Efficient Barrier and Allreduce on IBA clusters using hardware multicast and adaptive algorithms

AMITH R MAMIDALA, JIUXING LIU, DHABALESWAR K PANDA

Technical Report
OSU-CISRC-5/04-TR36

Efficient Barrier and Allreduce on IBA clusters using hardware multicast and adaptive algorithms[*]

Amith R Mamidala Jiuxing Liu Dhabaleswar K Panda

Computer and Information Science
The Ohio State University
Columbus, OH 43210
{mamidala, liuj, panda}@cis.ohio-state.edu

Abstract

Current algorithms for doing Barrier and Allreduce like pair-wise exchange, dissemination and gather-broadcast do not give an optimal performance when there is skew in the system. In pair-wise exchange and dissemination, all the nodes must arrive for the completion of each step. The gather-broadcast algorithm assumes a fixed tree topology. In this paper, we propose to use hardware multicast of InfiniBand in the design of an adaptive algorithm that performs well in the presence of skew. In this approach, the topology of the tree is not fixed but adapts depending on the skew. The last arriving node becomes the root of the tree if the skew is sufficiently large.

We carried out in-depth evaluation of our scheme and use synchronization delay as the performance metric for barrier and allreduce in the presence of skew. Our performance evaluation shows that our design scales very well with system size. Our designs can reduce the synchronization delay by a factor of 2.28 for Barrier and by a factor of 2.18 in the case of Allreduce. We have examined different skew scenarios and showed that the adaptive design performs either better or comparably to the existing schemes.

1. Introduction

Clusters built from commodity PCs are increasingly being used in the high performance computing arena. This is because they are very cost-effective and affordable. (MPI) [11] programming model has become the defacto standard to develop parallel applications that deliver high performance. MPI provides both *point-to-point* and *collective* communication functions. There are many applications which take advantage of these collective operations. Applications such as IS and FT in the NAS Parallel Benchmark suite [9] use these collectives almost exclusively for communication. Thus, providing high performance and scalable collective communication support is critical for many cluster systems.

Most of the current network interconnects provide features to support efficient collective communication. Recently, Infiniband has been emerging as a powerful interconnect technology. One of the notable features of Infiniband is that it supports hardware multicast. By using this feature, a message can be sent to several nodes in an efficient manner. Also, it has other important features like Remote Direct Memory Access(RDMA) operations. We can exploit these features to provide efficient and scalable collective operations over Infiniband clusters.

In this paper, we focus on two important collective operations, MPI_Barrier and MPI_Allreduce. MPI_Barrier is used as a synchronization call. Every process that has called a barrier blocks until all the participating processes have called this operation. In MPI_Allreduce, each process supplies a vector of certain length which is fixed across all the processes. All these vectors are reduced to a single vector using the operator provided in the collective call. Finally each process receives the resulting vector. Also note that there is an implicit synchronization of all the nodes participating in the Allreduce.

Many algorithms have been proposed in the literature for doing barrier and allreduce [4]. The most popular ones are the pair-wise exchange, dissemination and gather-broadcast. An implementation of these algorithms over Infiniband is discussed in [3]. As Infiniband clusters are becoming increasingly larger, one important factor to be considered in designing any collective is the skew of the

[*]This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, a grant from Sandia National Laboratory, a grant from Los Alamos National Laboratory, and National Science Foundation's grants #CCR-0204429 and #CCR-0311542.

processes involved in the operation. This is true especially with operations that involve synchronization like barrier and allreduce. The problem with the traditional algorithms discussed above is that they assume very little skew in the system to achieve good performance. Take for example the pair-wise exchange algorithm where each process goes through log(n) steps before synchronization is achieved. Every node is required to participate for all the steps to proceed. Thus, if one single node arrives late, the algorithm blocks until this last node has arrived. This tight coupling of the nodes also exists in the other algorithms like dissemination and gather broadcast. As a result, in presence of skew the above mentioned algorithms do not perform optimally.

