An Efficient Hardware-Software Approach to Network Fault Tolerance with InfiniBand

Abhinav Vishnu #1, Manojkumar Krishnan #2 and Dhabaleswar K. Panda *3

# *Pacific NorthWest National Lab*
902 Battelle Blvd, Richland, WA 99352
* *Department of Computer Science and Engineering*
The Ohio State University,
Columbus, OH 43210
1 abhinav.vishnu@pnl.gov
2 manoj@pnl.gov
3 panda@cse.ohio-state.edu

***Abstract*—In the last decade or so, clusters have observed a tremendous rise in popularity due to excellent price to performance ratio. A variety of Interconnects have been proposed during this period, with InfiniBand leading the way due to its high performance and open standard. Increasing size of the InfiniBand clusters has reduced the mean time between failures of various components of these clusters tremendously. In this paper, we specifically focus on the network component failure and propose a hybrid hardware-software approach to handling network faults. The hybrid approach leverages the user-transparent network fault detection and recovery using Automatic Path Migration (APM), and the software approach is used in the wake of APM failure. Using Global Arrays as the programming model, we implement this approach with Aggregate Remote Memory Copy Interface (ARMCI), the runtime system of Global Arrays. We evaluate our approach using various benchmarks (siosi7, pentane, h2o7 and siosi3) with NWChem, a very popular *ab initio* quantum chemistry application. Using the proposed approach, the applications run to completion without restart on emulated network faults and acceptable overhead for benchmarks executing for a longer period of time.**

I. Introduction

The computational needs of today's scientific scientific applications has led to the augmentation of high performance computing. Combining commercial off the shelf processors with commodity interconnects has led to cluster computing [1], a very effective methodology for achieving excellent price-to-performance ratio. As the commodity processors continue to grow, commodity interconnects such as Myrinet [2], Quadrics [3], and InfiniBand [4] have been introduced to combine these commodity processors. As reflected by the TOP500 [5] rankings, InfiniBand in particular has been observing wide acceptance due to its high performance and open standard, with 28% of the systems using InfiniBand as their interconnect. The current largest InfiniBand cluster uses more than 60000 processor cores at TACC [6], and larger scale systems are being planned for the near future. [1]

[1]This work is supported by Computational Science and Mathematics Division program of the Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, with a contract for Pacific NorthWest National Lab, operate by Battelle.

The increasing scale has tremendously reduced the mean time between failures (MTBF) of various components of these clusters. In this paper, we specifically focus on the network component failure, which includes the failure of network cables and switches. Failure of these network components breaks the existing path of communication between application processes, resulting in the execution failure of the application and requiring re-execution. In our previous work, we have designed various programming model independent modules based on InfiniBand Automatic Path Migration [7]. This approach requires an alternate path to be specified statically, which is used as an escape route, when the primary path fails. This approach performs well in the presence of a healthy alternate path. it is inadequate for unhealthy alternate path scenarios. Even with popular Constant Bisection Bandwidth (CBB) Fat Tree topology, it is difficult to find completely non-intersecting paths. The situation is further exacerbated with over-subscribed Fat Tree topologies in systems like Atlas [8] and Chinook [9]. Clearly, APM approach alone is not sufficient for handling all network faults in the system.

To handle this limitation, we propose a hybrid hardware-software approach (Hybrid - InfiniBand Network Fault Tolerance (IBNFT)) for handling network faults with InfiniBand. The hybrid approach leverages the user-transparent network fault detection and recovery using APM as much as possible. The failure in hardware approach results in notification being sent to the software based network fault tolerance module. The hardware and software based approaches are referred to as Hardware-IBNFT and Software-IBNFT, respectively. Compared to the previously proposed approach, the Software-IBNFT approach leverages the early network fault detection and a scalable out-of-band connection management mechanism for connection re-establishment. Using Global Arrays [10], a widely used shared memory programming model, we implement our hybrid approach using Aggregate Remote Memory Copy Interface (ARMCI) [11], the run time system of Global Arrays. We evaluate our Hybrid-IBNFT using up to 1024 tasks and multiple benchmarks (siosi7, pentane, h2o7 and siosi3) with NWChem [12], a very popular *ab initio* quantum chemistry application. Using the proposed approach, the application runs to completion for different benchmarks without restart on emulated network faults and acceptable overhead for benchmarks executing for a longer period of time. To the best of our knowledge, this is the first design and implementation of a hybrid hardware-software approach for network fault tolerance with InfiniBand.

The rest of the paper is organized as follows. In section II, we present the background of our work. We present the motivation of our work in section III. In section IV, we present the design of Hybrid-IBNFT. The performance evaluation of the approach implemented is presented in section V. We present the related work in section VI. We conclude and present our future directions in section VII. We begin with the description of the background work.

II. Background

In this section, we present the background of our work. We begin with an introduction to InfiniBand [4] and the state transitions associated with a Queue Pair (QP). We also provide a brief introduction to Automatic Path Migration, Global Arrays [10] and Aggregate Remote Memory Copy Interface (ARMCI) [11].

A. Overview of InfiniBand and QP Transition States

The InfiniBand Architecture (IBA) [4] 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. 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 and unreliable datagram model. In reliable connection 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.

