

Designing Topology-Aware Collective Communication Algorithms for Large Scale InfiniBand Clusters: Case Studies with Scatter and Gather*

Krishna Kandalla¹, Hari Subramoni¹, Abhinav Vishnu² and Dhabaleswar K. (DK) Panda¹

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

² High Performance Computing Group,
Pacific Northwest National Laboratory
{abhinav.vishnu}@pnl.gov

Abstract

Modern high performance computing systems are being increasingly deployed in a hierarchical fashion with multi-core computing platforms forming the base of the hierarchy. These systems are usually comprised of multiple racks, with each rack consisting of a finite number of chassis, and each chassis having multiple compute nodes or blades, based on multi-core architectures. The networks are also hierarchical with multiple levels of switches. Message exchange operations between processes that belong to different racks involve multiple hops across different switches and this directly affects the performance of collective operations. In this paper, we take on the challenges involved in detecting the topology of large scale InfiniBand clusters and leveraging this knowledge to design efficient topology-aware algorithms for collective operations. We also propose a communication model to analyze the communication costs involved in collective operations on large scale supercomputing systems. We have analyzed the performance characteristics of two collectives, MPI_Gather and MPI_Scatter, on such systems and we have proposed topology-aware algorithms for these operations. Our experimental results have shown that the proposed algorithms can improve the performance of these collective operations by almost 54% at the micro-benchmark level.

1. Introduction

Large scale supercomputing systems have witnessed significant growth in the recent history and some of them are poised to break the Exaflop barrier in the coming years. At the heart of these systems are

compute nodes based on modern multi-core architectures and high speed networks. As shown in Figure 1, clusters such as the TACC Ranger [1] and the Roadrunner [2] have thousands of compute cores and are typically comprised of hundreds of racks, with each rack consisting of a few chassis, and each chassis including tens of compute-blades. The compute blades typically have multiple compute cores and are coupled together through high-speed networks such as InfiniBand [3], 10GigE [4] or other proprietary networks, through multiple stages of switches. The core density in compute nodes is constantly on the rise with upcoming architectures such as the Intel Nehalem-EX [5] and the Magny-Cours [6] from AMD, which will include 8-12 cores per socket. These systems offer vast amounts of computing power and resources to application developers and are allowing scientific applications to scale out to several thousands of processes.

Most of these systems typically operate at near-peak load and handle thousands of requests that are submitted through a batch-submission system and it is necessary to schedule the resources in a fair manner across several users. Softwares such as the Sun Grid Engine (SGE) [7] and STORM [8] are used to address this requirement. Such schedulers typically aim to lower the amount of time various jobs spend in the submission queue and try to keep the overall system as busy as possible. As we show in Section 2, processor allocation patterns can affect the time required to perform a simple zero-byte point-to-point operation on a large scale cluster by almost 80%. Also, such systems allow several jobs to concurrently run on various nodes of the system and they all contend for the network. This limits the effective network bandwidth available per application. The effects of network contention become more severe when the processes are allocated across various racks in the system. This significantly affects the performance of collective message exchange operations and directly affects the application run-times. But, in real-world systems, guaranteeing contiguous processor allocation

*This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342, #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, Advanced Clustering, Appro, QLogic, and Sun Microsystems.

places a higher burden on the job scheduler and would also imply that jobs would be spending a considerably higher amount of time in the submission queues, which may not be acceptable. This leads to the following open challenge: *Can collective algorithms be designed to be aware of network topology and to be resilient to network traffic to deliver optimal performance to applications.*

In this paper, we take on the following important design challenges :

- How do we efficiently discover the topology of a large scale InfiniBand cluster?
- What are the challenges involved in designing efficient collective algorithms that are aware of the network topology?
- Can we derive communication cost models for collective operations on large-scale systems with several levels of hierarchies?
- What is the effect of the background traffic on the performance of collective operations? Can we leverage the topology information to design algorithms that are resilient to network contention?

