Design and Evaluation of Network Topology-/Speed- Aware Broadcast Algorithms for InfiniBand Clusters*

H. Subramoni[1], K. Kandalla[1], J. Vienne[1], S. Sur[1], B. Barth[3], K. Tomko[2], R. McLay[3], K. Schulz[3], and D. K. Panda[1]

[1] *Department of Computer Science and Engineering,*
The Ohio State University
{subramon, kandalla, viennej, surs, panda}
@cse.ohio-state.edu

[2] *Ohio Supercomputing Center,*
Columbus, Ohio
{ktomko}
@osc.edu

[3] *Texas Advanced Computing Center,*
The University of Texas at Austin
{bbarth, mclay, karl}
@tacc.utexas.edu

***Abstract*—It is an established fact that the network topology can have an impact on the performance of scientific parallel applications. However, little work has been done to design an easy to use solution inside a communication library supporting a parallel programming model where the complexities of making the application performance network topology agnostic is hidden from the end user. Similarly, the rapid improvements in networking technology and speed is resulting in many commodity clusters becoming heterogeneous with respect to networking speed. For example, switches and adapters belonging to different generations (SDR - 8 Gbps, DDR - 16 Gbps and QDR - 36 Gbps speeds in InfiniBand) are integrated into a single system. This leads to an additional challenge to make the communication library aware of the performance implications of heterogeneous link speeds. Accordingly, the communication library can perform optimizations taking link speed into account. In this paper we propose a framework to automatically detect the topology and speed of an InfiniBand network and make it available to users through an easy to use interface. We also make design changes inside the MPI library to dynamically query this topology detection service and to form a topology model of the underlying network. Accordingly, we have re-designed the broadcast algorithm to take into account this network topology information and dynamically adapt the communication pattern to best fit the characteristics of the underlying network. To the best of our knowledge, this is the first such work for InfiniBand clusters. Our experimental results show that, for large homogeneous systems and large message sizes, we get up to 14% improvement in the latency of the broadcast operation using our proposed network topology-aware scheme over the default scheme at the micro-benchmark level. At the application level, the proposed framework delivers up to 8% improvement in total application run-time especially as job size scales up. The proposed network speed-aware algorithms are able to attain micro-benchmark performance on the heterogeneous SDR-DDR InfiniBand cluster to perform on par with runs on the DDR only portion of the cluster for small to medium sized messages. We also demonstrate that the network speed-aware algorithms perform 70% to 100% better than the naive algorithms when both are run on the heterogeneous SDR-DDR InfiniBand cluster.**

***Keywords*-Network Topology, InfiniBand, Topology Aware Collective Algorithms**

I. INTRODUCTION

Across various scientific domains, application scientists are constantly looking to push the envelope of their research by running large scale parallel jobs on supercomputing systems. The need to achieve high resolution results with smaller turn around times has been driving the evolution of supercomputing systems over the last decade. Current generation supercomputing systems are typically comprised of thousands of compute nodes based on modern multi-core architectures. Interconnection networks have also rapidly evolved to offer low latencies and high bandwidths to meet the communication requirements of parallel applications. Together, these systems are allowing scientists to scale their parallel applications across tens of thousands of processes.

InfiniBand has emerged as a popular high performance network interconnect and is being increasingly used to deploy some of the top supercomputing installations around the world. Most supercomputing systems are organized as racks of compute nodes and use complex network architectures comprised of many levels of leaf and spine switches, as shown in Figure 1. The Message Passing Interface(MPI) [1] has been the dominant programming model for high performance computing applications, owing to its ease of use and scalability. The current MPI Standard, MPI-2.2, defines a set of collective operations that may be used to communicate data among a group of participating processors. Owing to their ease of use and portability, these collective operations are very commonly used by application developers and scientific applications are also known to spend a considerable fraction of their run-times in MPI collective operations. The performance of collective operations is very critical to the overall application performance, particularly at large scales.

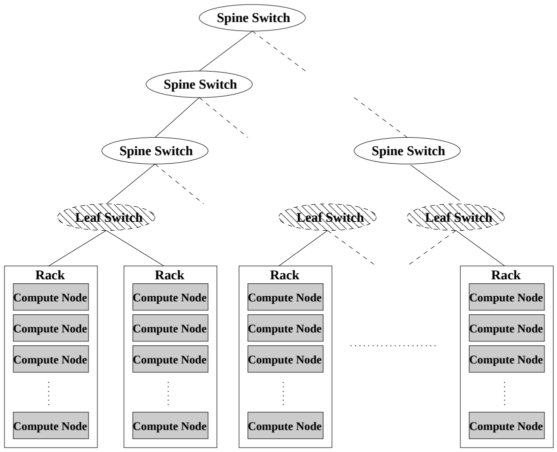

Figure 1. Typical InfiniBand Fat-tree Topology

Most MPI libraries such as MPICH2, Open-MPI, MVAPICH2 [2] use aggressive techniques to optimize the latency of collective operations [3], [4]. Researchers have proposed various multi-core aware shared-memory based techniques to minimize the communication latency of collective operations. However, these techniques are limited to leveraging the node-level hardware topology and do not consider the network topology. In [5], [6], researchers have demonstrated that the network topology plays a critical role in affecting the latency of collective performance and scientific applications. We believe that it is critical to design MPI libraries in a network topology-aware manner to improve the communication costs of collective operations at large scales to deliver high performance to parallel applications. However, the topology information of InfiniBand networks is not readily available

*This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749, #DE-FC02-06ER25755 and contract #DE-AC02-06CH11357; National Science Foundation grants #CCF-0621484, #CCF-0702675, #CCF-0833169, #CCF-0916302 and #OCI-0926691; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Obsidian, Advanced Clustering, Appro, QLogic, and Sun Microsystems.

to parallel applications. Hence, it is also necessary to detect the topology of large scale supercomputing systems and expose this information to MPI libraries efficiently.

