A Software Based Approach for Providing Network Fault Tolerance in Clusters with uDAPL interface: MPI Level Design and Performance Evaluation

ABHINAV VISHNU, PRACHI GUPTA, AMITH MAMIDALA AND DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-5/06-TR58

A Software Based Approach for Providing Network Fault Tolerance in Clusters with uDAPL interface: MPI Level Design and Performance Evaluation*

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

Abstract

In the arena of cluster computing, MPI has emerged as the de facto *standard for writing parallel applications. At the same time, introduction of high speed RDMA-enabled interconnects like InfiniBand, Myrinet, Quadrics, RDMA-enabled Ethernet has escalated the trends in cluster computing. Network APIs like uDAPL (user Direct Access Provider Library) are being proposed to provide a network-independent interface to different RDMA-enabled interconnects. Clusters with combination(s) of these interconnects are being deployed to leverage their unique features, and network failover in wake of transmission errors. In this paper, we design a network fault tolerant MPI using uDAPL interface, making this design portable for existing and upcoming interconnects. Our design provides failover to available paths, asynchronous recovery of the previous failed paths and recovery from network partitions without application restart. In addition, the design is able to handle network heterogeneity, making it suitable for the current state of the art clusters. We implement our design and evaluate it with micro-benchmarks and applications. Our performance evaluation shows that the proposed design provides significant performance benefits to both homogeneous and heterogeneous clusters. Using a heterogeneous combinations of IBA and Ammasso-GigE, we are able to improve the performance by 10-15% for different NAS Parallel Benchmarks on 8x1 configuration. For simple micro-benchmarks on a homogeneous configuration, we are able to achieve an improvement of 15-20% in throughput. In addition, experiments with simple MPI micro-benchmarks and NAS Applications reveal that network fault tolerance modules incur negligible overhead and provide optimal performance in wake of network partitions.*

*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, Sun Microsystems, Linux Networx and Mellanox; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, IBM, Advanced Clustering and Sun Microsystems.

1 Introduction

In the arena of cluster computing, MPI has emerged as the *de facto* standard for writing parallel applications. At the same time, introduction of high speed RDMA-enabled interconnects like InfiniBand, Myrinet, Quadrics, RDMA-enabled Ethernet has escalated the trends in cluster computing. Network APIs like uDAPL (user Direct Access Provider Library) are being proposed to provide a network-independent interface to different RDMA-enabled interconnects. Clusters with combination(s) of these interconnects are being deployed to leverage their unique features, and to provide network failover in wake of transmission errors. However, wide variety of interconnects pose portability issues. This limits their different combinations to be used in network failures, in addition to providing optimal performance. In this paper, we take these challenges. We design a network fault tolerant MPI using uDAPL interface, making this design portable for existing interconnects.

Our design provides failover to available paths, recovery of the previous failed paths asynchronously and recovery from network partitions without application restart. In addition, the design is able to handle network heterogeneity, making it suitable for the current state of the art clusters. To achieve these goals, we design low overhead *completion filter and error-detection*, *message (re)-transmission* and *path recovery and network partition handling* modules which perform completion filter and detection, (re)-transmission and recovery from network partitions respectively. We implement our design and evaluate it with micro-benchmarks and applications. Our performance evaluation shows that the proposed design provides significant performance ben-

efits to both homogeneous and heterogeneous clusters. Experiments reveal that network fault tolerance modules incur very low overhead and provide optimal performance in wake of network failures for simple MPI micro-benchmarks and applications. In addition, in the absence of such failures, using a heterogeneous 8x1configuration of IBA and Ammasso-GigE, we are able to improve the performance of NAS Parallel Benchmarks by 10-15% for different benchmarks. For simple micro-benchmarks, we are able to improve the throughput by 15-20% for uni-directional and bi-directional bandwidth tests. Even though, the evaluation in the paper has been done using InfiniBand and Ammasso-GigE, there are emerging interconnects, which plan to support uDAPL interface and are not available in market commercially. The proposed design is generic and capable of supporting any interconnect with uDAPL interface.

The rest of the paper is organized as follows. In section 2, we provide background of our work. In section 3, we present the related work. In section 4.1, we present basic infrastructure *multi-network abstraction layer and communication methodology for multiple interconnects* associated with our work. In section 4.2, we discuss the network fault tolerance modules *completion filter and error detection module, message (re)-transmission module, and path recovery and network partition handling module*. In section 5, we present the performance evaluation for homogeneous and heterogeneous clusters, both in the absence and presence of faults. In section 6, we conclude and present our future directions.

2 Background

In this section, we provide the background information for our work. We begin with a brief introduction of interconnects, followed by an overview of the uDAPL interface. We also discuss major internal communication protocols used by Message Passing Interface (MPI).

2.1 Overview of Interconnects

InfiniBand has emerged as a major player in the arena of high performance computing. The InfiniBand Architecture (IBA) [11] defines a System Area Network with a switched, channel-based interconnection fabric. IBA 4X can provide bandwidth up to 10 Gbps. Switches and Adapters with capabilities of 12X bandwidth have also become available in the market, providing performance upto 30Gb/s. InfiniBand defines Verbs for user-level applications to leverage its capabilities. VAPI (Verbs API) by Mellanox has been widely used for powering large scale clusters. In addition, an open source effort, OpenIB [16] has also become available. In this paper, we have used DAPL libraries designed over VAPI for performance evaluation.

The Ammasso interconnect is a RDMA-enabled Gigabit Ethernet adapter [4]. It is a full duplex 1Gbps Ethernet Adapter also provides the interface for vanilla sockets based applications. Ammasso defines a CCIL interface, for applications to leverage its RDMA capabilities. In this paper, we have used Ammasso 1100 and DAPL libraries designed over CCIL interface for performance evaluation.