In this paper, we propose topology-aware algorithms for two collective operations - MPI_Scatter and MPI_Gather. We also derive communication models to account for the costs associated with various levels of hierarchy that are found in large scale supercomputing systems. The rest of the paper is organized in the following manner. In Section 2, we discuss the topology of large-scale clusters, its impact on communication costs and collective algorithms in MVAPICH2. We describe the methodology involved in discovering the InfiniBand topology and the creation of the various sub-communicators to capture the topology information in Section 3. In Section 4, we propose topology-aware algorithms for Scatter and Gather operations. In Section 5, we give experimental results for our proposed algorithms. We list the relevant research contributions in Section 6.

2. 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.

2.1. Network Architectures

Large scale supercomputers such as the TACC Ranger 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 1, we include MVAPICH point-to-point latency for communication between 2 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-link 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 the network contention can significantly impact the performance of applications when the processor allocation is scattered across different racks in the cluster. It is critical to discover the topology of such clusters and design collective message exchange algorithms that are aware of the topology in order to improve the overall performance of real-world applications.

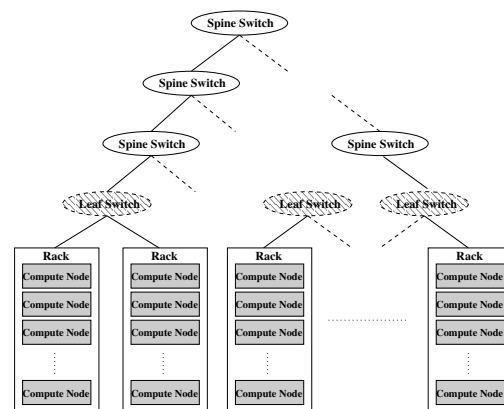


Figure 1. A Typical Topology

2.2. InfiniBand

InfiniBand [3] is a processor and I/O interconnect standard that has become popular in the high performance computing area and being used increasingly to connect commodity clusters. Almost 36% of the

Process Location		Number of Hops	MPI Latency (μs)
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

Table 1. MVAPICH Communication Performance Across Varying Levels of Switch Topology on TACC Ranger

clusters in the Top500 supercomputers are currently using InfiniBand as the interconnect.

2.3. Collective Operations in MPI-2

The MPI standard [10] specifies various types of collective operations such as *All-to-All*, *All-to-One*, *One-to-All* and *Other*. Collective operations belonging to the types *All-to-One* and *One-to-All* assign one process as the *root* and involve a message exchange pattern in which the root either acts as the source or the sink. *Personalized* collective operations such as `MPI_Scatter`, `MPI_Gather` and `MPI_Alltoall` involve each process sending and/or receiving a distinct message. Operations of this type can benefit with networks that offer higher bandwidths. In this paper, we have studied the performance of `MPI_Scatter` and `MPI_Gather` collective operations, in-depth.

2.4. Collective Message Passing Algorithms used in MVAPICH2

In MVAPICH2 [11], we use optimized shared-memory based algorithms to optimize several collective operations. 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. Currently, for each communicator, we create an internal shared-memory communicator to contain all the processes that are within the same compute node and share the same address space. We assign one process per compute node as the node-level leader process and create a node-level-leader communicator object to include all the node-level leader processes. For collectives such as `MPI_Bcast`, `MPI_Reduce`, `MPI_Allreduce` and `MPI_Barrier`, we schedule message exchange operations across these communicators to achieve lower latencies. These methods have two significant advantages:

- The shared-memory space is leveraged for exchanging messages between processes that are within the same node. This can lead to a higher degree of concurrency, than compared to exchanging messages through point-to-point calls, as shown in [12], [13], [14], [15]

- The intra-node stages of the communication operations can happen without any data movement across the network and can also minimize network contention