Many supercomputing systems are also becoming increasingly heterogeneous, either in terms of processing power or networking capabilities [7], [8]. Most MPI libraries, including MVAPICH2, operate in a compatibility mode when the networks are detected to be heterogeneous. However, this may degrade the performance of collective operations and parallel applications. An alternative is to design MPI communication libraries that are aware of the network speeds of all the links that are being used for a given parallel job. This information may be leveraged to compute lower latency routes along the network to optimize the communication latency of collective operations. Figure 2 indicates the design space available for developing topology-aware algorithms.

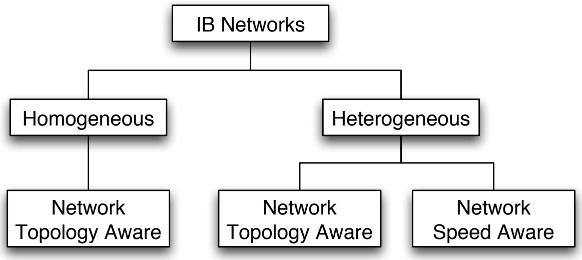

Figure 2. Design Space for Topology Aware Algorithms

These scenarios lead to the following open challenges:

1) Is it possible to design a Network Topology Detection Service to automatically detect the topology of large scale InfiniBand clusters?
2) Can we re-design the communication schedule of collective operations (say Broadcast) in a network topology-aware manner to achieve better performance?
3) Can the communication schedule for collective operations (say Broadcast) be re-designed in a network speed-aware manner to deliver best performance?
4) How much benefit can these enhancements provide for broadcast operations at the micro-benchmark and application level?

In this paper, we take on these challenges. We propose a light-weight Network Topology Detection service that can be queried by MPI libraries to learn about the topology of the network, for a given parallel job. We also propose topology-aware algorithms for the MPI_Bcast operation that leverage this information and re-order the communication graph to achieve lower communication latency. To the best of our knowledge, this is the first such work for InfiniBand clusters. Our experimental results show that, for large homogeneous systems and large message sizes, we get up to a 14% improvement in the latency of the broadcast operation using our proposed topology aware scheme over the default one at the micro-benchmark level. At the application level, we see up to an 8% improvement in total application performance as we scale up the job size. For a heterogeneous SDR-DDR cluster, we see that the proposed network speed-aware algorithms are able to deliver performance on par with homogeneous DDR clusters for small to medium sized messages. We also see that the network speed-aware algorithms perform 70% to 100% better than the naive algorithms when both are run on the heterogeneous SDR-DDR InfiniBand cluster.

II. Motivation

The broadcast algorithms in MVAPICH2 currently use the *k-nomial* exchange for small and medium sized messages and the *scatter-allgather* algorithm for larger messages. Figure 3 shows the communication pattern for the *k-nomial* algorithm with 32 processes, with rank 0 as the root. The tree-based algorithms are useful for minimizing the latency for small and medium messages, because they can complete the entire operation in $(O(logN))$ steps, where N is the number of participating processes. However, the tree based algorithms also have an imbalanced communication pattern. As demonstrated in Figure 3, Rank 16 receives the data from the root and becomes the co-root of a sub-tree that includes all processes - ranks 17 through 31. Similarly, rank 4 is another co-root that receives data from the root, but is responsible for broadcasting the data for a smaller subset of processes - ranks 5 through 7. It is to be noted that if either ranks 16 or 4 get delayed in receiving the data from the root due to poor process placement or slower network link, this delay gets propagated down the tree and affects the latency of the entire broadcast operation. This effect becomes more significant at larger scales as the trees become more imbalanced. The scatter phase of the scatter-allgather algorithm also uses a binomial-tree algorithm. The imbalanced tree affects the latency of large message broadcasts in a similar manner.

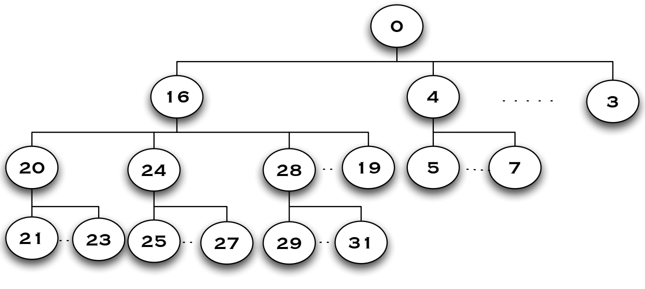

Figure 3. K-nomial (K=4) Tree Topology with 32 Processes

In our previous work [5], we proposed designing topology-aware collective algorithms by creating internal communicators that mirror the network topology and scheduling the communication recursively across these communicators. While such an approach was shown to be effective to improve the latency of collective operations, as we can see in Table I, it is quite possible that the nodes are allocated across several racks in the system. In such cases, it might not be beneficial to create rack-level communicators, because the number of racks are of the same order as the number of nodes for a given job. This observation leads us to the following challenge: *Is it possible to leverage the network topology information to re-organize the communication pattern of collective operations to improve their latency?*

Table I
AVERAGE JOB DISTRIBUTION ON RANGER IN APRIL 2011

Job Size	Average Hosts	Average Racks
<256	2	2
256 - 512	28	22
512 - 1K	58	40
1K - 2K	106	58
2K - 4K	224	74

III. BACKGROUND

In this section, we give an overview of the network architectures of large scale supercomputing systems, InfiniBand, the effect of topology on communication costs and collective message exchange algorithms used in MVAPICH2.