2.2 uDAPL

As mentioned in the section 2.1, multiple interconnects have emerged that provide RDMA capabilities. However, these interconnects do not provide a common set of Application Programming Interfaces (APIs). In addition, upcoming interconnects face a similar challenge and the turnaround time for developing MPI on these adapters can be prohibitive. To alleviate this situation, DAT (Direct Access Transport) Collaborative [8] has defined a DAPL interface, providing a common interface for different interconnects. User Direct Access Programming Library (uDAPL) is a lightweight, transport-independent, platform-independent user-level library, potentially capable of providing high productivity for upcoming and existing interconnects.

uDAPL allows processes to communicate by defining *End Points (EPs)*. EPs need to be connected to each other, before communication can take place. *Work Requests or descriptors* can be posted on the end points for sending or receiving data from other processes. uDAPL supports memory semantics by leveraging RDMA and channel semantics by providing *send/receive mechanism*. The completion status of the previously posted descriptors can be ascertained by using *completion queue* mechanism. Completion queue returns status of the posted descriptor, in terms of success/failure and the error code.

2.3 Overview of MPI Protocols

MPI defines four different communication modes: *Standard*, *Synchronous*, *Buffered*, and *Ready*. Two internal protocols, *Eager* and *Rendezvous*, are usually used to implement these four communication modes. These protocols are handled by a component in the MPI implementation called *progress engine*. In Eager protocol, the message is pushed to the receiver side regardless of its state. In Rendezvous protocol, a handshake takes place between the sender and the receiver via control messages before the data is sent to the receiver side. Usually, Eager protocol is used for small messages and Rendezvous protocol is used for large messages. Figure 1 shows examples of typical Eager and Rendezvous protocols.

For the transfer of large data buffers, it is beneficial to avoid extra data copies. A zero-copy Rendezvous protocol implementation can be achieved by using RDMA write. In this implementation, the buffers are pinned down in mem-

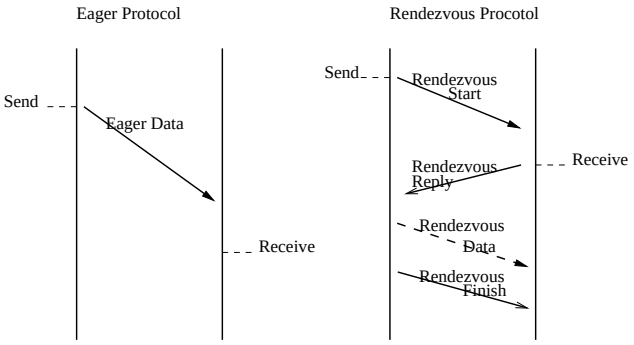

Figure 1. MPI Eager and Rendezvous Protocols

ory and the buffer addresses are exchanged via the control messages. After that, the data can be written directly from the source buffer to the destination buffer by doing RDMA write. Similar approaches have been widely used for implementing MPI over different interconnects [15, 12, 6].

For small data transfer in Eager protocol and control messages, the overhead of data copies is small. Therefore, we need to push messages eagerly toward the other side to achieve better latency. This requirement matches well with the properties of InfiniBand send/receive operations. However, send/receive operations have their disadvantages such as lower performance and higher overhead. In [14], we proposed a scheme that uses RDMA operations for small data and control messages. This scheme improves both latency and bandwidth of small message transfers in MPI. The solution is available in an open source manner with MVAPICH [1]

3 Related Work

A couple of researchers have focused on designing MPI for multiple interconnects and providing network fault tolerance. MPI-2 using uDAPL interface has been proposed in [3]. However, in this work a combination of interconnects was not used simultaneously. In our previous work [13, 18], we have focused on combining multiple IBA HCAs. However, the design is applicable only to IBA. LA-MPI [9, 10, 2] is an MPI implementation developed at Los Alamos National Labs. LA-MPI was designed with the ability to stripe message across several network paths. LA-MPI is able to handle striping across different interconnect types. Open-MPI [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 [17], which provides support for multiple interconnects and failover. Also, the design uses TCP for RDMA-enabled Ethernet Adapters, however uDAPL is capable of leveraging RDMA for RDMA-enabled Ethernet Adapters. Buntinas et. al. has also proposed Nemesis [7], which is capable of supporting multiple interconnects. At the time of writing this paper, nemesis channel over InfiniBand is not available. However, none of the above works have focused on providing network fault tolerance with support for network partitions and asynchronous recovery of failed paths in addition to providing portability with the uDAPL interface.

[1] MVAPICH/MVAPICH2 [15] are high performance MPI-1 and MPI-2 implementations from The Ohio State University, currently being used by more than 330 organizations across 33 countries.

4 Overall Design for uDAPL Based Network Fault Tolerant MPI

In this section, we present the overall design for our uDAPL based network fault tolerant MPI. This is further illustrated in Figure 2. The figure represents the overall design and an example node configuration consisting of both IBA and GigE devices. In section 4.1, we present *multi-network* abstraction layer, which provides a uniform interface to our design for clusters with network heterogeneity. We also discuss the implementation issues associated with using multiple interconnects.

In section 4.2, we present the workhorse of our design, *communications and network fault tolerance layer*. Figure 5 presents the interaction of different components in communications and network fault tolerance layer. This layer comprises of modules, which work together for scheduling the communication in an efficient manner to providing network fault tolerance. The *message (re)-transmission module* in this layer is responsible for scheduling the communication on available paths according to scheduling policy. The *completion filter and error detection* module detects error and provides information to *message (re)-transmission* about the failed work request. The *path repository* maintains the available paths for every pair of communication nodes. This layer also consists of *path recovery and network partition handling module*, which is responsible for recovery of failed paths and dealing with network partitions. To the best of our knowledge, this is the first implementation of network fault tolerant MPI which supports dealing with network partitions and recovery of the failed paths, without application restart.