3. Determining and Using the Topology Information for InfiniBand Networks

In this section, we briefly describe the InfiniBand tools we used to gather information about our cluster. The InfiniBand fabric manager assigns a 16 bit local address (LID) to each device in the network. These addresses can be used to learn about the topology of the system. *ibnetdiscover* is one of the tools supported by the InfiniBand standard and it provides information about LID mapping across all the active ports in the switches and the network interfaces. It gives us detailed information of the connectivity between the hosts and the chassis to the leaf-switches and between switches. We have leveraged this information to create sub-communicators to group various processes according to the topology of the system. During the initialization phase of the MPI library, we create intra-switch communicators to include all the processes that are connected to the same leaf-switch. We also create an intra-chassis communicator to group all processes that belong to the same chassis. We assign one process in each intra-chassis communicator as the chassis-leader, and one process in the intra-switch communicator as the switch-leader. We then create switch-leader and chassis-leader communicators that include the respective leader processes. In Section 4, we have used these sub-communicators to design topology-aware algorithms. In this work, we create the sub-communicators based on the topology of the system at job-launch time.

4. Designing Topology-Aware Collective Algorithms

In Section 2, we illustrated the impact of topology on small message latency between a pair of processes. Collective operations involve many processes exchanging messages and are sensitive to network noise [16]. Since production environments allow several applications to run concurrently, the effective bandwidth

available per application also plays a key role in determining the time consumed for a collective operation. In this section, we propose topology-aware algorithms for optimizing the performance of collective operations on large scale computing systems.

In [17], the authors proposed models to predict the costs of various collective algorithms for small scale clusters with compute nodes comprising of a single core. In this paper, we extend these models to incorporate the communication costs associated with various levels of hierarchy in modern large scale clusters. Let t_s -intra-node be the start-up cost associated with an intra-node message exchange operation and t_w -intra-node be the cost involved in sending a word of data to a peer process within the same node. Similarly, t_s -intra-switch and t_w -intra-switch are the costs associated with a message exchange operation between processes that are connected to the same leaf switch and t_s -inter-switch and t_w -inter-switch are the costs involved for an inter-switch transfer operation. Owing to the hierarchy of the system, the following is true :

$$t_s\text{-intra-node} < t_s\text{-intra-switch} < t_s\text{-inter-switch} \\ t_w\text{-intra-node} < t_w\text{-intra-switch} < t_w\text{-inter-switch}$$

It is to be noted that the inter-switch costs depends on the number of switch-hops involved in the message exchange operation. Also, we assume that the contention involved in intra-node and intra-switch operations are insignificant when compared to that of inter-switch exchanges.

4.1. MPI_Gather and MPI_Scatter - Default algorithms

As explained in [18], MPI implementations such as MPICH2 [19], Open-MPI [20] and MVAPICH2 [11] use tree-based algorithms to implement these operations. These algorithms were proposed and optimized for conventional single-core systems. However, on large scale production systems, it is necessary to design efficient topology-aware algorithms to lower the communication costs. Since these algorithms are popularly used across various MPI implementations, we have chosen to study the behavior of these algorithms on large scale InfiniBand clusters.

Consider the case in which an MPI_Gather operation is being invoked with a message size of N bytes, with a group of P processes, that are distributed in some manner across R racks. Let $T_{binomial}$ be the cost of performing an MPI_Gather operation using the binomial algorithm on such a system. For simplicity, we assume that the root of the operation