A. Network Architectures

Large scale supercomputers such as the TACC Ranger and Glenn at OSC have tens of thousands of computing cores. These cores are organized hierarchically across different racks, with each rack consisting of a few chassis, each chassis includes tens of compute blades. Each blade is based on a suitable multi-core architecture. Processes that belong to different blades, but within the same chassis are connected to one port of the leaf-switch and can communicate with a single hop within the leaf-switch. Processes that belong to the same rack, but different chassis will be connected to the same leaf-switch, but any communication will incur an additional hop within the leaf switch. Communication between processes that belong to different racks involve multiple hops across the leaf-level and the spine switches and will incur higher latency. In Figure 1, we give a high-level view of the topology of modern large-scale clusters. In Table II, we include MVAPICH point-to-point latency for communication between two processes that belong to different parts of the cluster. We can see that the cost of communication between processes that are across different racks is almost 81% higher than when compared to the cost of an intra-switch communication. In this work, we focus on clusters that are organized in a hierarchical tree-like manner. However, large scale clusters can be organized in various architectures such as the torus [9]. Such architectures offer high bi-section bandwidth for near-neighbor communication patterns, but network contention can significantly impact the performance of applications when the processor allocation is scattered across different racks in the cluster. While we focus on tree-based topologies in this paper, it is equally important to discover the topology on other types of networks and design collective message exchange algorithms that are aware of the topology in order to improve the overall performance of real-world applications.

B. InfiniBand

InfiniBand is a very popular switched interconnect standard being used by almost 41% of the Top500 Supercomputing systems [10]. InfiniBand Architecture (IBA) [11] defines a switched network fabric for interconnecting processing nodes and I/O nodes, using a queue-based model. It has multiple transport services including Reliable Connection (**RC**) and Unreliable Datagram (**UD**), and supports two types of communication semantics: Channel Semantics (Send-Receive communication) over RC and UD, and Memory Semantics (Remote Direct Memory Access - RDMA communication) over RC. Both semantics can perform zero-copy data transfers, i.e. the data can directly be transferred from the application source buffers to the destination buffers without additional host level memory copies.

C. Broadcast Algorithms used in MVAPICH2

In MVAPICH2 [2], we use optimized multi-core aware, shared-memory based algorithms to implement the *One-to-All* Broadcast operation. However, these optimizations are limited to identifying and grouping processes that are within the same compute node and we have no knowledge of the topology at the switch-level. For each communicator, we create an internal shared-memory communicator to contain all the processes that are within the same compute node. We assign one process per compute node as the node-level leader process and create a node-level-leader communicator to include such processes. We schedule the MPI_Bcast operation across these communicators to minimize the communication latency. The intra-node phase of the MPI_Bcast operation can either use the direct shared-memory approach or a k-nomial algorithm through basic point-to-point operations. For the inter-leader phase, we use the k-nomial algorithm for small and medium sized messages, and a scatter-allgather approach for larger messages.

D. WindJammer

The example application, Windjammer, issues two large broadcast calls in its main loop. Windjammer is a parallel Neighbor-Joining MPI C++ program used to build approximate evolutionary trees, known as phylogenetic trees, from either DNA or amino acids sequences, known as taxa. The Neighbor-Joining algorithm was first described in 1987 [12]. Windjammer is based on the scalar NINJA program developed by Wheeler [13]. In particular, it relies heavily on the Wheeler optimization described as "d-filtering". The application is used to evaluate our proposed broadcast algorithms.

IV. DESIGN

In this section, we describe the details of our network topology-/speed- aware design and implementation of the broadcast algorithm. The overall design of our network topology-aware framework is as shown in Figure 4.

Table II
MVAPICH Communication Performance Across Varying Levels of Switch Topology on TACC Ranger

Process Location		Number of Hops	MPI Latency (*us*)
Intra-Rack	Intra-Chassis	0 Hops in Leaf Switch	1.57
	Inter-Chassis	1 Hop in Leaf Switch	2.04
Inter-Rack		3 Hops Across Spine Switch	2.45
		5 Hops Across Spine Switch	2.85

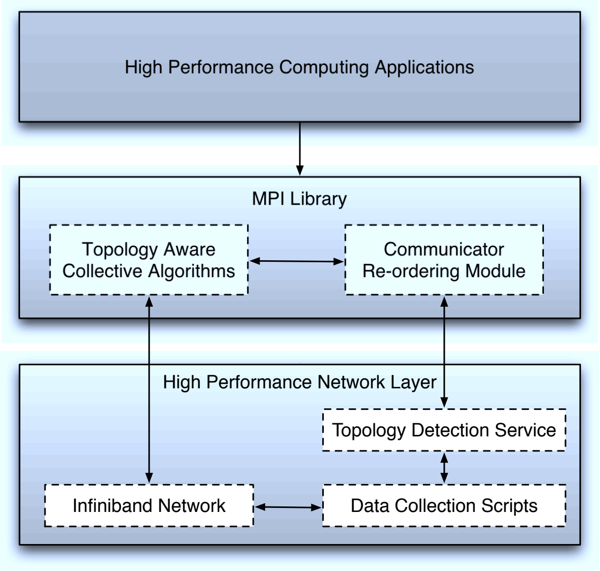
High Performance Computing Applications
MPI Library
Topology Aware Collective Algorithms
Communicator Re-ordering Module
High Performance Network Layer
Topology Detection Service
Infiniband Network
Data Collection Scripts

Figure 4. Overall Framework

A. Designing Topology Detection Service

InfiniBand network topology data is not available in a mode easily used by other programs. The physical connections between entities in the fabric are discoverable with the `ibnetdiscover` utility from the OFED distribution[14]. This program dumps the physical topology data, which includes port-to-port switch connectivity by GUID and port numbers, in a human-readable, free-form ASCII text file. Logical routing data is available by querying each switch in turn and dumping its Linear Forwarding Table (LFT) with the `ibroute` utility from OFED. The LFT on each switch specifies which outbound port on that switch to forward received packets for each unique entity in the fabric.

