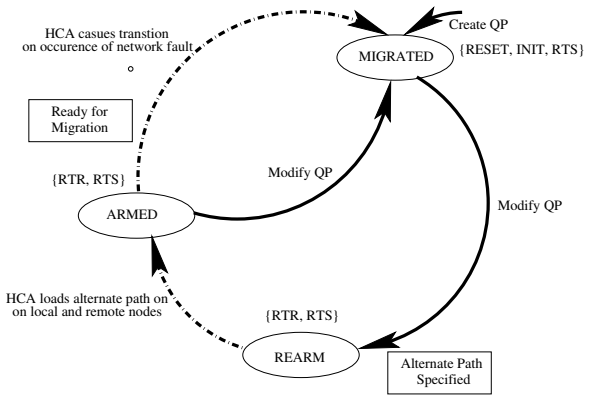

Figure 2. QP Path Migration State Diagram

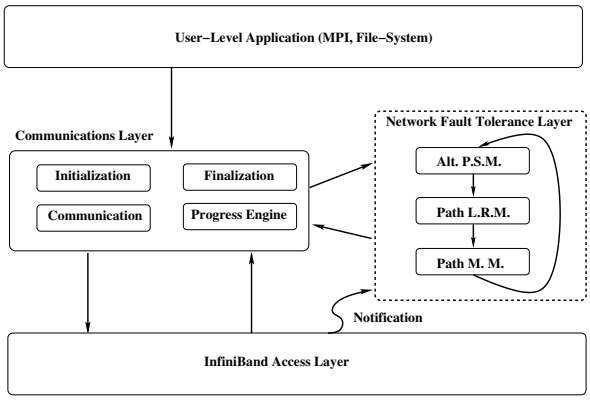

Figure 3. Overall Design of Network Fault Tolerance Modules and Interaction with User Applications

primary path of communication. The alternate path is specified by the user. This specification of the alternate path can be done at any point, beginning the INIT-RTR transition 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 REARM. Once the path has been loaded, 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. As an alternative, 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.

3 Overall Design

In this section, we present the overall design of network fault tolerance modules, their interactions with a user-level application and the communication layer of a user-level application. The interaction is shown in Figure 3. For simplicity, we have assumed that the interface between the access layer and the user-level application consists of a communication layer with the modules; *Initialization Module*, *Communication Module*, *Progress Engine* and *Finalization Module*. For different user-level applications, some of the modules may have more functionality than the others. Nevertheless, the modules are portable for different user-level applications (MPI, File-Systems etc). In this paper, we have used MPI as one of the candidate user-level application for integration with the network fault tolerance modules.

Figure 3 also shows the order in which the network fault tolerant modules can be called by the communications layer modules. The *alternate path specification module (PSM)* can be called at any point during the execution of the program. The *path loading request module (LRM)* can be called in conjunction with the alternate path specification module. It can also be called separately during the execution of the application. The *path migration module (PMM)* can be called only if the QP(s) for which the request is made are in the ARMED state. The notifications for different transition states of the APM are handled by the network fault tolerance modules.

3.1 Design of Network Fault Tolerance Modules

In this section, we present the modules which form the core in providing network fault tolerance for our design. The PSM is responsible for deciding the alternate path to be used in the presence of network fault(s). The LRM is responsible for requesting the alternate path to be loaded in the path migration state machine. The PMM is responsible for transition of alternate path to the primary path of communication. We decouple these modules with each other to make them more generic and portable for different user-level applications.

3.1.1 Alternate Path Specification Module

This module is responsible for specifying an alternate path to be used by a queue pair. The request for alternate path to be used can be done manually, or automatically by the HCA, should an error occur on the primary path of communication.

In our design, the alternate path can be specified by the user or chosen automatically by the module. Specification of the alternate path requires providing a couple of parameters: alt_{DLID} (the destination LID of the alternate path), alt_{PORT} (the HCA port for the alternate path), $alt_{SRC-PATH-BITS}$ (the LMC value to be used for the alternate path). A primary benefit of using APM

is that the connection remains established during the movement of path. This is achieved by keeping qp_{num} (the QP number) to be the same for the alternate path.

3.1.2 Path Loading Request Module

This module is responsible for initiating the loading of the alternate path for a QP. The module accepts a parameter for the list of the processes, for which this step needs to be done. This module can be invoked during anytime of the program execution after the RESET-INIT transition sequence has been completed for the QP(s). The completion of the request can be done using asynchronous events or polling mechanism. We discuss the tradeoffs of these approaches as follows.