In this paper we propose to use adaptive algorithms together with Infiniband's hardware multicast to implement efficient barrier and allreduce in varying skew scenarios. Further, these schemes perform comparably to the traditional algorithms in the absence of any skew. We have made use of a combining tree in our approach. The basic idea behind our scheme is to change the topology of the combining tree used in the collective call so that by the time last node arrives most of the barrier or allreduce would have been done. We use *Synchronization Delay* as the metric to evaluate the different approaches of doing barrier and allreduce. This is defined as the time difference between the last node leaving the barrier to the last node entering the barrier. The same kind of approach is taken in [1]. The average time spent in the collective is generally used to evaluate the performance of different designs. This is not a good metric on large size clusters as it includes the skew present in the system.

We have implemented our designs and integrated them into the MPI implementation over Infiniband. Compared to the existing schemes, our scheme can reduce the synchronization delay by a factor of 2.28 in the case of barrier and by a factor of 2.18 in the case of allreduce when sufficient slack is available. We expect to see much higher benefits for a large size cluster as we use hardware multicast which scales very well. Our design performs comparably to the existing schemes in no skew conditions. Also, we have evaluated different skew scenarios and our designs are significantly better than the current designs.

The rest of the paper is organized in the following way. In Section 2, we provide an overview of the Infiniband Architecture. In Section 3, we explain the motivation for our scheme. In section 4, we discuss detailed design issues. We evaluate our designs in section 5 and talk about the related work in section 6. Conclusions and Future work is presented in section 7.

2 InfiniBand Overview

The InfiniBand Architecture (IBA) [2] defines a switched network fabric for interconnecting processing nodes and I/O nodes. It provides a communication and management infrastructure for inter-processor communication and I/O. In an InfiniBand network, processing nodes and I/O nodes are connected to the fabric by Channel Adapters (CA). Channel Adapters usually have programmable DMA engines with protection features. There are two kinds of channel adapters: Host Channel Adapter (HCA) and Target Channel Adapter (TCA). HCAs sit on processing nodes.

The InfiniBand communication stack consists of different layers. The interface presented by Channel adapters to consumers belongs to the transport layer. A queue-based model is used in this interface. A Queue Pair in InfiniBand Architecture consists of two queues: a send queue and a receive queue. The send queue holds instructions to transmit data and the receive queue holds instructions that describe where received data is to be placed. Communication operations are described in Work Queue Requests (WQR), or descriptors, and submitted to the work queue. The completion of WQRs is reported through Completion Queues (CQs). Once a work queue element is finished, a completion queue entry is placed in the associated completion queue. Applications can check the completion queue to see if any work queue request has been finished. InfiniBand also supports different classes of transport services. In current products, Reliable Connection (RC) service and Unreliable Datagram (UD) service are supported.

InfiniBand Architecture supports both channel semantics and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, InfiniBand provides Remote Direct Memory Access (RDMA) operations, including RDMA write and RDMA read. RDMA operations are one-sided and do not incur software overhead at the remote side.

2.1 Hardware Multicast in InfiniBand

One of the notable features provided by the InfiniBand Architecture is hardware supported multicast. It provides the ability to send a single message to a specific *multicast address* and have it delivered to multiple processes which may be on different end nodes. Although the same effect can be achieved by using multiple point-to-point communication operations, hardware multicast provides the following benefits:

- Since only one send operation is needed to initiate the multicast, it greatly reduces host overhead at the sender. By reducing this overhead, multicast latency as seen by each receiver is also reduced.

- With hardware supported multicast, packets are duplicated by the switches only when necessary. Therefore, network traffic is reduced by eliminating the cases that multiple identical packets travel through the same physical link.

- Since the multicast is handled by hardware, it has very good scalability.

In InfiniBand, before multicast operations can be used, a multicast group which is identified by a multicast address must be created. Creating and joining multicast groups can be achieved through the help of InfiniBand subnet manager.

In InfiniBand, hardware multicast operation is only available under the Unreliable Datagram (UD) transport service. In UD, a connectionless communication model is used. Messages can be dropped or arrive out of order.

3 Motivation

The present schemes of doing barrier or allreduce give good performance when there is no skew. But in the presence of skew, this is no longer the case.