The required routing data is generally not available to users of IB systems who do not have administrative privileges. To get IB routing data available to general users of a system, we have taken a two-step approach. First, we have developed a set of scripts to be run by the administrator that use `ibnetdiscover` and `ibroute` to dump the physical routing data and LFTs from each switch. Then, we have developed a routing query server that ingests this data at startup and continuously as it is generated and provides a query service for the routes that it describes. Clients may connect to the server over TCP/IP sockets and make queries of the form:

```
query hosta:hostb
```

The service returns the GUIDs of the switches traversed between the hosts “hosta” and “hostb”. E.g.,

```
query i101-101:i102-201
i101-101 0x00144fa50f240050 0x00144f0000a60369
0x00144f0000a6037a 0x00144f0000a6036a
0x00144fa43db80050 i102-201
```

The server is multi-threaded and starts a new thread for each client connection. It is designed to handle a large number of simultaneous connections in order to support large-scale machines with many simultaneous jobs requesting data.

The LFT data in each switch is created and programmed by the Subnet Manager[15] during its startup. But any event like the insertion or removal of a host or switch, the activation or deactivation of a switch port, or other change in the fabric could affect the LFT. Given the potentially dynamic nature of the LFT data, the data collection scripts watch the Subnet Manager logs for changes in the fabric and re-dump the data whenever it changes. The topology query service watches for new raw data and reloads it whenever new data is available.

Future plans for the service include the addition of ingress and egress port data for each hop in the route between nodes, direct query of topology details using the OpenSM plugin interface, and supporting commands for a network-topology-based job scheduling service.

B. Designing Network-Topology/-Speed Aware Broadcast Algorithms

A topology discovery service similar to the one described in Section IV-A alone does not provide any benefit to end applications unless the implementations of various programming models these applications use, such as the MPI library, utilize this topology information to make intelligent routing choices while performing communication that spans the network. As we saw in Section III-C, MVAPICH2 uses some form of tree based pattern as the underlying communication scheme for almost all message sizes while performing the broadcast operation. As we saw in Section II, delaying a message at one of the tree nodes with many children tends to have a greater impact on the performance of the broadcast operation than delays created at the other nodes of the tree. In this context, we propose certain heuristics which are aimed at reducing the delay in this critical section of the broadcast algorithm.

The central idea is to to *map the logical broadcast tree to the physical network topology in-order to reduce the possible delays in the critical sections of the broadcast operation* described earlier. Note that we are not trying to achieve a perfect mapping of the logical tree to the physical topology. We are only trying to map certain critical regions of the broadcast which have been shown to have most impact on the performance of the broadcast operation as a whole. To this end, we re-order the ranks in the *node-level-leader* communicator described in Section III-C with information about the network in such a way as to allocate the fastest / shortest network links to the nodes in the critical region thus reducing possible network delays. We choose the classic *Depth First Traversal* (DFT) and *Breadth First Traversal* (BFT) methods as our heuristics to discover the critical sections of the broadcast operation as they lend themselves naturally to the tree based communication pattern used by the broadcast operation.

For each node-level leader in the broadcast, we create a new network aware communicator for it on an *on-demand* basis. When a node-level leader initiates a broadcast exchange operation for the first time, we create a new communication graph with the node level leader as the root and use either the depth first or the breadth first graph traversal method (as requested by the user) to traverse it. Figure 5 depicts the flow on control inside the MPI library with the addition of the topology aware collectives. Due to the nature of the traversal algorithms and the underlying communication graph, we will reach the nodes having most children before the other ones. For maintaining correctness of the application, the node on which the root of the broadcast operation resides is fixed. For each child process, we pick a node that is closest to its parent process. We use an N x N delay matrix (N = number of node level leaders) to define the 'closeness' of two processes. As we are focusing on delays caused due to reduced physical capacity of the network and the number of hops in the critical communication path, there are two broad methods in which we can create the delay matrix. In the coming sections, we discuss the design of these solutions in detail.

1) Delay Matrix for Network Topology Aware Broadcast: It is common for the various processes belonging to an MPI job to get distributed across various switch blades on large networks, like the one used by the *Ranger* supercomputing system at TACC. As the current state of the art collective communication algorithms do not take the network topology into account, it is possible that the messages intended for nodes in critical sections of the broadcast operation are required to traverse multiple hops before they reach the intended destination resulting in large delays and variable performance. In this context, we query the topology detection service to create the delay matrix based on the number of hops between any pair of node-level leaders. In order to avoid overwhelming the topology detection service, only Rank-0 of the node-level leader communicator queries it. A delay matrix is created locally and broadcasted to all other node-level leaders on the job. This is done only once at job initialization time and cached at each process for future use.