Completion of Path Loading Request: The completion of the request for alternate path can be done using notification mechanism. Alternatively, the Verbs API provides a $query - qp$ function call to check the path migration state of a QP. Using the $query_qp$ mechanism, we can ascertain the path migration state of a QP (path migration state should be ARMED to call path migration module, should be migrated to call the path loading request module). We have noticed that the cost of querying a QP is higher than the overhead generated with the asynchronous notification. Hence, we use an asynchronous thread based notification handling of these events. The completion of the request(s) is notified by asynchronous event(s), which we refer to as the $event_{ARMED}$ in this paper.

3.1.3 Path Migration Module

This module is invoked when a user wants to use the alternate path to be used as a primary path of communication, in the absence of a network fault. This functionality is useful in providing load balancing with the available paths. 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 notified with the help of asynchronous events, which are referred as $event_{MIGRATED}$ in this paper. The asynchronous thread discussed in the previous section is enhanced to handle these events. In the performance evaluation section, the invoking of this module is referred by Armed-Migrated legend.

3.2 Interaction of Communication Threads and Network Fault Tolerance Modules

In this section, we present the interactions of Main Execution Thread and the Asynchronous Thread with the Network Fault Tolerance Modules. Figure 4 shows the possible interactions. The interactions from the main execution thread are shown with solid lines, the interactions with asynchronous thread are shown with dotted lines. Although, both threads can interact with the network fault tolerance modules, the main execution thread can execute the modules at any stage of the application execution. The asynchronous thread can call the path migration module on the occurrence of $event_{ARMED}$. On the occurrence of $event_{MIGRATED}$, the asynchronous thread can call alternate path specification module and path loading request module or the alternate path specification module only. We limit the asynchronous thread to execute the modules only at the occurrence of events, since the thread is active only on the occurrence of events.

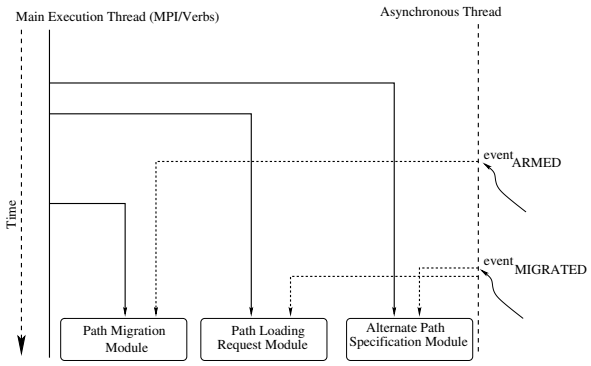

Figure 4. Interaction of Network Fault Tolerance Modules with Main Execution Thread and Asynchronous Thread

3.3 Integration of Network Fault Tolerance Modules at Verbs and MPI Layer

We implement our network fault tolerance modules, so that various user level applications can leverage them without any changes to the modules specific to the application. For the micro-benchmarks at the Verbs Layer, we extend the micro-benchmark suite discussed in our previous work [10, 11].

For integration at the MPI layer, we use MVAPICH[1],

[1]MVAPICH/MVAPICH2 [12] are currently used by more than 445 organizations worldwide. It has enabled several large InfiniBand clusters to obtain top 500 ranking. A version is also available in an integrated manner with OpenFabrics Enterprise Distriburion (OFED)

a popular MPI over InfiniBand. We implement an asynchronous thread, and add data structures to reflect the path migration state of a QP. The solution will be available in an open-source manner in our upcoming release of MVAPICH.

4 Performance Evaluation

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

4.1 Experimental Testbed

Our Experimental Testbed consists of a set of Intel Xeon nodes each having a 133 MHz PCI-X slot. Each node has two Intel Xeon CPUs running at 2.4 GHz , 512 KB L2 cache and 1 GB of main memory. This cluster uses 2nd Generation MT23218 4X Dual Port HCAs from Mellanox [1]. We used the Linux 2.6.9-15.EL kernel version [2] and Verbs API (VAPI) from Mellanox provided with the InfiniBand Gold CD (IBGD). The HCA firmware version used is 3.3.2. The nodes are connected with a 144-port Single Data Rate (SDR) switch. The switch uses OpenSM; a popular subnet manager provided with IBGD. Since each HCA has two ports, we connect both ports to the switch, and use first port as the primary path and second port as the alternate port for communication.

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 use a ping-pong latency test. To understand the impact on a large scale cluster, we create multiple QPs 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 and one QP is used.