We consider barrier using hypercube based pairwise exchange algorithm as an example to illustrate the point. In the absence of skew it takes log(n) number of steps to achieve synchronization using this algorithm as shown in *b* of Figure 1. For a hypercube of dimension 3, this value is equal to 3. The nodes of the hypercube are named as shown in *a* of the figure. Now take the case when all the nodes do not call barrier at the same time. Assume that one node calls the barrier much later than the other nodes.Let node 7 be the last arriving node.

In the first step of the algorithm, all the nodes except node 6 exchange messages in a pair-wise manner. Node 6 is waiting for node 7 which has not arrived yet, c of Figure 1. The dotted lines indicate that the nodes have not exchanged messages so far. After the third step, we see that some of the nodes are blocked in step 1, some in 2 and others in 3. Now, when the last node arrives, it would take an extra 3 steps before the node 0 is released. The problem with this scheme is that that though most of the nodes call barrier very early, they do nothing other than waiting for the last node to arrive. Thus, the amount of steps required to complete the synchronization remain the same no matter when the last node arrives. The same problem exists in other schemes like the dissemination and the gather-broadcast algorithm. In the gather-broadcast algorithm, a combining tree is used to gather acks from all the nodes to the root of the tree. After collecting all the acks, the root releases all the other nodes. If one of the leaf nodes arrives late, it would take an extra number of steps equal to the height of the tree before the root can release everybody. Note that the latency of these extra steps gives us the synchronization delay of the barrier. It is of the order log(n) for all the schemes where n is the number of nodes involved in the barrier.

An effective strategy to minimize the synchronization delay would be to make the topology used in the algorithm adapt to the skew pattern. The decision on which topology to choose should be such that the nodes which arrive early do useful work to minimize the synchronization delay. Ideally, by the time the last node arrives, most of the barrier should have been done and very less time is spent between the last node's arrival and the release of all the nodes. This topology change can be dynamic where all the required steps to change the topology are performed in one single barrier call as it progresses. Or it could be semi-dynamic where the topology used in one barrier is based on the previous barriers done. Please note that though we are using barrier operation to explain the various design issues, the discussion applies to allreduce as well. One draw back of the semi-dynamic scheme is that it assumes that the skew pattern is fixed across the barriers. If this is not the case, it fails to give good performance. The totally dynamic scheme does not rely on any such assumption and is optimal in varying skew conditions.

We have used a token-based combining tree approach in our design to implement an adaptive scheme for barrier and allreduce. The basic idea in this approach is that the node who possesses the token is the root of the combining tree. It can either decide to pass the token to one of its children or release all the nodes. We explain in detail the various steps involved in the token based design in the following sections of the paper.

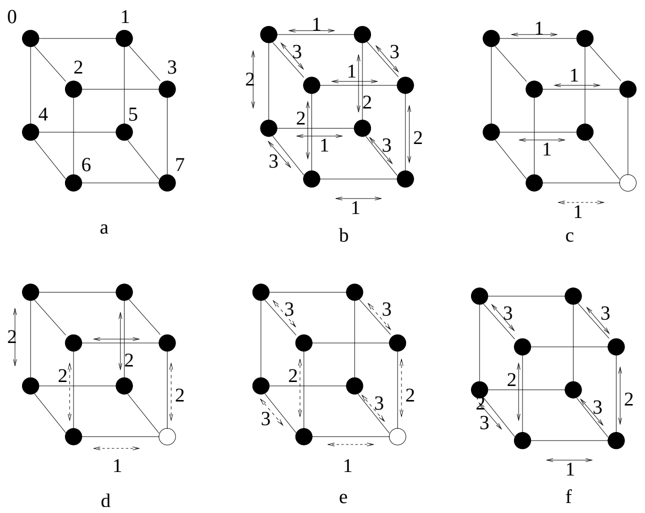

Figure 1. Hypercube based pair-wise exchange

4 Design of the Adaptive Algorithm

In this section we discuss in detail the various design issues involved in the adaptive scheme.