is also one of the rack-level leaders. The inter-rack communication can involve more than one inter-switch exchange operation. Since we know that the latency involved in inter-switch message exchange operations is higher when compared to intra-switch and intra-node exchanges, the cost of the entire operation is dominated by the number of inter-switch operations, the cost associated with each inter-switch exchange and the network contention. Consider α to be the overhead introduced by the contention at various levels of switches in the system. The cost involved in binomial tree based approaches with P processes on traditional single-core systems could be expressed in terms of $\log(P)$. Let us introduce three variables C_1 , C_2 and C_3 to account for the hierarchy in the system. These parameters represent the number of intra-node, intra-switch and inter-switch exchanges such that, $C_1 + C_2 + C_3 = \log(P)$. Their values also depend on the distribution of processes across different racks. Depending on the processor allocation, each process can potentially be involved in an inter-switch exchange at some point during the MPI_Gather operation. So, we can assume that the value of C_3 can be $\geq R$. Also, in an MPI_Gather operation, different processes exchange different volumes of data depending on their position in the binomial tree. Since the cost of communication also depends on where the processes are physically located in the cluster, we introduce three positive variables γ , β and δ to indicate the fact that the t_w component of $T_{binomial}$ is obtained by adding up the individual costs of various intra-node, intra-switch and inter-switch operations, respectively. The values of these parameters are chosen such that, $C_1 * \gamma$, $C_2 * \beta$ and $C_3 * \delta$ belong to the interval $\{0, (P - 1)/P\}$. If all the processes are physically connected to just a single switch, there will be no inter-switch exchanges and the value of δ will be 0, and $C_1 * \gamma + C_2 * \beta = (P - 1)/P$. On extending the models presented in [18], we can express the cost of an MPI_Gather operation as

$$T_{binomial} = (t_s\text{-inter-node} * C_1 + t_s\text{-intra-switch} * C_2 \\ + \alpha * t_s\text{-inter-switch} * C_3) + t_w\text{-intra-node} \\ * (C_1) * (N * \gamma) + t_w\text{-intra-switch} \\ * (C_2) * (N * \beta) + \alpha * t_w\text{-inter-switch} \\ * (C_3) * (N * \delta)$$

Since the costs involved in inter-switch operations dominate the cost of the entire operation, the following is true:

$$T_{binomial} > (\alpha * t_s\text{-inter-switch} * C_3) \\ + (\alpha * N * t_w\text{-inter-switch} * C_3 * \delta)$$

4.2. Topology-Aware MPI_Gather and MPI_Scatter algorithms

From the above cost model, we can observe that if we schedule the inter-switch exchanges in an optimal manner, we can minimize the costs incurred due to the effects of network contention and also the costs associated with making multiple hops across various levels of the switches.

As described in Section 3, we create sub-communicators to reflect the topology of the system. This allows us to perform the entire collective operation efficiently in a recursive manner by minimizing the number of inter-switch operations and lowering both the t_s and t_w components of the costs of collective operations which result in better performance for both small and large messages. We can perform the entire collective operation in the following manner:

- The rack-leader processes independently perform an intra-switch gather operation. This phase of the algorithm does not involve any inter-switch exchanges.
- Once the rack-leaders have completed the first phase, the data is gathered at the root through an inter-switch gather operation performed over the R rack-leader processes.

However, it is to be noted that the messages exchanged between the rack-leader processes are the aggregated messages of size $M \geq N$, which were gathered at the rack-leader processes at the end of the first phase of the algorithm. Let T_{topo} be the cost of the proposed topology-aware algorithm. Since each rack-leader process is now a part of a binomial tree and has vectors of size M , T_{topo} can be expressed in a manner similar to the regular binomial exchange algorithm across R processes with a vector size of M bytes. Since we established a lower bound for the performance of $T_{binomial}$, in Section 4.1, we can express the lower bound of the cost of our proposed topology-aware algorithms in the following manner:

$$T_{topo} > (\alpha * t_s\text{-inter-switch} * \log(R)) + (\alpha * (1 - 1/R)) / (M * t_w\text{-inter-switch})$$

We indicated in Section 4.1, that the number of inter-switch exchange operations is a linear function of R . However, with our proposed topology-aware algorithm, the number of inter-switch operations is a function of $(\log R)$ and this should theoretically improve the latency component of the cost function by a factor of $(R/\log R)$ and this plays a key-role in minimizing the

overall time consumed for small message exchanges. For larger messages, the factor of improvement can be expressed as:

$$T_{binomial}/T_{topo} = N * \delta * C_3 / (M * (1 - 1/R))$$

$$T_{binomial}/T_{topo} = f(R)$$

In a general case, the root of the operation might not be physically located to be assigned as a rack-level leader. In such cases, our proposed algorithm will involve an additional communication step to move the aggregated data from the rack-leader process to the root process. Since this happens within a rack, this involve data movement within the leaf switch, but will not lead to any traffic over any other switches in the network. Since MPI_Gather and MPI_Scatter are symmetric operations, we have designed a similar topology-aware algorithm for the MPI_Scatter operation as well.