4.1 Basic Infrastructure For Network Fault Tolerance Design

In this section, we discuss the basic design, which acts as an infrastructure to providing network fault tolerance. Our design is capable of providing network fault tolerance for

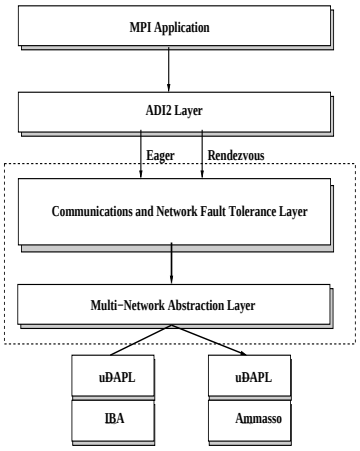

Figure 2. Overall Design of Network Fault Tolerant MPI with a node comprising of Multiple Networks

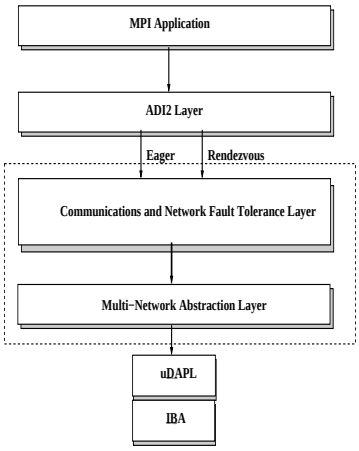

Figure 3. A node with only IBA Network

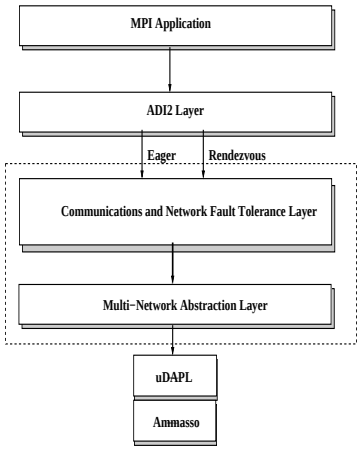

Figure 4. A node with only Ammasso Network

clusters using single interconnect, in addition to a combination of interconnects supporting uDAPL interface. We begin with the introduction of multi-network abstraction layer.

4.1.1 Multi-Network Abstraction Layer

As shown in Figures 2, 3 and 4, our design is capable of combining clusters with a combination of interconnects, supporting uDAPL interface to the user applications. Homogeneous clusters are a special case of this configuration. Each interconnect specifies its own DAPL library, built over the interconnect's access layer. Hence, presence of an equivalent abstraction is imperative to hiding network heterogeneity. This layer provides an equivalent interface of multiple uDAPL interfaces to the communications and fault tolerance layer. To provide such an abstraction, this layer maintains unified data structures for end point(s), public service point(s), completion queue(s) and available paths between processes. Our design is currently limited by the fact that it requires atleast one common network interface between all communicating nodes. In the performance evaluation section, we have used Ammasso-GigE as the common interface. However, the design is generic and any interconnect providing DAPL interface can be used. In future, we plan to extend our design to handle more complicated cases of network heterogeneity.

4.1.2 Implementing Abstraction Layer over DAPL

In our previous work with uDAPL [3], we have presented *asynchronous* and *polling* based connection management schemes to connect EPs associated with different processes. In the design, the EP(s) information is exchanged, followed by mandatory *ep_connect* function call to connect them as specified by the uDAPL specification. However, the design assumed the presence of only one network interface. To support network heterogeneity, each node exchanges its node configuration at the MPI initialization phase. Node configuration comprises of *DAPL provider* information, and associated parameters with different interconnects. This information is communicated to peers at the time of EP exchange phase, to avoid multiple messages being sent for node configuration exchange. Thread-based EP connection scheme is used for connecting various EPs. At the end of this step, each node updates its *path repository* for communication to every other node in the cluster.

4.1.3 Communication Methodology for Multiple Interconnects

As mentioned in section 2, uDAPL allows user to use RDMA for data transfer. One of the key requirements is that the user buffer be registered with the corresponding interconnect. Since our design supports multiple interconnects, for simplicity, we register the complete buffer with all interconnects. In addition, for the rendezvous protocol, completion notifications need to sent on all interconnects participating in data transfer to the communicating process. Presence of multiple paths also leads to *out-of-order* messages. MPI requires messages to be processed in order. Hence, we maintain out-of-order queues, and periodically poll on them.

As discussed in [13, 18], scheduling policies have a great deal of impact on performance, when a combination of paths are available. Simple policies like *even striping*, *round robin*, *process binding* and *weighted striping* provide

comparable performance for a combination of interconnects with similar peak bandwidth. However, these policies provide a sub-optimal performance when different paths have different bandwidths. In [13], we have also shown that *adaptive striping* stands out the best candidate in such scenarios. Hence, without much further a do, we use this policy, so that our design leverages multiple networks in an optimal fashion, in addition to using them for failover. A *striping threshold* value is used, below which the primary network for communication is used. In the performance evaluation section, we have used *adaptive striping* policy by default, unless mentioned otherwise. In the performance evaluation section, we have used InfiniBand as the primary path, wherever possible.

4.2 Design of Communications and Network Fault Tolerance Layer

In this section, we discuss the design of communications and network fault tolerance layer. We discuss various modules associated with this layer and their interactions. This is shown in more detail in the Figure 5. We begin with the error detection module.