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 transitioned to the INIT state by invoking modify_qp function. The modify_qp function is provided by the access layer of InfiniBand [4]. 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 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 transitioned 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 transitioned back to the RESET state and the above-mentioned transition sequence (RESET-RTS) needs to be re-executed. The RTS-SQD transition is an important mechanism to ensure that the outstanding data requests have completed. After a QP is in SQD state, it can be transitioned to RTS state directly to allow messages to be sent/received from the communicating pair.

1) Data Transfer Requests and Completion Queue: The data transfer requests are initiated by posting a send/receive descriptor to the send/receive queue of the QP. Once the request is completed, an entry is generated at the completion queue. The completion queue entry can be checked to see if the request was a success. We use this InfiniBand mechanism in the design of Softare-IBNFT.

2) Unreliable Datagram: The unreliable datagram model uses connection-less model for communication. Each process creates a QP for every other process in the job. The underlying layer does not guarantee data delivery, however, it does guarantee, maximum once delivery of data with checksum. In this paper, we use the unreliable datagram as the out-of-band mechanism for connection re-establishment.

3) LID Mask Count and Shared Receive Queue: LID is a local identifier, which is used for identifying network ports in a InfiniBand switch. Each port can have more than one LID depending up on the value of LID Mask Count (LMC). LMC is an InfiniBand mechanism to provide multi-pathing. InfiniBand defines a *subnet manager*, which is responsible for finding multiple routes and specifying the InfiniBand routing tables. LMC value 0-7 can be specified by the user and the subnet manager uses this value to create a maximum of 128 paths between any pair of nodes. In practice, since most InfiniBand topologies are a variant of Fat Tree [13], multiple paths typically span multiple switches. The subnet manager typically creates multiple paths using as many links as possible in the subnet [14]. We leverage this property for defining an alternate path for APM.

We presented the queues associated with a QP above. Shared receive queue is a mechanism by which multiple QPs can share the same receive queue. This leads to a much better memory and buffer utilization compared to the individual receive queues. Sur et al. have presented a design for MPI, which achieves much better memory utilization than the previously proposed schemes [15].

B. 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

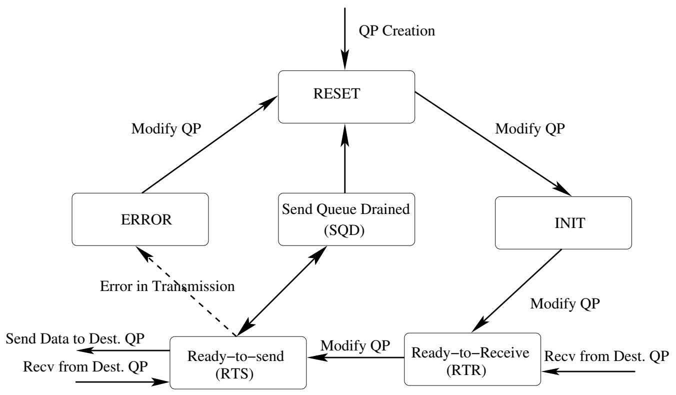

Fig. 1. QP Communication State Diagram

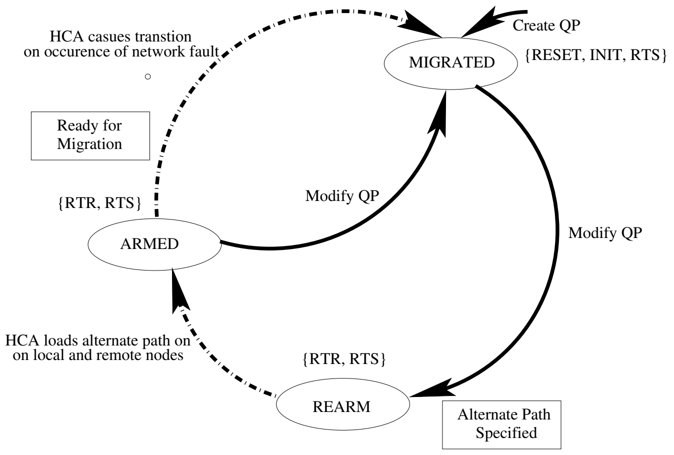

Fig. 2. QP Path Migration State Diagram

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.

APM defines a concept of alternate path, which is used as an escape route should an error occur 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, 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.

C. Global Arrays and ARMCI

"The Global Arrays programming model provides an efficient and portable "shared-memory" programming interface for distributed-memory computers. Each process in a MIMD parallel program can asynchronously access logical blocks of physically distributed dense multi-dimensional arrays, without need for explicit cooperation by other processes. Unlike other shared-memory environments, the GA model exposes to the programmer the non-uniform memory access (NUMA) characteristics of the high performance computers and acknowledges that access to a remote portion of the shared data is slower than to the local portion. The locality information for the shared data is available, and a direct access to the local portions of shared data is provided." [10]. Global Arrays uses Aggregate Remote Memory Copy Interface (ARMCI) [16], as the runtime system for communication.

The purpose of the ARMCI library is to provide a general-purpose, efficient, and widely portable remote memory access (RMA) operations (one-sided communication) optimized for contiguous and non-contiguous (strided, scatter/gather, I/O vector) data transfers. In addition, ARMCI includes a set of atomic and mutual exclusion operations. ARMCI exploits native network communication interfaces and system resources (such as shared memory) to achieve the best possible performance of the remote memory access/one-sided communication. It exploits high-performance network protocols on clustered systems. Optimized implementations of ARMCI are available for the Cray Portals, Myrinet (GM and MX), Quadrics, Giganet (VIA) and InfiniBand (using OpenFabrics and Mellanox Verbs API). It is also available for leadership class machines including Cray XT4 and BlueGene/P.