5. Experimental Evaluation

In this section, we briefly describe our experimental testbed. We also provide experimental results comparing the default algorithms and the proposed topology-aware algorithms on our cluster.

5.1. Experimental Testbed

We have used three InfiniBand DDR switches, Switches A, B and C to create a tree topology. Switch A is connected to 8 nodes based on the quad-core, quad-socket AMD Barcelona architecture and Switch B is connected to 32 nodes based on the quad-core, dual-socket Intel Clovertown architecture. Switches A and B are connected to Switch C with two InfiniBand DDR links each. All nodes have ConnectX DDR HCA's. For our experiments, we have used 4 nodes (64 processes) from Switch A and 29 nodes (232 processes) from Switch B. We have also used a simple benchmark code that iterates through various message sizes starting from 0 bytes up to 1M bytes and invokes a collective call several times in a loop. For each message size, the benchmark computes the average time each process takes to complete the collective operation by performing a global reduction operation.

As indicated earlier, it is important to study the impact of background traffic on the performance of collective algorithms. We have first measured the average time required to complete these collective operations for various message sizes when the system is quiet. Next, we have executed the same benchmark in the presence of a background MPI job involving a separate

set of K processes doing MPI_Alltoall operations in a continuous loop. These K processes are organized such that $K/2$ processes are connected to switch A and the remaining $K/2$ processes are connected to switch B. We have modified the Alltoall algorithm to perform only the inter-switch exchanges to maintain a constant traffic over the switches. Suppose the background process is performing the MPI_Alltoall operation with a vector size of M bytes, we will have a constant traffic of $K*M$ bytes over the switches. In our experiments, we have arbitrarily chosen to run the background job with 96 processes and we vary the buffer size in the Alltoall job to study the impact of network congestion on the performance of the collective operations. In order to ensure uniformity, we have run the background job on machines that are equipped with InfiniBand ConnectX DDR network interfaces.

5.2. MicroBenchmark Results

In Figure 2 (a), we compare the performance of the default binomial tree algorithm for MPI_Gather under quiet and busy conditions. We can see that this algorithm is quite sensitive to the presence of background traffic. When the background job is run with buffer sizes larger than 1MB, we see a performance degradation of almost 21% for large messages. When the background job is run with a buffer size of 4MB, the time taken to complete the MPI_Gather operation with 1MB message size is almost 10% higher than the case when the buffer size of the background job is 1MB. We can infer that the presence of background traffic can significantly affect the performance of the binomial exchange algorithm for MPI_Gather. In Figure 2 (b), we compare the performance of the default algorithm with the proposed topology-aware algorithm under both quiet and busy conditions. The background Alltoall job has been executed with a buffer size of 4MB to clearly delineate the performance difference between the default binomial exchange algorithm and the proposed topology-aware algorithm. We can infer that the proposed algorithm delivers an improvement of up to 50% for most messages, even when the network was busy. In Section 4.2, we had indicated that for large message MPI_Gather exchanges, we could potentially achieve a speed-up of a factor of R with our proposed topology-aware algorithms. In our experimental setup, we have distributed the processes across $R = 2$ racks and we would expect to see a performance benefit of a factor of 2. Hence, the experimental results that we have obtained match well with the theoretical analysis we had proposed in Section 4.2.