4.2.1 Completion Filter and Error Detection Module

As mentioned in section 2, uDAPL library allows a user application to make work requests by posting send work requests or descriptors. The status of these requests can be determined by using the completion queue mechanism. As shown in Figure 5, completion notifications generated from the network are stored in the completion queue. uDAPL also provides completion notification interrupt to be generated for solicited work requests, however this mechanism leads to increased latency, particularly for small messages. In our design, we use *polling* on the completion queue to determine the status of the work request. This is to be noted, that a completion queue entry (CQE) is generated, independent of success/failure in completing the work request. Upon receipt of a successful CQE, this module updates the weight(s) of different path(s) of communication to the communicating process, as shown in Figure 5. However, on receiving a failed CQE, associated error code in the CQE is used determine the cause of the failure. We leverage this uDAPL capability to ascertain the failure in completion of a send or a receive work request. The *remote access error* failure opcode shows the un-reachability of the *remote destination*. This failure implies that even after multiple retries by the Network Interface Card (NIC), the path could not be reached. This is to be noted, such a failure can also occur, when the *rkey* value for RDMA operation is wrong. However, in both cases, occurrence of even a single failure on an End point breaks the connection and all posted work requests (send or receive) result into error. The recovery mechanism of the broken EP is handled in the *Path Recovery and Network Partitioning Module*. Once the error is detected, the control is transferred to *message re-transmission module*.

4.2.2 Message (Re)Transmission Module

This module is activated upon receiving an input from the completion filter and error detection module or receiving an input from the ADI layer for message transmission. If the request is received from the ADI layer, the appropriate scheduling policy is used for message transmission. The interaction is further illustrated in Figure 5. Upon receipt of a failed CQE from completion filer and error detection module, the first step is to update the path repository, marking the associated communication path to the destination process *unavailable*. Upon receipt of a failed CQE with receive *opcode*, the corresponding buffer is simply released, however another receive descriptor is not posted, since the connection is already broken. As mentioned before, posting another work request on a broken connection results in error. However, when a CQE failed send opcode is received, *path repository* is queried for the available paths to the destination rank. The return from the path repository can be *success* with a list of the available path(s) or a *failure* in case of network partition (it is not be noted that the sender may still have communication paths to other processes). The failed send descriptor consists of information about the length of the work request. To post this descriptor to available paths, length of each work request is adjusted in conjunction with scheduling policy and associated lkey for interconnect is used. In addition, if an RDMA operation is requested, the associated rkey is updated for data transfer.

4.2.3 Path Recovery and Network Partition Handling Module

The design mentioned upto now provides failover, when network paths fail and message re-transmission in such cases. However, network errors can be transient and this should not limit the application to be able to use the corresponding paths when they recover. In addition, an application should not abort in case of network failures, since the process state is intact. Long running applications should also be able to use the recovered paths, and be able to extract the best performance out of the SAN. This layer meets delivers the above requirements by using an asynchronous thread based recovery mechanism.

In order to facilitate this capability, the broken End Point associated with the failed network path needs to be brought back to the connected state. This is further illustrated in the Figure 7. The DAT specification mentions that an End Point in an error state should not be moved to disconnected

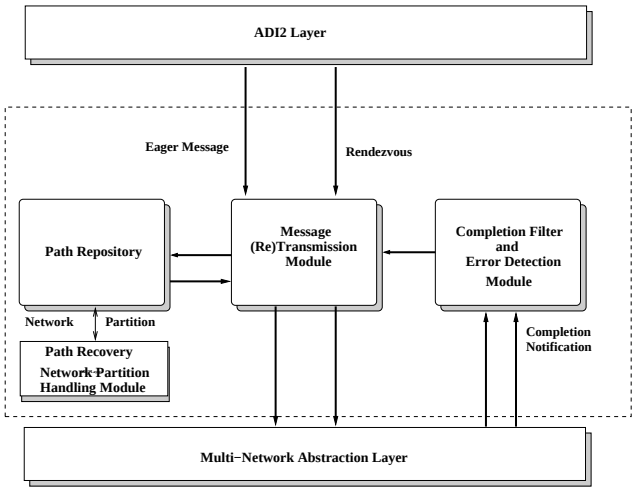

Figure 5. Communications and Network Fault Tolerance Layer

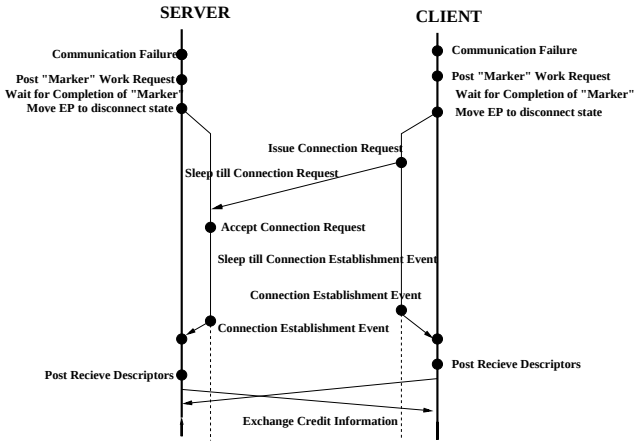

Figure 6. Communication Protocol For Recovery from Network Partitions and Previously Failed Paths

state at the discovery of first failure, else would result in loss of the previously posted work requests. Hence, in our design, we post a send work request called *marker*. Since Work requests always finish in order on the requested side, after receiving a CQE associated with the marker, the End Point can be moved to the disconnected, followed by the unconnected state as shown in Figure 7.

The communication protocol for recovery from network partition is further illustrated in Figure 6. When a process receives the first communication failure, it initiates an asynchronous thread which initiates request(s) for bringing back the End Points to the connected state. As mentioned in our previous work [3], each process acts as a server for processe(s) with higher MPI rank and sends connect requests only to processe(s) with lower rank. Since connection requests can possibly arrive at any point of time, the asynchronous server thread remains in *sleep* state during the program execution and wakes up only during connection request(s) from the client(s). Similarly, the client thread initiates request(s), goes to sleep and only activates, when the connection event(s) are generated. Once a client and server have received the connection events, each of their End Points are in *connected* state. At this point, each of the processes post receive descriptors, and exchange the credit information by sending a *connect* message. Once the processes receive the message, they are ready for communication. This is to be noted, Since these threads are in *sleep* state for most of the time during program execution, they incur little contention to the main thread.