III. Motivation

As discussed in the previous section, APM provides user-transparent network fault detection and failover. Hence, applications performing long computation phases can benefit from this feature immensely. The software based approach would discover the failure of data transfer requests at the end of the computation phase. Hence, it is almost always beneficial to use the APM as much as possible.

Figure 3 shows a popular 144-port switch topology for InfiniBand clusters [17], [18]. In this topology, there are twelve completely disjoint paths between any pair of nodes, which are connected to different leaf blocks. In our previous work [14], we have shown that with LID Mask Count (LMC) mechanism provided by InfiniBand, multiple disjoint paths are being configured by the InfiniBand subnet manager. We noticed that the subnet manager is able to use different spine blocks completely for creating alternate paths. As an example, to communicate between a node attached to block_1 and block_2, the subnet manager is able to define paths traversing blocks 1-13-2, 1-14-2 . . . , (not necessarily in that order) and so on. Each of these paths can be used by specifying an identifier (PathID) during QP creation. We note that these paths are completely disjoint above the leaf blocks. Hence, faults occurring in the

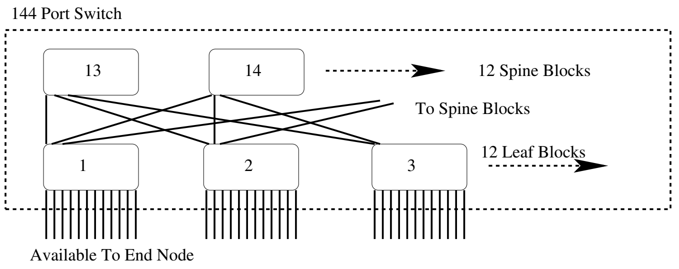

Fig. 3. 144-Port InfiniBand Switch Topology

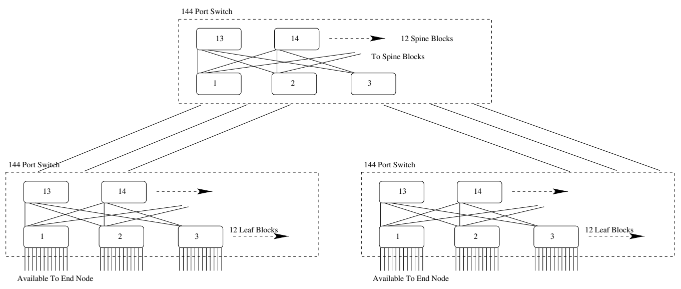

Fig. 4. Oversubscribed Fat Tree InfiniBand Topology Variant

spine blocks and links connecting them to leaf blocks can be handled by APM by specifying an alternate path with a different PathID than the primary path.

Using the previously shown 144-port switch as the building blocks, we can realize a larger multi-stage oversubscribed Fat Tree topology as shown in the Figure 4. We have not shown the specific inter-connectivity between the leaf 144-port switches and the spine 144-port switches. A possible realization can be achieved by connecting the output ports available to the end nodes of the leaf 144-port switches, to the spine switches. This variant is used in Chinook [9], where eight output ports of the leaf switches are available to the end nodes and the other four output ports are connected to spine switches. Other variants can be achieved by connecting the spine switch ports of the leaf 144-ports switches. With these topology variants, it is very difficult and sometimes impossible to find completely disjoint paths between end nodes. This problem is exacerbated with increasing number of nodes, and increasing oversubscription of Fat Trees.

With an inability to find completely disjoint primary and alternate path for APM, the chances of APM failure on a network fault increase significantly, particularly for increased oversubscription of Fat Trees. To the best of our knowledge, most large scale InfiniBand clusters use the variant of the topology discussed in Figure 4. Given the benefits of APM described above, it is important to design a software based approach in conjunction with APM for network fault tolerance. In the next section, we present the design of Hybrid-IBNFT, which achieves this objective.

IV. Hybrid-IBNFT Design

In this section, we present the design of our Hybrid-InfiniBand Network Fault Tolerance (Hybrid-IBNFT). We begin with the description of Automatic Path Migration based approach [7]. This approach is referred to as Hardware-IBNFT for the rest of the paper. We also present the design of the Software-IBNFT approach. Figure 5 shows the overall design of Hybrid-IBNFT. The Hybrid-IBNFT interfaces between the InfiniBand Access Layer and ARMCI. The interfaces between different layers are bi-directional to indicate the data transfer requests being sent to the InfiniBand Access layer and completions being sent to the Global Arrays layer.

A. *Hardware-IBNFT*

The Hardware-IBNFT uses an Alternate Path Specification Module, Path Loading Request Module and Path Migration Module for providing network fault tolerance [7]. The Alternate Path Specification Module allows us to specify an alternate path. The current framework allows to specify an alternate port/route as an alternate path. The InfiniBand access layer does not allow an alternate adapter to be used as an alternate path. The Path Loading Request Module is used to request the loading of a path in APM state machine, as discussed in section II. Since invoking Path Loading Request Module is expensive [7], this module is typically invoked immediately after invoking Alternate Path Specification Module. The conventional wisdom says to invoke this module as late as possible. We intend to study the impact of this delay in future, when the overhead of incurring Path Loading Request Module is reduced. The interaction of these modules is presented in Figure 6.