Similarly, in Figure 3(a), we compare the performance of the default binomial tree algorithm for

MPI_Scatter under quiet and busy conditions. We observe that the presence of background traffic has resulted in a smaller overhead in this operation, than MPI_Gather. In Figure 3(b), we compare the performance of the default algorithm with the proposed topology-aware algorithm under both quiet and busy conditions. We can infer that the proposed topology-aware algorithm performs almost 23% better than the default algorithm when the network is quiet.

6. Related Work

The concept of gathering topology information and leveraging this knowledge to design better algorithms has been applied in the past to grid based systems. Several researchers have explored the potential performance benefits of creating a multi-level communication framework to optimize the performance of parallel applications in grid environments [21], [22], [23], [24]. In [25], the authors have proposed a tool to automatically detect the topology of switched clusters for Ethernet based clusters. In [26], the authors have proposed an optimized algorithm for MPI_Allreduce operation that utilizes the network bandwidth in an efficient manner. In [27], the authors have proposed hierarchical algorithms for MPI_Scatter and MPI_Gather operations. However, these studies mostly focused on grid based systems that are loosely coupled through Ethernet or other proprietary networks. In our work, we have proposed topology-aware collective algorithms for large scale supercomputing systems that are tightly coupled through inter-connects such as InfiniBand. Resource management is again a well studied area in grids, [28]. Researchers have also explored optimizing collectives on a BlueGene system based on SMP nodes connected in a 3D torus architecture in [29]. However, in our study, we have explored the effects of topology across multiple chassis, racks and across multiple levels of switches. In [8], the authors have proposed an efficient resource manager for cluster based systems. But to the best of our knowledge, there is no resource manager that allocates compute nodes in a contiguous manner, at the same time keeping the overall system utilization high.

7. Conclusion

The scale at which supercomputers are being deployed continues to increase each year. These clusters offer a vast amount of computing resources to application developers. However, it is necessary to design efficient software stacks that can fully leverage the performance benefits offered by such systems to allow applications to achieve good scalability. In this paper, we have explored the need for detecting the

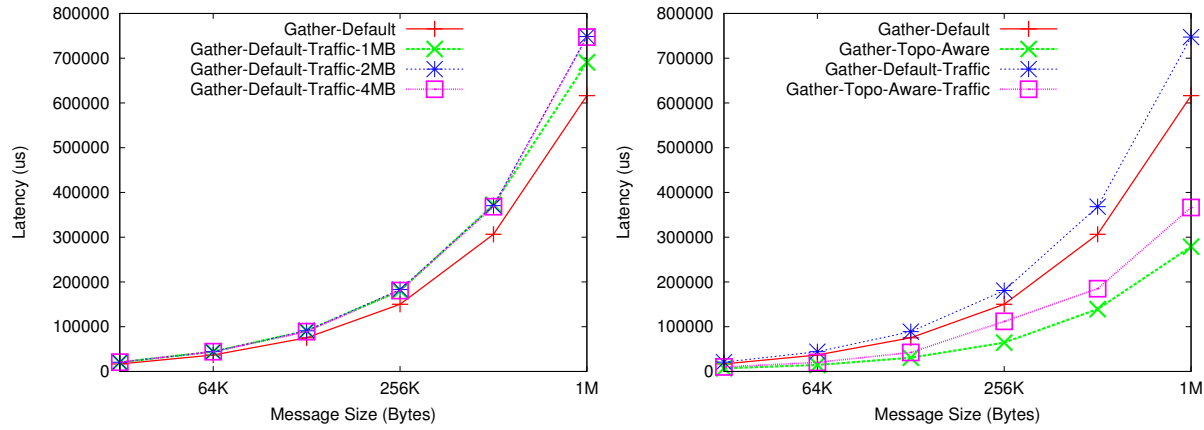


Figure 2. Gather latency for 296 processes: (a) Effect of background traffic on MPI_Gather and (b) Default Vs Topology-Aware MPI_Gather algorithms