5 Performance Evaluation

In this section, we evaluate the performance of our design. We call our design *MN-uDAPL* and compare its performance with MVAPICH-0.9.7 for OSU Tests [15] and NAS Parallel Benchmarks [5]. Our Performance Evaluation is further divided into multiple cases:

- No network fault(s) occur during the application execution in the SAN. This evaluation helps us understand the performance improvement which can be achieved when there are multiple interconnects in the SAN, in homogeneous/heterogeneous environment.
- One or more Network faults occur during the application execution in the SAN. We evaluate the cases when a previously failed path recovers during the application execution, to the cases of network partitioning. This helps us understand the overhead incurred by network fault tolerance modules, when such faults occur.

We begin with a brief description of our experimental testbed.

5.1 Experimental Testbed

Figure 8 is a block diagram for our experimental testbed. This cluster consists of eight SuperMicro SUPER X5DL8-GG nodes with ServerWorks GC LE chipsets. Each node has dual Intel Xeon 3.0 GHz processors, 512 KB L2 cache, and PCI-X 64-bit 133 MHz bus. We have used InfiniHost MT23108 Dual-Port 4x HCAs from Mellanox. The ServerWorks GC LE chipsets have two separate I/O bridges and three PCI-X 64-bit 133 MHz bus slots. The kernel version

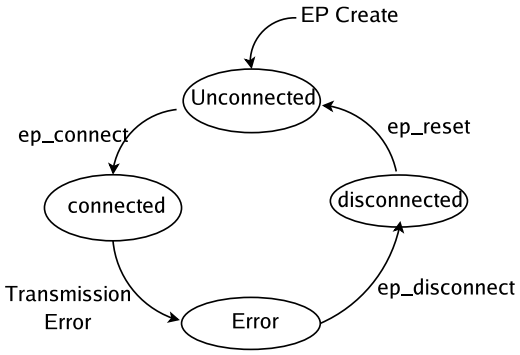

Figure 7. End Point State Transition Diagram

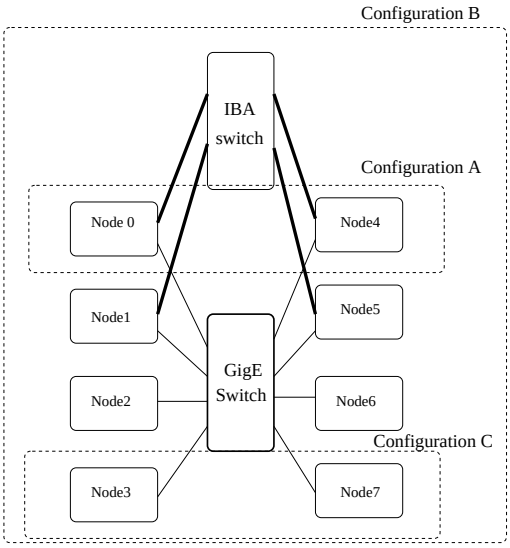

Figure 8. Experimental Testbed Configuration

we used is Linux 2.6.9smp. The IBGD version is 1.8.2 and HCA firmware version is 3.3.2. The Front Side Bus (FSB) of each node runs at 533MHz. The physical memory is 2 GB of PC2100 DDR-SDRAM.

Four nodes in the cluster comprises of one InfiniBand(InfiniHost MT23108 Dual-Port 4x HCAs from Mellanox) and eight nodes comprise of Ammasso (Ammasso 1100 RDMA-enabled Gigabit-Ethernet Adapter) each. uDAPL libraries provided by Mellanox and Ammasso are used for performance evaluation.

5.2 Performance Evaluation on Configuration A

As shown in Figure 8, this configuration comprises of two nodes which have both IBA and GigE network interface cards. Figure 9 shows the latency of small messages for different devices of MVAPICH, 0.9.7. Since the messages are small, only IBA device is used for communication. Messages above the striping threshold (256K) use adaptive striping for communication. In comparison to MVAPICH-0.9.7, uDAPL device, our MPI incurs negligible overhead. The overhead in latency, when compared to VAPI device is due to the absence of *inline functionality* in uDAPL library. This functionality allows data to be posted alongwith the descriptor, hence reducing the number of I/O bus transactions. Figure 10 shows the performance of latency for large messages. Messages above the striping threshold are able to be benefited by using the adaptive striping policy. For 512Kbyte message, the latency improves by almost 10%.

Figure 11 and 12 show the performance for OSU unidirectional and bi-directional bandwidth test. As explained above, MN-uDAPL uses only InfiniBand device for messages of size lesser than the striping threshold. Adaptive striping provides a peak uni-directional bandwidth of 963 MB/s compared to 880 MB/s for MVAPICH-0.9.7, uDAPL device(IBA) only. The GigE device can only provide around 100 MB/s. Similarly, a performance improvement of 18% is seen for peak bandiwth in the bi-directional bandwidth test, which improves from 931 MB/s to 1095 MB/s.

5.3 Performance Evaluation on Configuration B

In this section, we evaluate the unified performance of the cluster, also the configuration B as shown in the Figure 8. We use MVAPICH 0.9.7, uDAPL device for evaluation and compare its performance with MN-uDAPL. Since MVAPICH-0.9.7 is capable of utilizing only one interface at a time, we evaluate it under two configurations for our cluster. In one configuration, it is able to utilize nodes with IBA cards only, and the other configuration can utilize nodes with GigE cards only. Figures 13 and 14 compare the performance of these configurations with MN-uDAPL, which is capable of handling this network heterogeneity in a unified manner. In Figure 13, we use CLASS A, IS and CG benchmarks. For IS, IBA only with 4 nodes takes 2.09 seconds, only GigE takes 1.90 seconds. MN-uDAPL is able to reduce the time taken to 1.75 seconds, which is an improvement of 8% from GigE only and 17% from IBA case only. A respective improvements of 20% and 9% are seen for the CG application kernel. Figure 14, shows the performance comparisons for FT and MG benchmarks. We notice that the application time does not improve much with respect to the network. However, a slight improvement in performance is shown by using MN-uDAPL than GigE device only.