4.2.1 Impact of PSM and LRM on QP Transitions

In Figures 5 and 6, we present the time consumed in Migrated-Armed and Armed-Migrated transition sequences with the increasing number of QPs between the processes. To calculate the timings for Migrated-Armed transition, the alternate path specification module is invoked during INIT-RTR phase and time is calculated till $event_{ARMED}$ for all QPs is received by the asynchronous thread. For calculating the time for the Armed-Migrated transition, path migration module is invoked for all QPs. Once the asynchronous thread receives $event_{MIGRATED}$ for all QPs, the shared data structures between the main thread and the asynchronous thread are updated. A linear trend is observed with the increasing number of QPs in these transitions. 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%. The main purpose of the above tests is to calculate the maximum penalty observed by a user-level application. However, since these requests are non-blocking, it remains to be seen, how these transitions impact the ongoing communication.

4.2.2 Impact of Network Fault Tolerance Modules on Latency

Figure 7 compares the performance of the original case with different transition sequences using the ping-pong latency test. We slightly modify the test to report the latency observed at every iteration to clearly understand the impact of different transitions on the latency. In our evaluation, we note that the latency observed increases, till all the events corresponding to a transition sequence are received. For those latency values, we calculate the average and report them in the figure. We have also observed that the number of iterations in the test, which have impact on latency is very close to the number of QPs for which the transition is requested. As a result, we almost see a flat curve for the average latency. The results show that both Migrated-Armed and Armed-Migrated requests add significant overhead to the ongoing communication. However, this overhead remains constant with the increase in the message size.

We now show the results for our acid test, the impact of performance on latency, when a network fault occurs. After the alternate path is loaded, we disable the primary path of communication by un-plugging the cable corresponding to the primary path of communication on the sender side. The HCA automatically moves the alternate path as a primary path of communication for the currently used QP. Since QPs are used in a round robin fashion, this step is executed for all QPs. We measure the average latency observed till the $event_{MIGRATED}$ for all QPs has been generated. This test helps us understand the impact on latency for small messages on large scale clusters, when each process pair uses one QP for communication. Figure 8 shows

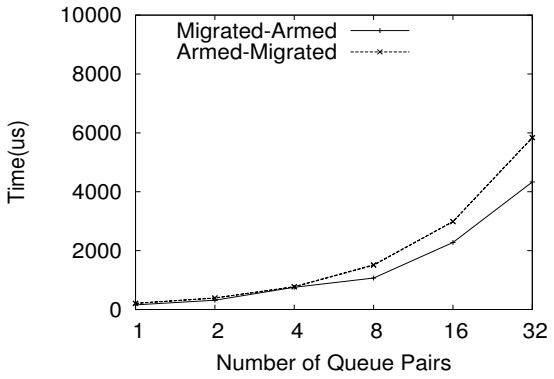

Figure 5. Timings for different transition states in APM, Small Number of QPs

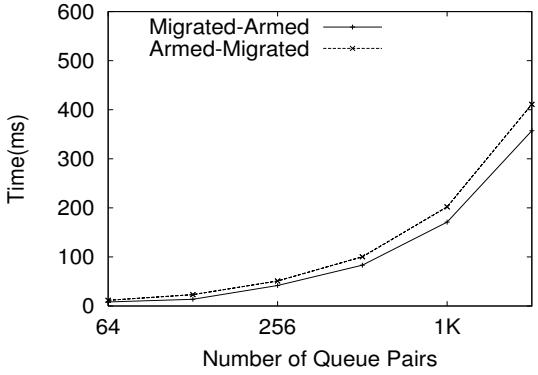

Figure 6. Timings for different transition states in APM, Large Number of QPs

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 the same independent of the message size.