The Path Migration Module can be optionally used by user to manually transition an alternate path as the primary path for data transfer. We do not use this module in this paper. With Hardware-IBNFT, the primary and alternate pathIDs used by a QP can be represented by using consecutive values in a round round-robin fashion with respect to the total number of paths available in the network.

At the occurence of a network fault, InfiniBand Access layer generates an event at each of the end nodes of the QP, indicating the successful completion of the path migration. At the occurence of this event, we query the current attributes of the QP, and invoke the alternate path specification module and the path migration module to specify the alternate path and request the transition. However, as discussed in section III, Hardware-IBNFT works successfully only if the alternate path is in healthy state as well. In the next section, we present the design of Software-IBNFT, which is initiated with the failure of Hardware-IBNFT.

B. *Software-IBNFT*

In this section, we present the design of the Software-IBNFT. We present the mechanisms provided by InfiniBand for an Early Detection of Network Fault and design for Scalable Out-of-Band Connection Manager, which Software-IBNFT uses for connection re-establishment. We also discuss

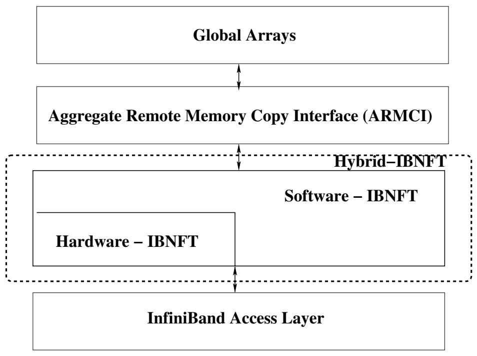

Fig. 5. Overall Design of Hybrid-IBNFT and Interface with ARMCI and InfiniBand Access Layer

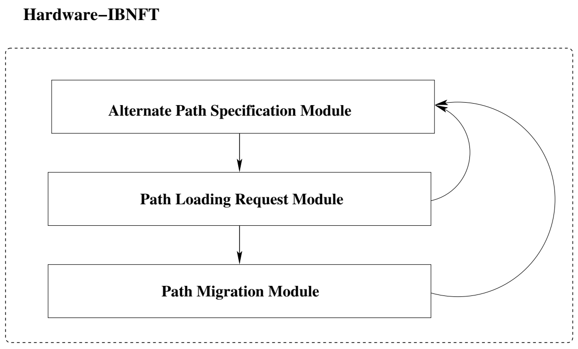
Hardware–IBNFT
Alternate Path Specification Module
Path Loading Request Module
Path Migration Module

Fig. 6. Overall Design for Hardware-IBNFT [7]

the framework for putting these mechanisms altogether in order to provide Software-IBNFT.

1) Early Detection of Network Fault: InfiniBand provides mechanisms to register event notification handler with the adapter. Using this event handler, the registered client can perform corrective actions, on the occurence of an event. Events corresponding to the state of the path migration, and QP state are generated, whenever possible. We leverage this event generation mechanism provided by InfiniBand for early detection of network fault(s). As discussed above, the path migration success generates an event marking success of the path migration. Similarly, an event is generated on the failure of path migration. We leverage this event for early detection of network fault.

There are certain cases, when the failure event in not generated, particularly when the path loading request module is active, however the alternate path has not been loaded completely in the APM state machine. An event is also generated on the receiver side, when the data transfer has been initiated and the network fault occurs during the transfer. We also use this event as much as possible for early detection of a network fault. These features are not available with uDAPL based network fault tolerance work done in our previous paper [19]. Clearly, this limits us to detect the network faults only at the communication progress by the main thread.

The event generation mechanism requested only increases with the number of active communication instances, and not with the number of outstanding data requests on these tasks. Hence, the overhead observed only increases with the number of active communication processes.

2) Scalable Out-of-Band Connection Manager: With Software-IBNFT, we use an unreliable datagram based connection manager, for connection re-establishment. This out-of-band connection manager is setup at the ARMCI Initialization phase. The connection manager scales very well, since its resource usage does not increase linearly with the increasing number of processes in a job. We use this connection manager to initiate connection re-establishment protocol, at the detection of the network fault. To ensure a timely response for connection-reestablishment, we implement the event listener portion of the connection manager as a separate thread blocking on an event. This thread is activated only in the presence of a network fault handled unsuccessfully by Hardware-IBNFT.

An important property of reliable connection transport is that the data transfer request to a QP in error state results in transition of the initiator QP to be in the error state. In our previous work with uDAPL [19], this can result in multiple re-tries before the connection can be re-established successfully. However, with unreliable datagram, the QP does not transition to an error state on occurence of a network fault. This results in a very efficient connection re-establishment protocol, compared to the previously proposed approach.

C. Putting It All Together

Figure 7 shows the overall design of Software-IBNFT. It has components which are responssible for detection of error by checking the completion queue and the early fault detection mechanisms described above. This component is referred to as Network Fault Detection module. The network fault detection module does not add any extra overhead, since the completion queue is checked for notifying the ARMCI layer of the completion of data transfer requests.