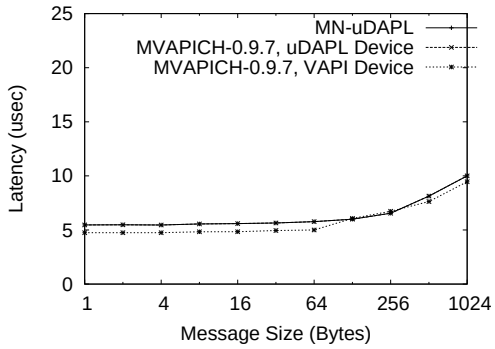

Figure 9. Latency Overhead for uDAPL and VAPI based MPI

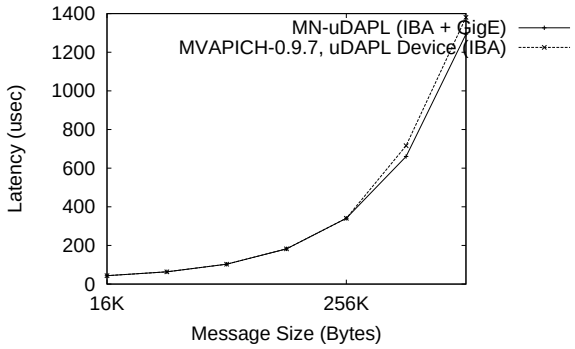

Figure 10. Large Message Latency

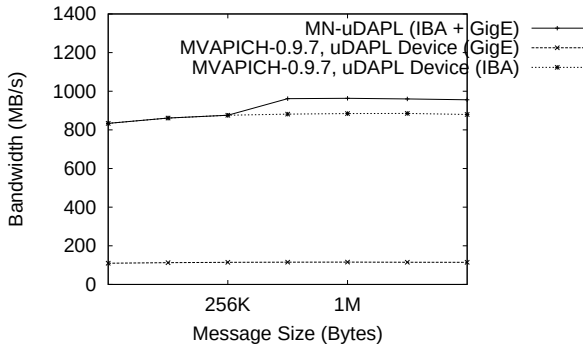

Figure 11. Uni-directional Bandwidth Comparison

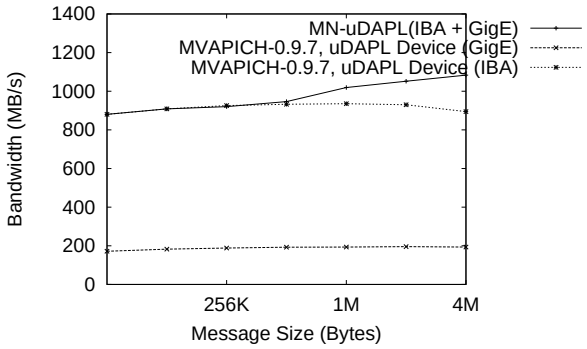

Figure 12. Bi-directional Bandwidth Comparison

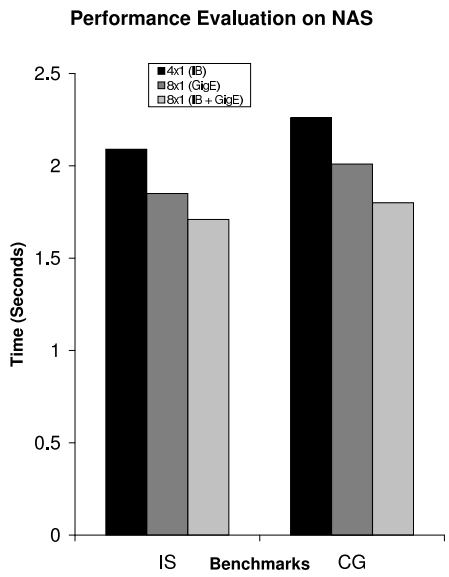

Figure 13. Performance Evaluation of IS and CG NAS Parallel Benchmarks, Class A

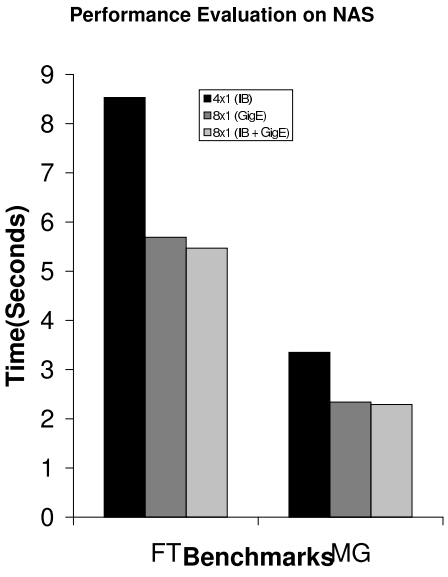

Figure 14. Performance Evaluation of FT and MG NAS Parallel Benchmarks, Class A

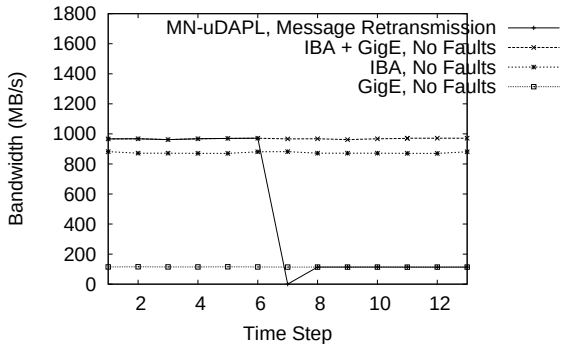

Figure 15. Uni-directional Bandwidth Comparison for Fault Tolerant Schemes, IBA Path Fails