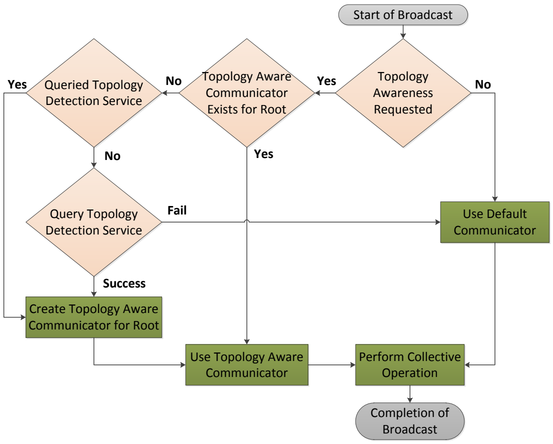
Start of Broadcast
Topology Awareness Requested
No
Yes
Topology Aware Communicator Exists for Root
No
Yes
Queried Topology Detection Service
Yes
No
Query Topology Detection Service
Fail
Success
Use Default Communicator
Create Topology Aware Communicator for Root
Use Topology Aware Communicator
Perform Collective Operation
Completion of Broadcast

Figure 5. Control Flow with Topology Aware Collectives

2) Delay Matrix for Network Speed Aware Broadcast: Heterogeneity adds a different dimension to the problem, requiring us to define the delay matrix in such a way as to take this into consideration. To this end, the node-level leaders perform an *Allgather* operation to obtain the link rate of the IB HCA's on all nodes taking part in the job. Once each node-level leader has the rate of all other nodes on the job, they create the delay matrix by taking the inverse of the of the speed of the HCA with the lower speed among a pair of nodes. This is done only once at job initialization time and cached at each process for future use.

V. Experimental Results

In this section, we describe the experimental setup, provide the results of our experiments, and give an in-depth analysis of these results.

A. Experimental Setup

Our experiments were conducted on the *Ranger* supercomputer at the Texas Advanced Computing Center and on the *Glenn* supercomputer at the Ohio Supercomputing Center. Ranger is a 3,936 node cluster with a total core count of 62,976. Each node is a SunBlade x6420 running a 2.6.18.8 Linux kernel with four AMD Opteron Quad-Core 64-bit processors (16 cores in all) on a single board, as an SMP unit. Each node has 32 GB of memory. The nodes are connected with InfiniBand SDR adapters. Ranger also runs the latest version of the topology discovery service that we described in Section IV-A. Thus, Ranger provides us a homogeneous environment to evaluate our network topology-aware broadcast algorithms.

Glenn, on the other hand, consists of multiple generations of InfiniBand adapters - SDR (8 Gbps) and DDR (16 Gbps). It is a 9532 core (1624 node) IBM Opteron cluster. The original system consists primarily of 877 dual socket, dual core 2.6 GHz compute nodes with 8 GB of RAM with an SDR two level InfiniBand fat-tree interconnect. The system was expanded to approximately twice its original size with the addition of 650 dual socket, quad core 2.5 Ghz compute nodes with 24 GB of memory connected via a DDR two level InfiniBand fat-tree. The SDR and DDR fat trees are connected via several links between the spine switches of the two trees. Thus, Glenn provides us a heterogeneous environment enabling us to evaluate our network speed-aware broadcast algorithms. Due to technical reasons, we were not able to get the topology discovery service up and running on the Glenn cluster. We use the OSU Benchmark suite and the WindJammer [16] application to evaluate the effectiveness of our algorithms.

Earlier work [17] has shown that noise (both in the network and on the host) is a factor that affects the performance of all parallel jobs that run on modern commodity supercomputing systems. In this context, we ran all the experiments that compare and contrast the performance of the default version of the broadcast algorithm with our modified topology-aware version in an interleaved fashion instead of in a block manner to mitigate the impact of network noise on the results. We also repeated the same series of experiments on different days over a period of a month to ensure the repeatability and consistency of the results.

B. Impact of Network-Topology/-Speed Aware Algorithms on Performance of Collective Benchmarks

In this section we look at the impact network topology-/speed- aware broadcast algorithms have on the performance of the broadcast operations on homogeneous and heterogeneous clusters.

1) Homogeneous Clusters: We evaluate the performance of the network topology-aware broadcast algorithms described Section IV-B1 here. We ran the benchmark for different core counts and message sizes to evaluate how the proposed scheme scales as the size of the job as well as the message increases. We observed that the impact of network topology on the performance of smaller messages were not that significant. This follows from the fact that most of the time spent in the network for small messages get spent in the startup phase. But as the size of the message increases, we slowly begin to see the impact of the topology-aware schemes on the performance of the broadcast operation.

Figure 6 (a) shows this trend for various message sizes when the benchmark was run on 1024 cores. The same trend exists for jobs of size 256 and 512 as well. We do not include these graphs due to their repetitive nature. Figure 6 (b), on the other hand, shows how the performance of the proposed topology-aware scheme scales as the size of the job increases for a fixed message size of 1 MB. As we can see, the topology-aware schemes do not have a significant impact at smaller job sizes. This is due to the fact that the broadcast trees similar to the ones seen in Section III-C are not large enough for the perturbations caused by delay in delivery of packets to the intermediate nodes to affect the performance of the broadcast operation as a whole. But as the size of the system scales, the impact that the topology-aware version of the algorithms have on the performance of the broadcast operation also increases. This is very significant as the size of supercomputing systems are expected to scale to exascale levels very soon. The same trend exists for messages upwards of 4 KB. We do not include these graphs here due to their repetitive nature. Our experimental results show that, for large system sizes and message sizes, we get up to a 14% improvement in the latency of the broadcast operation using our proposed topology-aware scheme over the default one.