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

In this section, we present the performance for NAS parallel benchmarks [5], when different APM sequence transitions are requested. The impact on performance in the presence of network faults is also studied. A 4x2 configuration (4 nodes and 2 processes per node) is used for executing the applications. The applications are profiled to make sure that network fault tolerance modules are invoked during the critical execution phase of the application. The primary communication path is broken by unplugging the cable at different points in the application execution for sixteen runs. The average performance observed is presented. At this point, we do not have a very systematic methodology for network fault injection. In future, we plan to design software based fault injection mechanism. Figures 9 and 10 show the results with different transitions sequences in APM using Integer Sort kernel, with Class A and Class B problem size. The results in the presence of network faults are also presented. In the absence of network faults, different APM transition sequences incur some overhead for Class A. In the presence of network fault, a very significant amount of overhead is observed. Since the results reflect an average case, they show a healthy mixture of the cases, when the application was busy computing, busy in communication and their combinations. Increasing the number of QPs to emulate a large scale cluster also shows an interesting trend. In the presence of a network fault, all QPs used in the round robin fashion observe a transition of alternate path to the primary path. Figure 10 shows the results for Class B problem size. The execution time is longer for the problem size. The impact of different APM transition sequences is lesser as a result. The time for QP transitions in the absence of faults and presence of faults largely remains independent of the message size as shown during the performance evaluation with the tests at the Verbs layer. The number of events generated are also largely dependent upon the number of QPs used. Hence as the execution time of an application increases, the relative overhead shown due to APM in both the absence and the presence of faults decreases. Figures 11 and 12 show the results for NAS FT Class B and LU Class B, respectively. Since the overhead incurred per QP almost remains same, when a network fault occurs, we notice that the percentage of performance degradation is much lesser in these cases. 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.

5 Related Work

A couple of researchers have focused on designing MPI for providing network fault tolerance. Recently, we have designed a software based solution for networks supporting uDAPL interface [14]. The error detection and re-transmission is done at the software layer. However, APM provides user-transparent error detection and recovery by using an alternate route. LA-