The message re-transmission module is responsible for re-issue of previously failed data transfer requests. We maintain a queue of the failed data transfers and re-issue them once the QP has been re-established for communication. Figure 8 shows the overall communication protocol for re-establishment with software-IBNFT.

We use process A and B to explain the communication re-establishment protocol. On the occurence of a network fault, the QP on process A is automatically transitioned to ERROR state by the hardware. At this point, the QP is transitioned to the SQD state. This ensures that the all the previously posted data requests on the QP have been posted to the completion queue. The QP is recycled to the RTS state at this point, and a request (REQ) for connection re-establishment is sent to the process B. This request generates an event at process B. This process in turn transitions its QP to the SQD state and queries the state of the QP. The ERROR state of the QP requires a re-cycling of the QP and reply being sent to process A. Once process A receives the REP, it sends an ACK to the process

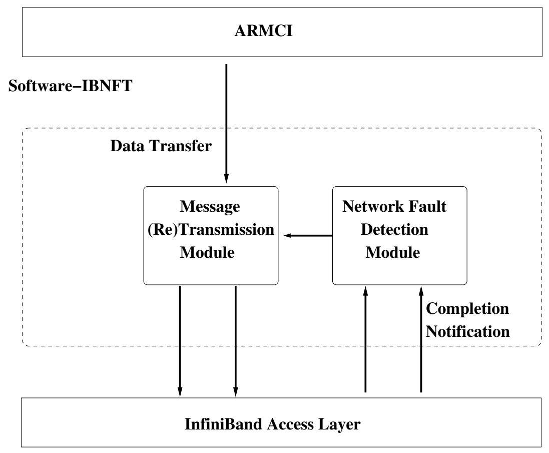

Fig. 7. Overall Design for Software-IBNFT

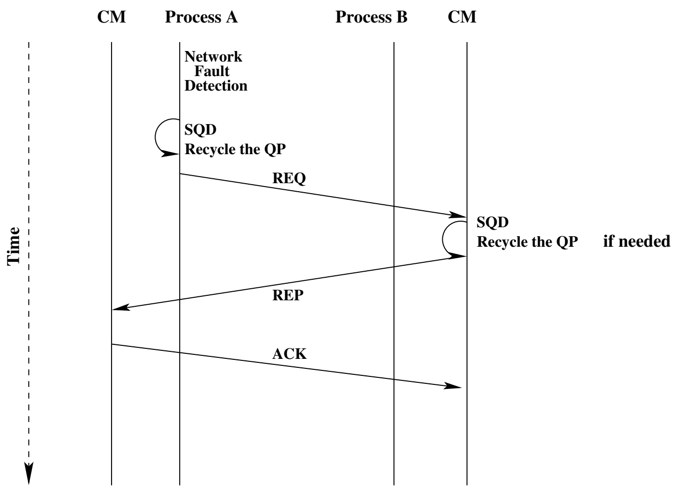

Fig. 8. Communication Protocol Re-establishment with Software-IBNFT

B. Note that some of these control messages are sent by the asynchronous thread.

We use a timeout based mechanism to re-send the messages of the communication protocol. A timer-pop mechanism is used to re-send the messages, which have not been acknowledged after a timeout. Duplicate messages are silently ignored in the communication re-establishment protocol.

D. *Detailed Design Issues*

In this section, we discuss the implementation details of Hybrid-IBNFT. We focus on the usage of Shared Receive Queue mechanism provided by InfiniBand and a need for ordering requirements for Out-of-Band connection manager.

1) Shared Receive Queue: Shared Receive Queue provides memory utilization benefits, since individual queues are not required for communication. This property has an important consequence. At the point of the QP transition to error, the associated SRQ does not transition to the error state. For normal receive queues, each of the previously posted receive buffers are flushed to the receive completion queue. With SRQ, only the buffers which are currently active in communication are reported to completion queue with error.

2) Ordering Requirements for Out-of-Band Connection Manager: In order to different a network fault from the previous network fault between the same pair of processes, a sequence ID is maintained for each pair of processes. This is also required, since unreliable datagram messages are not guaranteed to be delivered in-order on the receiving side.

V. Performance Evaluation

In this section, we present the performance evaluation of Hybrid-IBNFT. We compare this with the latest release of Global Arrays, which we refer to as ”Original” for the rest of the section. We use simple benchmarks to understand the overheads incurred by our new approach in the absence of faults. We also discuss the methodology of emulating network faults, and the behavior of Hybrid-IBNFT in the presence of emulated network faults. We follow this with the study of impact of network faults using NWChem with multiple benchmarks and different number of processors. We have used Chinook [9], an AMD Barcelona based SuperComputer at Pacific Northwest National Lab as the experimental tested for our evaluation.

A. *Experimental Testbed*

Chinook [9] is a 160 TFlops system that consists of 2310 HP DL185 nodes with dual socket, 64-bit, Quad-core AMD 2.2 GHz Barcelona Processors. Each node has 32 Gbytes of memory and 365 Gbytes of local disk space. Communication between the nodes is obtained using an InfiniBand interconnect from Voltaire [17] Switches and Mellanox [18] Adapters. The system runs a version of Linux based on Red Hat Linux Advanced Server. A global 297 Tbyte SFS file system is available to all the nodes.