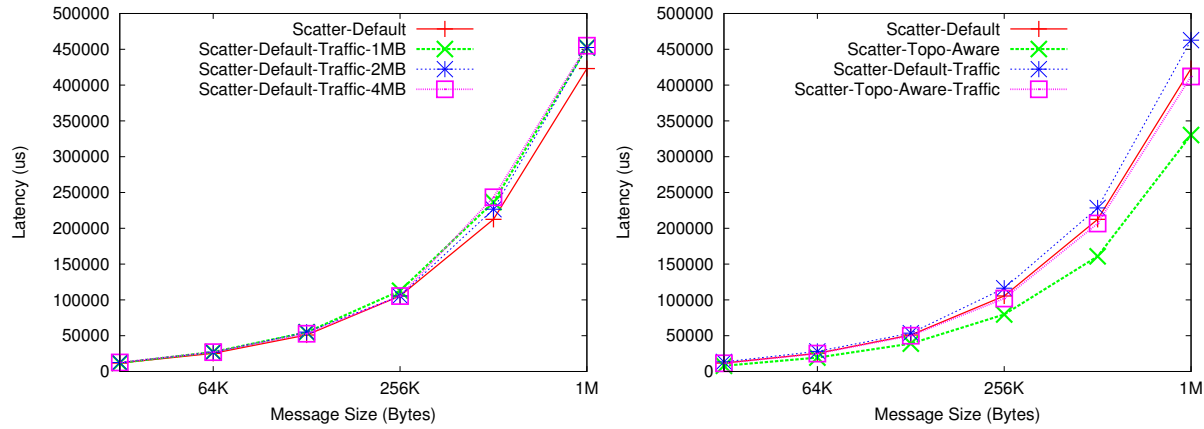


Figure 3. Scatter latency for 296 processes: (a) Effect of background traffic on MPI_Scatter and (b) Default Vs Topology-Aware MPI_Scatter algorithms

topology of large scale InfiniBand clusters and we have leveraged this knowledge to design efficient topology-aware collective algorithms for Scatter and Gather collective operations. We have proposed a communication model to theoretically analyze the performance of collective operations on large-scale clusters. We have compared the performance of our proposed topology-aware algorithms against the default binomial tree approaches and we have observed benefits of almost 54% with micro-benchmarks. In future, we plan on designing an interface to automatically detect the topology of InfiniBand clusters at job launch time and to intelligently deal with the possibility of the topology changing during the execution of applications. We also plan to design topology-aware algorithms for other collective operations such as the Alltoall Personalized Exchange, Allreduce and Broadcast operations and study their impact on real-world applications on large

scale clusters.

8. Acknowledgments

We would like to thank Dr. Bill Barth and Dr. Karl Schulz, TACC, for providing us with the MPI-level latency information on the TACC Ranger. We would also like to thank Jonathan Perkins for extending help with the hardware setup. We would also like to thank Adam Moody for sharing the benchmark codes.

References

- [1] Texas Advanced Supercomputing Center, <http://www.tacc.utexas.edu/resources/hpc/>.
- [2] Los Alamos National Laboratory, http://www.lanl.gov/discover/roadrunner_fastest_computer.
- [3] InfiniBand Trade Association, <http://www.infinibandta.org/>.