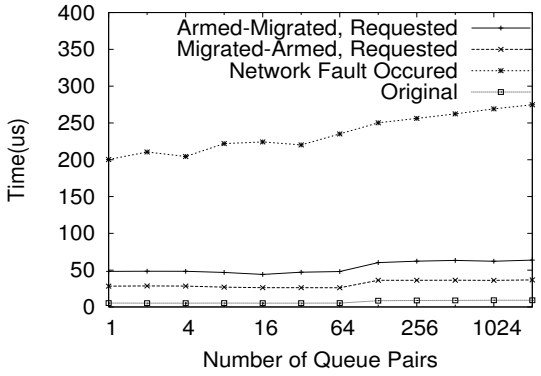

Figure 7. Impact on Latency for 128 Byte Message with Increasing Number of QPs

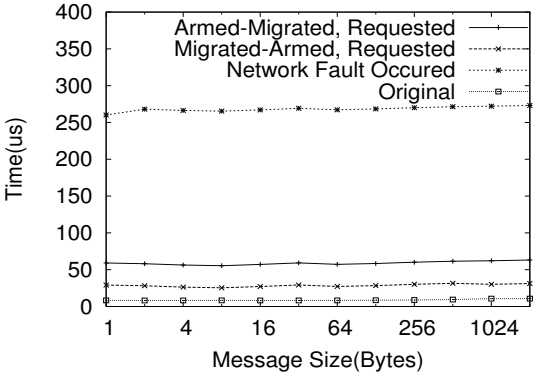

Figure 8. Impact on Latency for Small Messages using 512 QPs

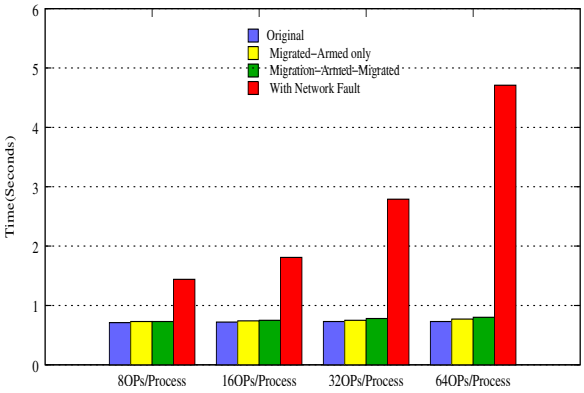

Figure 9. Performance Evaluation on IS, Class A, 4x2 Configuration

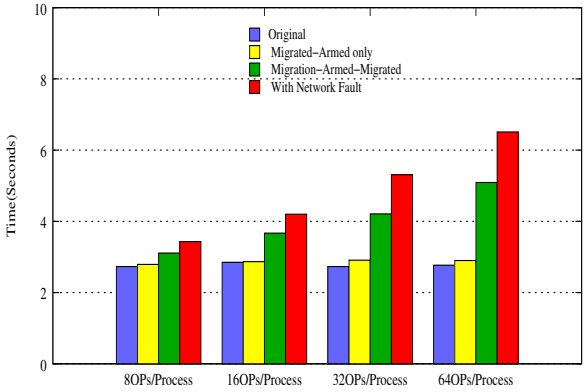

Figure 10. Performance Evaluation on IS, Class B, 4x2 Configuration

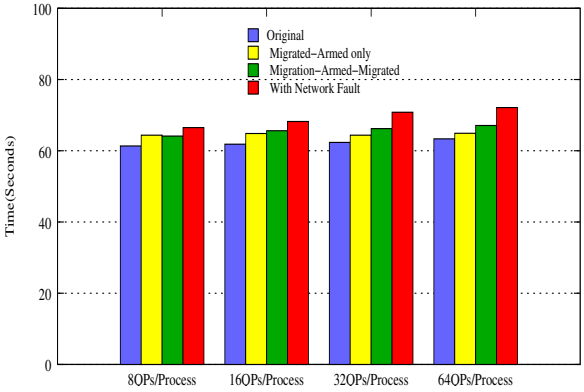

Figure 11. Performance Evaluation on FT, Class B, 4x2 Configuration

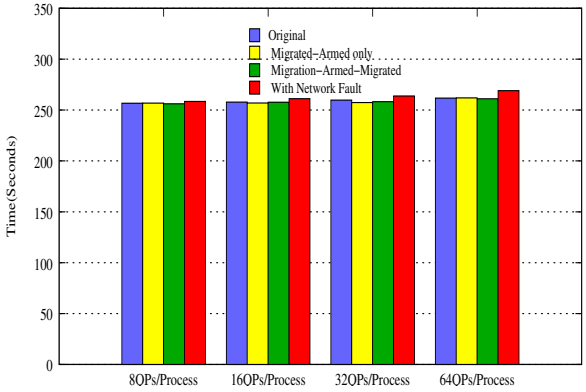

Figure 12. Performance Evaluation on LU, Class B, 4x2 Configuration

MPI [8] is an MPI implementation developed at Los Alamos National Labs. LA-MPI was designed with the ability to stripe message across several network paths and supports network fault tolerance at the software layer. OpenMPI [6] also provides striping across multiple interconnects. Recently, support for network fault tolerance has been proposed for OpenMPI [4]. Network Fault tolerance with Quadrics is implemented in the hardware [13] and Myrinet uses dispersive routing for implementing network fault tolerance [7].

However, none of the above works have focused on utilizing the hardware mechanism, APM for network fault tolerance over InfiniBand. In this paper, we have used this mechanism for network fault tolerance design at the Verbs and the MPI layer.

6 Conclusions and Future Work

In this paper, we have designed a set of modules; *alternate path specification* module, *path loading request module* and *path migration module*, which work together for providing network fault tolerance with APM for user level applications. We have integrated 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 have also integrated these modules with the MPI layer to provide network fault tolerance for MPI Applications. 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. To the best of our knowledge, this is the first study of APM over InfiniBand and its detailed study with the MPI applications.

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 plan to provide a combination of hardware and software based solution for network fault tolerance.

References

[1] Mellanox Technologies. http://www.mellanox.com.

[2] The Linux Kernel Archives. http://www.kernel.org/.

[3] TOP 500 Supercomputer Sites. http://www.top500.org.

[4] T. Angskun, G. E. Fagg, G. Bosilca, J. Pjesivac-Grbovic, and J. J. Dongarra. Self-Healing Network for Scalable Fault Tolerant Runtime Environments. In *Proceedings of 6th Austrian-Hungarian workshop on distributed and parallel systems*, Innsbruck, Austria, September 2006. Springer-Verlag.

[5] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. Number 3, pages 63–73, Fall 1991.

[6] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, 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 *EuroPVM/MPI*, pages 97–104, 2004.

[7] P. Geoffray. Myrinet eXpress (MX): Is Your Interconnect Smart? In *HPCASIA '04: Proceedings of the High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference on (HPCAsia'04)*, pages 452–452, Washington, DC, USA, 2004. IEEE Computer Society.

[8] 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. volume 31, pages 285–303, Norwell, MA, USA, 2003. Kluwer Academic Publishers.

[9] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2, October 2004.

[10] J. Liu and B. Chandrasekaran and W. Yu and J. Wu and D. Buntinas, S. P. Kinis, P. Wyckoff, and D. K. Pand. Micro-Benchmark Level Performance Comparison of High-Speed Cl uster Interconnects. In *Hot Interconnect 11*, August 2003.

[11] J. Liu, A. Mamidala, A. Vishnu, and D. K. Panda. Performance Evaluation of InfiniBand with PCI Express. In *Hot Interconnect 12 (HOTI 04)*, August 2004.

[12] 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.

[13] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. volume 22, pages 46–57, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[14] 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.