2) Heterogeneous Clusters: In this section, we evaluate the performance of the network speed-aware broadcast algorithms described Section IV-B2. We ran the benchmark for the broadcast operation on different combinations of homogeneous and heterogeneous configurations to evaluate the performance of the proposed scheme. Figures 7 (a) and (b) shows the performance comparison of the proposed network speed-aware algorithm and the default network-speed un-aware algorithm on a heterogeneous 128-core allocation on the Glenn cluster consisting of hosts with DDR or SDR HCAs. We also show the performance of the default algorithm on a homogeneous 128-core allocation on Glenn just as a point of reference. We see that the network speed-aware broadcast algorithms are able to perform two times better than the network-speed un-aware version on a heterogeneous allocation. At the same time, we also see that its performance is on par with the performance of default algorithm on the homogeneous allocation for small to medium sized messages. But as the size of the message increases, the operation becomes more bandwidth bound and hence more dependent on the raw speed offered by the network. Consequently, the performance of the network speed-aware algorithm on the heterogeneous cluster begins to perform worse than that of the default algorithm on the homogeneous allocation. The slight degradation in performance seen for messages less than 128 bytes is due to difference in size of the messages that can be sent in an inline fashion by the IB HCA. While the newer DDR HCA is able to handle an inline size of 128 bytes, the older SDR HCA is only able to handle a maximum inline size of 64 bytes.

Figures 8 (a) and (b) shows the same results for a 256 process run of the broadcast operation.

C. Application Level Results

In this section, we look at the performance benefits obtained for the WindJammer application described in Section III-D with the network topology-aware version of the broadcast algorithm on *Ranger* with 128 and 256 processes.

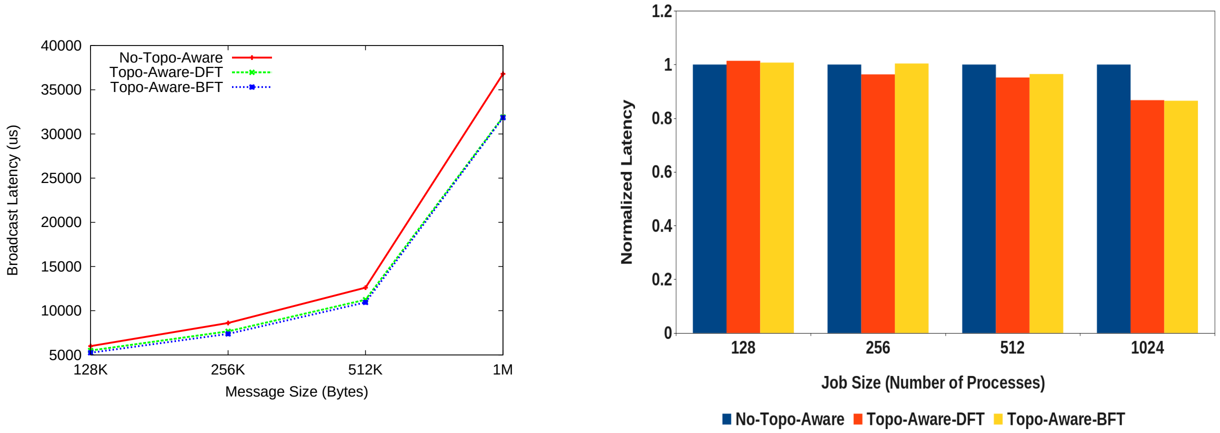

Figure 6. Impact of Network-Topology Aware Algorithms on Broadcast Performance for various: (a) Message Sizes at 1K Job Size and (b) Job Sizes

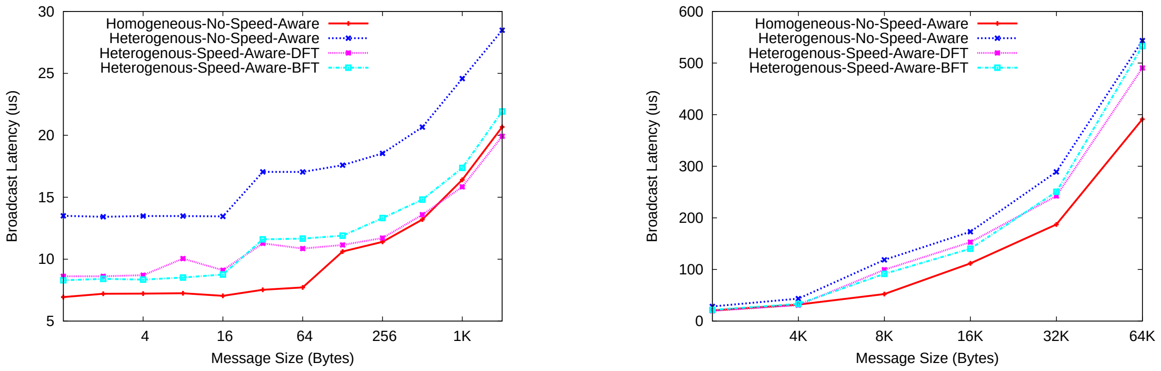

Figure 7. Impact of Network-Speed Aware Algorithms on a 128 process Broadcast Performance for: (a) Small Messages and (b) Medium Messages

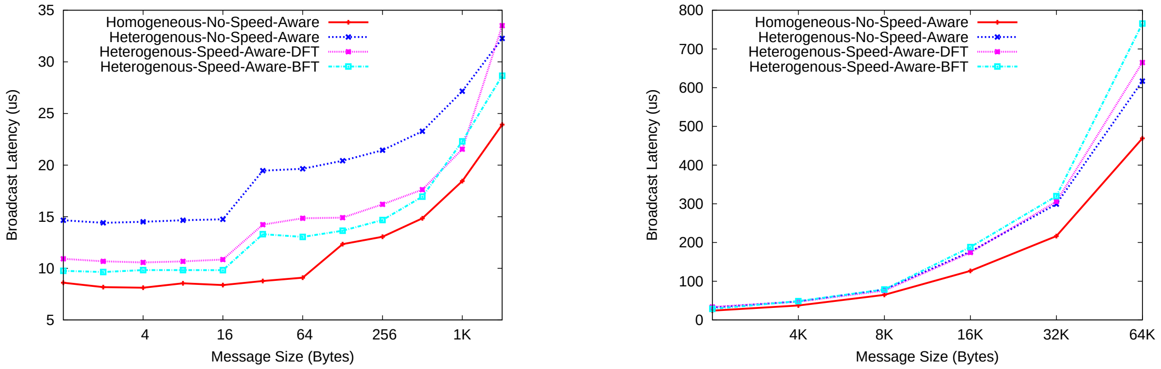

Figure 8. Impact of Network-Speed Aware Algorithms on a 256 process Broadcast Performance for: (a) Small Messages and (b) Medium Messages