- [4] IEEE 802.3 Ethernet Working Group, "IEEE 802.3," <http://www.ieee802.org/3/>.
- [5] Intel Nehalem-EX, <http://download.intel.com/pressroom/pdf/nehalem-ex.pdf>.
- [6] AMD Magny Cours, <http://blogs.amd.com/work/tag/magny-cours/>.
- [7] Sun Microsystems, <http://www.sun.com/software/sge/>.
- [8] Frachtenberg, E. Petrini, F. Fernandez, J. Pakin, S., "STORM: Scalable Resource Management for Large-Scale Parallel Computers," in *IEEE Transactions on Computers*, Volume 55, Issue: 12, Dec, pp. 1572–1587.
- [9] Redsky: Top500 Supercomputing Sites, <http://www.top500.org/system/10188>.
- [10] MPI Forum, "MPI: A Message Passing Interface," in *Proceedings of Supercomputing*, 1993.
- [11] MVAPICH2, <http://mvapich.cse.ohio-state.edu/>.
- [12] A. R. Mamidala and R. Kumar and D. De and D. K. Panda, "MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics," in *8th IEEE International Symposium on Cluster Computing and the Grid, 2008, Lyon*, pp. 130–137.
- [13] A. R. Mamidala, A. Vishnu and D. K. Panda, "Efficient Shared Memory and RDMA Based Design for MPI-Allgather over InfiniBand," in *13th European PVM/MPI User's Group Meeting, Bonn, Germany, September 17-20, 2006, Vol. 4192*, 2006.
- [14] R. Kumar, A. R. Mamidala and D. K. Panda, "Scaling Alltoall Collective on Multi-core Systems," in *IEEE International Symposium on Parallel and Distributed Processing, 2008, IPDPS, 2008, Miami, FL*.
- [15] K. Kandalla, H. Subramoni, G. Santhanaraman, 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, Rome*.
- [16] T. Hoeftler, T. Schneider, and A. Lumsdaine, "The Impact of Network Noise at Large-Scale Communication Performance," in *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*, 2009, pp. 1–8.
- [17] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," in *Int'l Journal of High Performance Computing Applications*, 2005, pp. (19)1:49–66.
- [18] R. Thakur and W. Gropp, "Improving the Performance of Collective Operations in MPICH," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, 2003, Vol. 2840/2003*, 2003, pp. 257–267.
- [19] MPICH2, <http://www.mcs.anl.gov/research/projects/mpich2/>.
- [20] Open-MPI, <http://www.open-mpi.org/>.
- [21] B. Jakimovski and M. Gusev, "Improving Multilevel Approach for Optimizing Collective Communications in Computational Grids," in *Advances in Grid Computing - EGC, Lecture Notes in Computer Science, Volume 3470/2005*, 2005, pp. 548–556.
- [22] C. Coti, T. Herault, and F. Cappello, "MPI Applications on Grids: A Topology Aware Approach," in *Euro-Par 2009 Parallel Processing, Lecture Notes in Computer Science, Volume 5704/2009*, 2009, pp. 466–477.
- [23] Craig A. Lee, "Topology-Aware Communication in Wide-Area Message-Passing," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, Volume 2840/2003*, 2003, pp. 644–652.
- [24] N. T. Karonis, B. R. de Supinski, I. T. Foster, W. Gropp and E. L. Lusk, "A Multilevel Approach to Topology-Aware Collective Operations in Computational Grids," *CoRR*, vol. cs.DC/0206038, 2002.
- [25] J. Lawrence and X. Yuan, "An MPI Tool for Automatically Discovering the Switch Level Topologies of Ethernet Clusters," in *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008, Miami, FL, April, 2008*, pp. 1–8.
- [26] P. Patarasuk and X. Yuan, "Bandwidth Efficient All-reduce Operation on Tree Topologies," in *IEEE International Symposium on Parallel and Distributed Processing, 2007. IPDPS 2007, Long Beach, CA, 2007*, pp. 1–8.
- [27] Jesper Larsson Traff, "Hierarchical Gather/Scatter Algorithms with Graceful Degradation," in *IPDPS, vol. 1, pp.80, 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*.
- [28] P. Bar, C. Coti, D. Groen, T. Herault, V. Kravtsov, A. Schuster and M. Swain, "Running Parallel Applications with Topology-Aware Grid Middleware," in *5th IEEE International Conference on e-Science (eScience 2009), Oxford, UK, 2009*.
- [29] Sameer Kumar, Yogish Sabharwal, Rahul Garg, Philip Heidelberger, "Optimization of All-to-All Communication on the Blue Gene/L Supercomputer," in *Proceedings of the 2008 37th International Conference on Parallel Processing*, pp. 320–329.