B. *Methodology for Emulating Network Faults*

To evaluate the performance of Hybrid-IBNFT, the best case would be to have physical access to the supercomputer and the privilege to unplug a cable during the communication, and observe the behavior. However, such privilege is prohibitive for systems like Chinook [9], which continually serve the needs of scientists at PNNL and worldwide.

Hence we design a software based mechanism to inject a network fault, which would result in failure of communication, and initiate the fault recovery mechanism. During the QP setup phase, we selectively specify incorrect destination LID for the destination QP. As a result, any data requests on the QP result in error, initiating the network fault recovery using Hybrid-IBNFT. The setup at Chinook does not use LMC yet, which results in only a single path of communication between any pair of processes. As a result, the emulated network fault results in APM failure. We are working to remove this limitation, since usage of LMC does not only benefit network fault tolerance, it also benefits hot-spot avoidance, as presented in our previous work [14]. We do not disable the Hardware-IBNFT path in our implementation. Clearly, this limitation

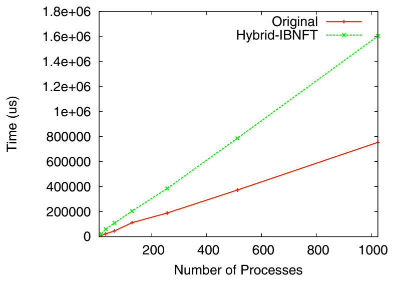

Fig. 9. Queue Pair Transistion Time

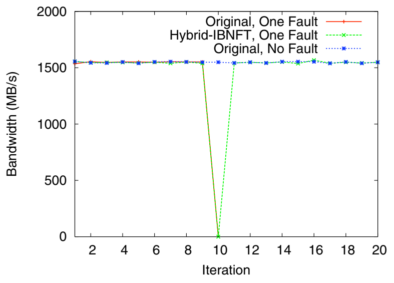

Fig. 10. ARMCI Get Unidirectional Bandwidth

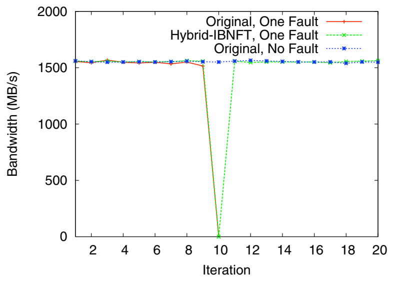

Fig. 11. ARMCI Put Unidirectional Bandwidth

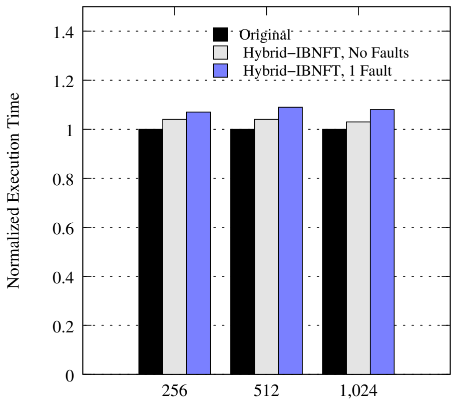

Fig. 12. NWChem, h2o7 Benchmark

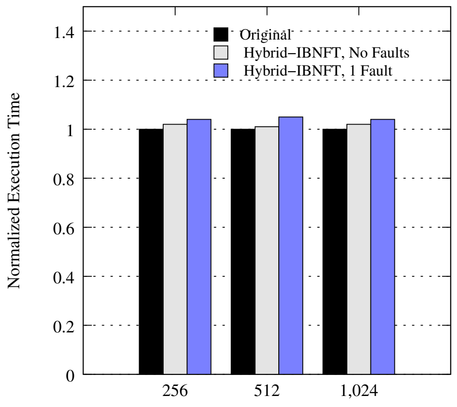

Fig. 13. NWChem, pentane Benchmark

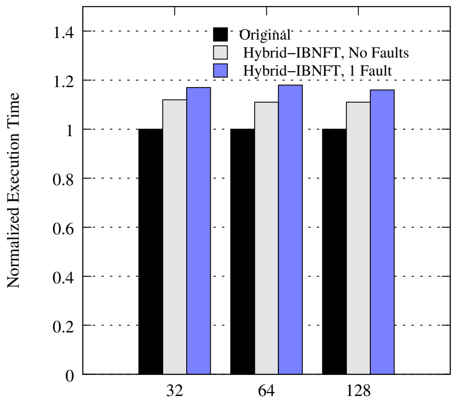

Fig. 14. NWChem, siosi3 Benchmark

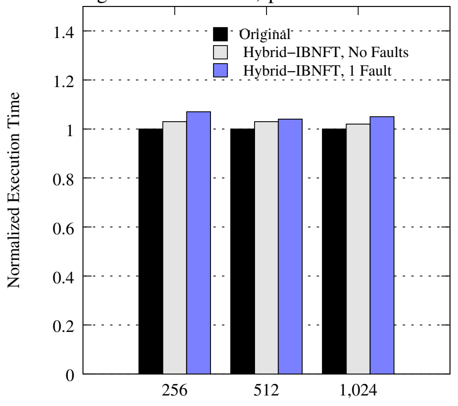

Fig. 15. NWChem, siosi7 Benchmark

only results in worse performance than if Hardware-IBNFT could be used successfully.