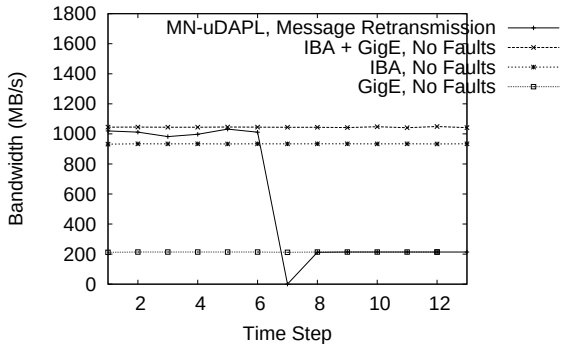

Figure 16. Bidirectional Bandwidth Comparison for Fault Tolerant Schemes, IBA Path Fails

5.4 Performance Evaluation with Network Faults

Figure 15, 16, 17, 18 show the results for the cases when network fault occur in the system. The comparisons are being shown for the message re-transmission scheme with the ideal case, when the same test is ran with no network faults. In order to show these results, we let the OSU Latency test report bandwidth at each iteration for a large number of iterations for a message size of 1 MB. The point of failure is reported as the middle point on the x-axis.

Figure 15 and 16 show the uni-directional and bi-directional bandwidth achievable, when IBA path fails during the communication. The message-retransmission scheme achieves the peak bandwidth as shown in the previous sections. However, at the point of failure, the re-transmission scheme almost achieves no-bandwidth due to multiple re-transmissions which occur at this point, before the DMA engine concludes the un-reachability of the destination process, and puts a failed CQE into the corresponding interconnect's completion queue. At this point, only GigE path is available. As can be noted from the graphs, our scheme incurs no overhead in providing the peak bandwidth. Figure 17 and 18 show a similar trend, the difference being the failure of the GigE path.

Figure 21 and 22 present the results for uni-directional bandwidth when running experiments in the configuration A. At the point 8 in the graph, both GigE and the IBA path fail and hence a network partition occurs in the system. At this point, the application hangs and waits for one of the connection paths to come up. The path recovery and network partition handling module generates an asynchronous thread and waits for the *connection* events from other process. After re-connection, the processes are able to achieve the peak uni-directional bandwidth which is achievable with GigE. At a later point, when the IBA path is available, we are able to achieve the peak bandwidth achievable in the presence of no-faults. Figure 22 shows a similar trend, however in this case the IBA path comes back earlier than the GigE path. However, in this case also we are able to achieve the peak uni-directional bandwidth achievable, when no network faults occur in the system.

Figure 19 and 20 present the results, when network paths fail at the beginning of the application itself. We notice from the figures that the performance degradation is negligible in comparison to the case 8x1 case, where only GigE is used for communication. This shows that the overhead of the message re-transmission module, generating an asynchronous thread for communication etc. incurs very low overhead on the communication performance.

6 Conclusions and Future Work

In this paper, we have designed a network fault tolerant MPI using uDAPL interface, making this design portable for existing and upcoming interconnects. Our design has provided failover to available paths, asynchronous recovery of the previous failed paths and recovery from network partitions without application restart. In addition, the design is able to handle network heterogeneity, making it suitable for the current state of the art clusters. To achieve these goals, we have designed low overhead *completion filter and error-detection*, *message (re)-transmission* and *path recovery and network partition handling* modules which perform completion filter and detection, (re)-transmission and recovery from network partitions respectively. We have implemented our design and evaluated it with micro-benchmarks and applications. Our performance evaluation have shown that the proposed design provides significant performance benefits to both homogeneous and heterogeneous clusters. Experiments also reveal that network fault tolerance modules incur very low overhead and provide optimal performance in wake of network failures for simple MPI micro-benchmarks and applications. In addition, in the absence

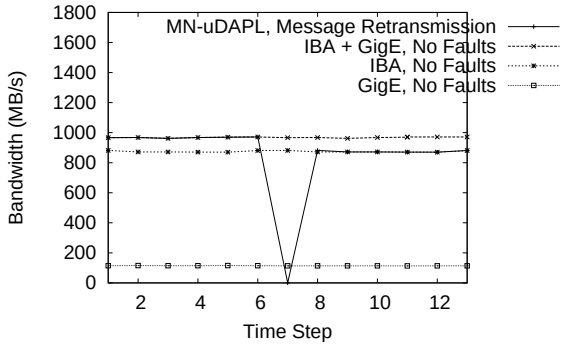

Figure 17. Uni-directional Bandwidth Comparison for Fault Tolerant Schemes, GigE Path Fails

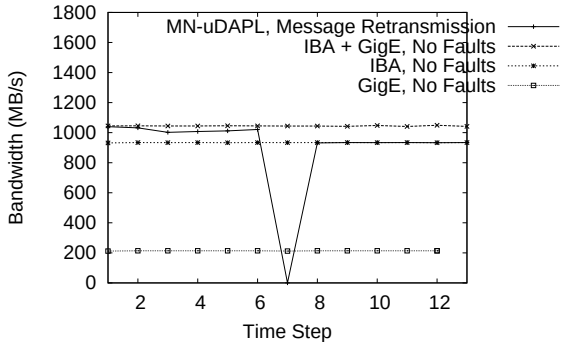

Figure 18. Bi-directional Bandwidth Comparison for Fault Tolerant Schemes, GigE Path Fails