Figure 9 depicts the performance of the default network-topology un-aware algorithm as well as the proposed network topology-aware algorithm for various system sizes. As we saw with the micro-benchmark results on homogeneous clusters in Section V-B1, the impact of the network topology-aware version of the algorithm is not felt that much at smaller system sizes. But as the system size scales, the improvement seen in performance also increases. We see that, at 256 processes, an improvement of up to 8 % is seen at the application level by using the topology-aware version of the broadcast algorithm over the default one.

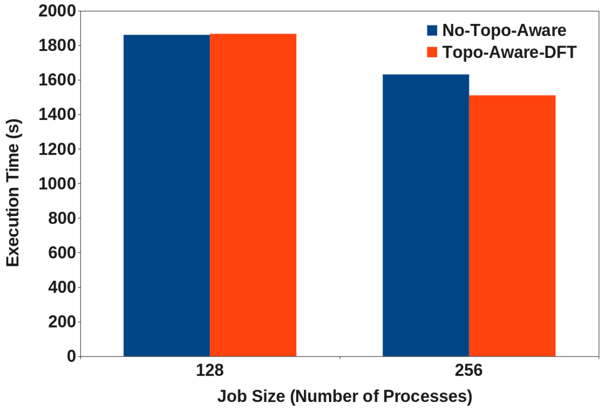

Figure 9. Impact of Network-Topology Aware Algorithms on WindJammer Application

We will include the experimental results at the application level on the Glenn cluster in the final version of the paper. Due to scheduling conflicts, our experiments did not complete before the submission deadline.

D. *Overhead of Creating Topology Aware Communicator*

In this section we analyze the various overheads involved in creating the topology aware communicator. Table III shows the time spent by the MPI library in various phases of topology discovery for various number of leader processes. The total job size would be 128, 256 and 512 respectively on a 8 core system and 256, 512 and 1024 respectively on a 16 core system. As we can see, most of the cost involved in topology discovery is a one time cost making the approach scalable.

VI. Related Work

Extensive research has been conducted around the mapping problems in parallel and distributed computing. The general idea being to map a task graph to a network graph while minimizing the overhead of communication and balancing computational load. This problem can be reduced to a graph-embedding problem which has been proved to be $\mathcal{NP} - Complete$ [18], [19].

From the 70's through the 90's, researchers proposed different solutions to solve the mapping problem based on heuristic algorithms (*local optimization* [20], [21], *greedy* [22], [23], *branch-and-bound* [24] etc.) and physical optimization algorithms (*graph contraction* [25], [26],*simulated annealing* [27], [28], *neural networks* [29], *genetic algorithm* [30] etc.). But most of these works targeted interconnect topologies such as hypercubes, array processors or shuffle-exchange networks, while modern system topologies are mainly meshes, tori and fat-trees. With Petascale clusters, the communications are becoming the bottleneck. It's becoming critical to improve the mapping of communication graphs on these interconnect topologies to reduce the impact of contention on large scale parallel applications.

Recent works of Bhatele [31], [6], [32] et. al. demonstrated the importance of topology-aware mapping for communication bound applications on tori networks. They have presented a framework for automatic mapping of parallel applications using a suite of heuristics that they developed. Their framework was based on two steps: 1. Obtaining the communication graph using profiling libraries, 2. Based on the communication patterns and topology information, apply heuristics to obtain mapping solution. The topology information was obtained through system calls. Their work mainly focused on torus networks. In this paper, we have presented our approach to automatically obtain topology information and adapt MPI collective operations. Our work directly focuses on commodity networks and takes into account heterogeneous network speeds.

VII. Conclusions and Future Work

In this paper, we designed a Network Topology Detection Service to automatically detect the topology of large scale InfiniBand clusters. We re-designed the communication schedule of the Broadcast operation in a network topology and speed-aware manner. Our experimental results show that, for large homogeneous systems and large message sizes, we get up to a 14% improvement in the latency of the broadcast operation using our proposed topology-aware scheme over the default one at the micro-benchmark level. At the application level, we see up to an 8% improvement in total application performance as we scale the job size up. For a heterogeneous SDR-DDR cluster, we see that the proposed network speed-aware algorithms are able to deliver performance on a par with homogeneous DDR clusters for small to medium sized messages. We also see that the network speed-aware algorithms perform 70% to 100% better than the naive algorithms when both are run on the heterogeneous SDR-DDR InfiniBand cluster.

VIII. Acknowledgments

We would like to express our sincere appreciation to Doug Johnson of Ohio Supercomputer Center for his timely help in enabling us to evaluate our algorithms. We would also like to thank our colleague Jithin Jose for his help in getting the paper ready.