C. Performance Evaluation with Microbenchmarks

In this section, we present the results with microbenchmarks. We measure the increment in the QP state transition time, as a result of using Hardware-IBNFT. This penalty is incurred during the QP creation, during the execution of the Software-IBNFT recovery and successful completion of Hardware-IBNFT recovery. In addition, we use simple ARMCI blocking Get (ARMCI_Get) and blocking Put (ARMCI_Put) based microbenchmarks, which report the bandwidth observed at every iteration and study its performance in presence of emulated network faults.

Figure 9 shows the time taken in QP transition comparing the Original implementation with Hybrid-NFT. We have used up to 1024 tasks for the comparison. Leaving small task count aside, we notice that the QP transition follows a near linear trend with increasing number of processes. There is a significant overhead observed in transition time using Hybrid-IBNFT. This cost is incurred every time an alternate path is loaded. However, for long running applications, this cost would be amortized over the length of the application execution.

Figure 10 shows the results for a unidirectional Get bandwidth test at ARMCI level. We compare the results of the Original implementation with Hybrid-IBNFT using no or one-fault. The test reports the result at every iteration. For One

Fault case, we take a snapshot of the result and use the point of failure as the center point in the figure. We see that the Original, No Fault case shows close to 1500 MB/s of unidirectional bandwidth. Obviously, the benchmark does not run to completion in the case of Original, One Fault. The Hybrid-IBNFT approach shows a very interesting trend at the point of failure. At the failure, we observe almost 0 MB/s bandwidth. The primary reason is that multiple re-tries for data transfer are performed before APM transition is initiated, which results in a failure on our TestBed. We also use a high value of timeout, which is important to reduce the false positives of communication failure on large scale clusters. The timeout typically dominates the total overhead at communication failure.

We also observe, that there is no overhead incurred by Hybrid, IBNFT at non-failure iterations. Similar trends in performance are observed for different implementations using ARMCI Put unidirectional bandwidth, as shown in Figure 11.

D. Performance Evaluation with NWChem

In this section, we present the normalized execution time of Hybrid-IBNFT with NWchem [12], using various benchmarks including h207, pentane, siosi3 and siosi7. We use up to 1024 processors for the performance evaluation. For each of the benchmarks, we compare the results of the Original case with Hybrid-IBNFT, No Fault and Hybrid-IBNFT, 1 Fault case.

Figure 12 shows the results for h2o7 benchmark for 256, 512 and 1024 processors respectively. Compared to the original implementation, there is a slight overhead incurred by Hybrid-IBNFT, no fault case. This overhead is due to overhead incurred in APM transition, as presented in Figure 9. We observed that this overhead is linear with increasing number of processes. An overhead is also observed for creating an out-of-band connection manager. The overhead of creation is constant with increasing number of processes, due to the connectionless nature of the unreliable datagram transport of InfiniBand.

The Hybrid-IBNFT, 1 Fault case shows the results with one network fault. We observe that a slight overhead is observed for this case, since this case incurs the overhead of no-fault case, and the overheads of APM failure and multiple re-tries at the network layer, before software based re-transmission is initiated. However, the overall overhead for any of the processor counts is less than 10% with one network fault.

Figure 13 shows the results for h2o7 benchmark for 256, 512 and 1024 processors respectively. We observe an overhead of less than 5% for Hybrid-IBNFT case for each of the processor counts. An overhead of 5% is observed with 1 Fault case. For 1024 processors, the benchmark runs for 181 seconds.

Figure 14 shows the results for siosi3 benchmark for 32, 64 and 128 processors respectively. siosi3 is a smaller benchmark compared to the rest of the benchmarks used in this paper. We used this benchmark to understand the overhead for applications potentially running for a small period of time. The 32 processor run executes for 17 seconds. As shown in the figure, a considerable overhead is observed for No Fault and 1 Fault case with Hybrid-IBNFT. It can be concluded that Hybrid-IBNFT is targetted for applications running for a longer period of time.

Figure 15 shows the results for siosi7 benchmark for 256, 512 and 1024 processors respectively. This benchmark runs for 9 minutes and 14 seconds for 1024 processors. As this can be seen, the overhead observed is negligible for No Fault and 1 Fault case, compared to the Original execution of time.

Clearly, the inputs running for a longer execution time do not incur much overhead at the occurence of a network fault. We notice that the overhead is insignificant even for an application execution for 181 seconds. The systems at the current scale have much higher MTBF. Hence, the Hybrid-IBNFT provides an efficient solution for large scale clusters.

VI. Related Work

Providing Network Fault tolerance with high speed interconnnects has been studied by many researchers in last couple of years. Petrini et al. have provided mechanisms for network fault tolerance with Quadrics [3], [20]. With Quadrics, failover with multiple alternate paths are used before communicating processes are notified of the failure. Myrinet [2] uses a connectionless approach for communication, and hardware acknowledgement is not provided on data delivery. In our previous work, we have designed modules for network fault tolerance using Automatic Path Migration (APM) over InfiniBand [7]. However, this approach works fine, only if the alternate path is healthy. We have also worked on designing software based network fault tolerance approach with uDAPL based clusters [19]. However, uDAPL does not provide interface for hardware based support for APM, in addition to the limited reliable connection based data transfer. uDAPL does not provide support for Shared Receive Queue.