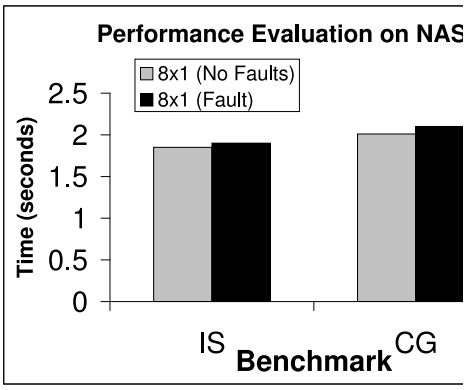

Figure 19. Performance Comparison on NAS Benchmarks, When the IBA Path Fails on First Message Transmission

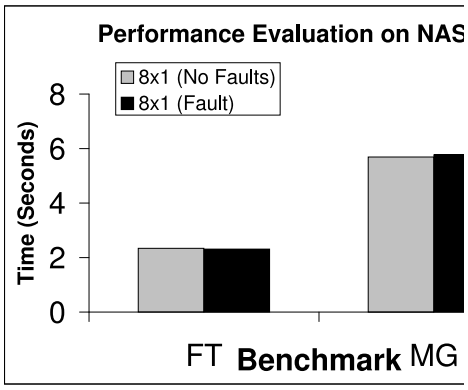

Figure 20. Performance Comparison on NAS Benchmarks, When the IBA Path Fails on First Message Transmission

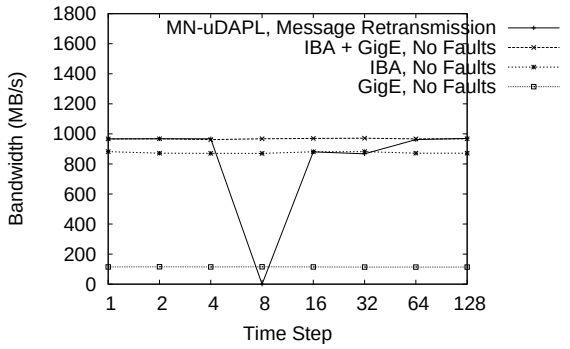

Figure 21. Uni-directional Bandwidth Comparison for Fault Tolerant Schemes with Network Partition

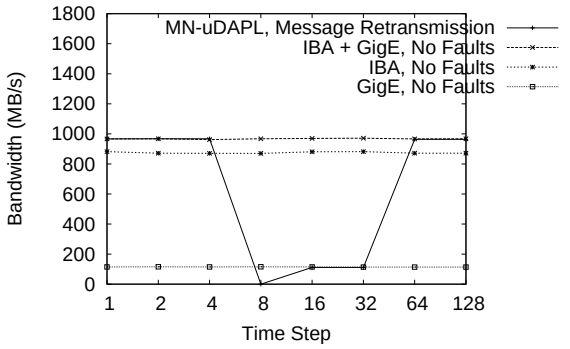

Figure 22. Bi-directional Bandwidth Comparison for Fault Tolerant Schemes with Network Partition

of such failures, using a heterogenous 8x1configuration of IBA and Ammasso-GigE, we have been able to improve the performance of NAS Parallel Benchmarks by 10-15% for different benchmarks. For simple micro-benchmarks, we have been able to improve the throughput by 15-20% for uni-directional and bi-directional bandwidth tests. Even though, the evaluation in the paper has been done using InfiniBand and Ammasso-GigE, there are emerging interconnects, which plan to support uDAPL interface and are not available in market commercially. The proposed design is generic and capable of supporting any interconnect with uDAPL interface.

In future, we plan to study hardware level mechanisms provided by RDMA-enabled interconnects for fault tolerance. In addition, we plan to handle more difficult cases of network heterogeneity, in which the presence of a common network interface card for all nodes is not mandatory. We also plan to study the impact of these designs on large scale clusters at application level.

References

[1] . Open MPI Project. http://www.open-mpi.org .

[2] *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*. IEEE Computer Society, 2004.

[3] L. Chai and R. Noronha and P. Gupta and G. Brown and D. K. Panda. Designing a Portable MPI-2 over Modern Interconnects Using uDAPL Interface., August 2005.

[4] Ammasso, Inc. The Ammasso 1100 High Performance Ethernet Adapter User Guide. http://www.ammasso.com/amso1100_usersguide.pdf, February 2005.

[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. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[6] M. Banikazemi, R. K. Govindaraju, R. Blackmore, and D. K. Panda. MPI-LAPI: An Efficeint Implementation of MPI for IBM RS/6000 SP Systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 1081–1093, October 2001.

[7] D. Buntinas, G. Mercier, and W. Gropp. The design and evaluation of Nemesis, a scalable low-latency message-passing communication subsystem. Technical Report ANL/MCS-TM-292, Argonne National Laboratory, 2005.

[8] DAT Collaborative. uDAPL and kDAPL API Specification V1.0, June 2002.

[9] 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. *Int. J. Parallel Program.*, 31(4):285–303, 2003.

[10] 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. *Int. J. Parallel Program.*, 31(4):285–303, 2003.

[11] Infiniband Trade Association. http://www.infinibandta.org/.

[12] Lawrence Livermore National Laboratory. MVICH: MPI for Virtual Interface Architecture, August 2001.

[13] J. Liu, A. Vishnu, and D. K. Panda. Building multirail infiniband clusters: Mpi-level design and performance evaluation. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 33, Washington, DC, USA, 2004. IEEE Computer Society.

[14] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *17th Annual ACM International Conference on Supercomputing (ICS '03)*, June 2003.

[15] 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, April 2006.

[16] OpenIB.org. http://www.openib.org/.

[17] S. Pakin and A. Pant. VMI 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management.

[18] A. Vishnu, G. Santhanaraman, W. Huang, H. W.Jin, and D. K. Panda. Supporting mpi-2 one sided communication on multirail infiniband clusters: Design challenges and performance evaluation. In *IEEE/ACM International Conference High Performance Computing*, 2005.