Table III
OVERHEAD OF TOPOLOGY DISCOVERY IN MPI LIBRARY (TIME IN MS)

Number of Leaders	8			16			32		
Phase	Initial Call	Subsequent (Same Root)	Subsequent (Different Root)	Initial Call	Subsequent (Same Root)	Subsequent (Different Root)	Initial Call	Subsequent (Same Root)	Subsequent (Different Root)
Discover Connected Switches	5.1	NA	NA	4.2	NA	NA	6.9	NA	NA
Compute Delay Matrix	5.7	NA	NA	23.7	NA	NA	103.1	NA	NA
Broadcast Delay Matrix	0.5	NA	NA	4.8	NA	NA	5.9	NA	NA
Create Topology Aware Communicator	0.13	NA	0.13	1.5	NA	1.5	2.4	NA	2.4
Total Time	11.6	0	0.13	30.8	0	1.5	115.5	0	2.4

REFERENCES

[1] MPI Forum, "MPI: A Message Passing Interface," in *Proceedings of Supercomputing*, 1993.

[2] MVAPICH2, http://mvapich.cse.ohio-state.edu/.

[3] K. Kandalla and H. Subramoni and G. Santhanaraman and M. Koop and D.K. Panda, "Designing Multi-leader-based Allgather Algorithms for Multi-core Clusters," in *IEEE International Symposium on Parallel and Distributed Processing, IPDPS*, 2009.

[4] R. Graham and G. Shipman, "MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science*, vol. Volume 5205/2008, 2008, pp. 130–140.

[5] K. Kandalla and H. Subramoni and D.K. Panda, "Designing Topology-Aware Collective Communication Algorithms for Large Scale InfiniBand Clusters : Case Studies wih Scatter and Gather," in *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing*, 2010.

[6] A. Bhatele, "Automating Topology Aware Mapping for Supercomputers," Ph.D. dissertation, Dept. of Computer Science, University of Illinois, August 2010.

[7] http://www.nas.nasa.gov/Resources/Systems/pleiades.html, nASA Pleiades Supercomputer.

[8] "Glenn, the ohio supercomputer center (osc)," http://www.osc.edu/supercomputing/hardware/.

[9] Redsky: Top500 Supercomputing Sites, http://www.top500.org/system/10188.

[10] Top500, "Top500 Supercomputing systems," Oct 2010, .

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

[12] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogentic trees," *Mol. Biol. Evol*, vol. 4, pp. 406–425, 1987.

[13] T. J. Wheeler, "Large-scale neighbor-joining with ninja," in *Proceedings of the 9th Workshop on Algorithms in Bioinformatics*, S. Salzberg and T. Warnow, Eds., WABI. Springer, Berlin, 2009, pp. 375–389.

[14] "Open fabrics enterprise distribution," http://www.openfabrics.org/.

[15] OFED, "Open Subnet Manager," http://www.openfabrics.org/downloads/management/README.

[16] www.tacc.utexas.edu/tacc-projects/mclay/windjammer, windjammer project page.

[17] A. Bhatele, L. Wesolowski, E. J. Bohm, E. Solomonik, and L. V. Kalé, "Understanding application performance via micro-benchmarks on three large supercomputers: Intrepid, ranger and jaguar," *IJHPCA*, vol. 24, no. 4, pp. 411–427, 2010.

[18] S. H. Bokhari, "On the mapping problem," *IEEE Transactions on Computers*, vol. 30, pp. 207–214, 1981.

[19] F. Erçal, J. Ramanujam, and P. Sadayappan, "Task allocation onto a hypercube by recursive mincut bipartitioning," *J. Parallel Distrib. Comput.*, vol. 10, no. 1, pp. 35–44, 1990.

[20] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–308, 1970.

[21] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[22] T. Baba, Y. Iwamoto, and T. Yoshinaga, "A network-topology independent task allocation strategy for parallel computers," in *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, ser. Supercomputing '90, 1990, pp. 878–887.

[23] E. Huedo, M. Prieto, I. M. Llorente, and F. Tirado, "Impact of pe mapping on cray t3e message-passing performance," in *European Conference on Parallel Processing*, 2000, pp. 199–207.

[24] S. Radhakrishnan, R. Brunner, and L. V. Kalé, "Branch and bound based load balancing for parallel applications," in *Proceedings of the Third International Symposium on Computing in Object-Oriented Parallel Environments*, ser. ISCOPE '99. London, UK: Springer-Verlag, 1999, pp. 194–199. [Online]. Available: http://portal.acm.org/citation.cfm?id=646895.709722

[25] F. Berman and L. Snyder, "On mapping parallel algorithms into parallel architectures," *Journal of Parallel and Distributed Computing*, vol. 4, pp. 439–458, 1987.

[26] N. Mansour, R. Ponnusamy, A. Choudhary, and G. C. Fox, “Graph contraction for physical optimization methods: a quality-cost tradeo for mapping data on parallel computers,” in *7th International Conference on Supercomputing*, 1993, pp. 1–10.

[27] S. W. Bollinger and S. F. Midkiff, “Processor and link assignment in multicomputers using simulated annealing,” in *ICPP (1)*, 1988, pp. 1–7.

[28] S. Wayne Bollinger and Scott F. Midkiff, “Heuristic technique for processor and link assignment in multicomputers,” *IEEE Trans. Comput.*, vol. 40, pp. 325–333, March 1991.

[29] N. Mansour and G. Fox, “Allocating data to multicomputer nodes by physical optimization algorithms for loosely synchronous computations,” *Concurrency - Practice and Experience*, vol. 4, no. 7, pp. 557–574, 1992.

[30] T. Chockalingam and S. Arunkumar, “Genetic algorithm based heuristics for the mapping problem,” *Computers & Operations Research*, vol. 22, pp. 55–64, 1995.

[31] A. Bhatele and L. V. Kalé, “An evaluative study on the effect of contention on message latencies in large supercomputers,” in *IPDPS*, 2009, pp. 1–8.

[32] A. Bhatele, E. J. Bohm, and L. V. Kalé, “Optimizing communication for Charm++ applications by reducing network contention,” *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 211–222, 2011.