Aulwes et al. have provided a network fault tolerance with LA-MPIaulwes:europvm03. However, this approach primarily focusses on TCP based clusters, and does not leverage the hardware based APM with InfiniBand. It also does not leverage the completion queue semantics provided by high speed interconnects. Similarly, OpenMPI [21], provides supports for multiple networks and allows striping by linking these networks at the Byte Teansport Layer (BTL). OpenMPI is also not able to leverage the hardware based network fault tolerance provided by InfiniBand.

VII. Conclusions and Future Work

In this paper, we have proposed a hybrid hardware-software approach (Hybrid - InfiniBand Network Fault Tolerance (IB-NFT)) for handling network faults with InfiniBand. The hybrid approach has used the user-transparent fault detection and recovery using APM as much as possible. Compared to the previously proposed approached for software based network fault tolerance, this approach has leveraged the early network fault notification and a scalable out-of-band connection management mechanism for connection re-establishment. Using Global Arrays, a widely used shared memory programming model, we have implemented our hybrid approach using

Aggregate Remote Memory Copy Interface (ARMCI), the run time system of Global Arrays. We evaluated our Hybrid-IBNFT using up to 1024 tasks and multiple benchmarks (siosi7, pentane, h2o7 and siosi3) with NWChem [12], a very popular *ab initio* quantum chemistry application. Using the proposed approach, the applications executed to completion without restart on emulated network faults and acceptable overhead for datasets executing for a longer period of time. To the best of our knowledge, this is the first design and implementation of a hybrid hardware-software approach for network fault tolerance with InfiniBand.

We plan to evaluate Hybrid-IBNFT on larger processor counts and use LMC to evaluate the performance of hardware approach. We also plan to combine the hot-spot avoidance approach, as proposed in our previous work to decide the best alternate paths with APM. Performance evaluation with other emerging Global Arrays applications like sub-surface transport over multiple phases with Hybrid-IBNFT is also of an immediate interest to us.

References

[1] D. A. Patterson, D. E. Culler, and T. E. Anderson, “A Case for NOW (Networks of Workstations),” in *Principles of Distributed Computing*, 1995, pp. 17–28.

[2] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su, “Myrinet: A Gigabit-per-second Local Area Network,” *IEEE Micro*, vol. 15, no. 1, pp. 29–36, February 1995.

[3] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, “The Quadrics Network: High-Performance Clustering Technology,” *IEEE Micro*, vol. 22, no. 1, pp. 46–57, 2002.

[4] InfiniBand Trade Association, “InfiniBand Architecture Specification, Release 1.2,” October 2004.

[5] “TOP 500 Supercomputer Sites,” http://www.top500.org.

[6] “Texas Advanced Computing Center,” http://www.tacc.utexas.edu/.

[7] A. Vishnu, A. Mamidala, S. Narravula, and D. K. Panda, “Automatic Path Migration over InfiniBand: Early Experiences,” in *Proceedings of Third International Workshop on System Management Techniques, Processes, and Services, held in conjunction with IPDPS’07*, March 2007.

[8] “ATLAS SuperComputer, Lawrence Livermore National Lab,” http://computing.llnl.gov/.

[9] “Chinook SuperComputer, Environmental Molecular Science Lab, PNNL,” http://emsl.pnl.gov.

[10] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Aprà, “Advances, applications and performance of the global arrays shared memory programming toolkit,” *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 203–231, 2006.

[11] J. Nieplocha, V. Tipparaju, M. Krishnan, and D. K. Panda, “High Performance Remote Memory Access Communication: The ARMCI Approach,” *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 233–253, 2006.

[12] R. A. Kendall, E. Aprà, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. Ju, J. A. Nichols, J. Nieplocha, T. P. Straatsma, T. L. Windus, and A. T. Wong, “High Performance Computational Chemistry: An Overview of NWChem, A Distributed Parallel Application,” *Computer Physics Communications*, vol. 128, no. 1-2, pp. 260–283, June 2000.

[13] C. E. Leiserson, “Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing,” *IEEE Transactions on Computers*, vol. 34, no. 10, pp. 892–901, 1985.

[14] A. Vishnu, M. J. Koop, A. Moody, A. R. Mamidala, S. Narravula, and D. K. Panda, “Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective,” in *Cluster Computing and Grid*, 2007, pp. 479–486.

[15] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda, “Shared receive queue based scalable mpi design for infiniband clusters.” in *IPDPS*, 2006.

[16] J. Nieplocha and B. Carpenter, “ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-Time Systems,” vol. 1586, 1999. [Online]. Available: citeseer.nj.nec.com/nieplocha99armci.html

[17] “Voltaire Technologies,” http://www.voltaire.com/.

[18] “Mellanox Technologies,” http://www.mellanox.com/.

[19] A. Vishnu, P. Gupta, A. R. Mamidala, and D. K. Panda, “A Software Based Approach for Providing Network Fault Tolerance in Clusters with uDAPL Interface: MPI Level Design and Performance Evaluation,” in *SuperComputing*, 2006, pp. 85–96.

[20] F. Petrini, E. Frachtenberg, A. Hoisie, and S. Coll, “Performance Evaluation of the Quadrics Interconnection Network,” *Journal of Cluster Computing*, vol. 6, no. 2, pp. 125–142, April 2003.

[21] 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*, 2004, pp. 97–104.