4.1 Basic Idea

In the standard gather-broadcast algorithm, the topology of the combining tree used is fixed. As previously discussed, the problem with using a static tree topology is that though the root arrived very early, it has to wait for acks from all of its children. Consider example 1, Figure 2 where all the nodes have arrived except for the last node. Further, all the nodes have sent acks to the root and are waiting for the release message from the root. The release message is posted only when the root has received the acks from both its children. As shown in the figure, it takes two more steps after the last node arrives before the root releases all the nodes.

Now take example 2 of the same figure. The root node now has an extra token which it keeps with itself until one of its children arrived. Soon after receiving the ack from node 1, the root passes on the token to its other child, node 2. Since node 2 has got hold of the token it is the new root and the tree topology changes as shown in *e* of the figure. As node 5 has already sent its ack, the token now moves on to the last node which has not arrived yet. As soon as this node arrives, it finds out that it has the token and releases everybody. Note that when the last node arrives, the tree looks as shown in *f*.

Using a token we were able to cut down the synchronization delay by two hops. Consider a cluster with large number of nodes where there are multiple level. In this case we would cut down the delay by the number of hops equal to the height of the tree. We use hardware multicast of Infiniband for sending the release message. This enables us to achieve constant synchronization delay for varying node sizes.

The only difference between our scheme and the standard scheme is the use of an extra token apart from the usual acks. However, unlike the earlier scheme where there is a fixed root the node holding the token becomes the root of the tree in our approach. In the example we have assumed a binary tree topology. We can easily extend this idea to any kind of tree topology. Also, it is possible that any node, irrespective of its position in the tree, become the owner of the token and hence the root of the tree.

4.2 Deciding the root of the tree

We now explain how the token is passed from the current node holding the token to one of its children. The current owner of the token after entering the collective starts polling

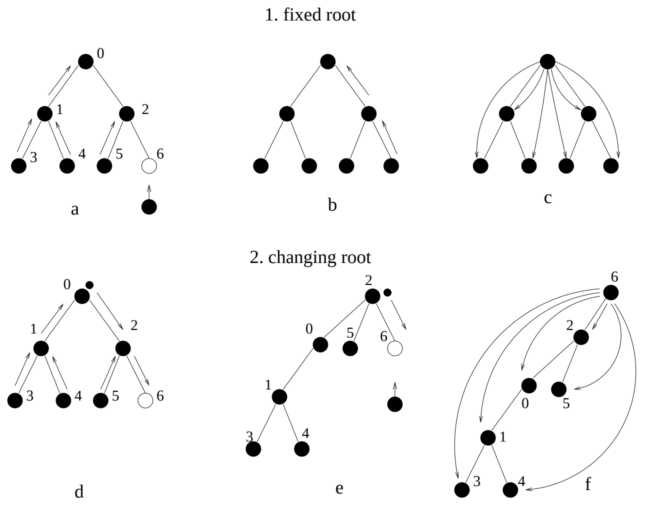

Figure 2. Adaptive vs Nonadaptive algorithm

for all its children. If all of them arrive, then all that it has to do is post a release message. If one of them has not arrived, then it passes the token to the child who has not arrived yet. To begin with, the token is present with node 0 who is the root of the tree. Note that the token is always passed from the node at a higher level to one of its children in the lower level. In the case of barrier, the token is an opaque object while in the case of allreduce, it carries data. We talk more about the details in the following subsections.

The nodes that are not involved in the token transfer behave the same way as in the standard combining tree algorithm. They ack the root as soon as they arrive. The intermediary nodes do the same once they collect the acks from the children. However, all these nodes have to wait for a token or a release message.

Other issues to be resolved are, how does the node know that it holds the token and how is it passed from one node to the other. Another problem faced by this scheme is that of race conditions. Consider this scenario where both the node holding the token and last of its children have arrived at the same time. In this case, the token from the parent and the ack from the child are exchanged simultaneously. This leads to race conditions which have to be avoided to make the implementation foolproof .Another issue to be discussed is that the hardware multicast is unreliable. The release message may not reach some of the nodes in which case they are kept waiting.We discuss all these issues in the following detailed design section.

4.3 Detailed Design Issues

4.3.1 RDMA approach to handle the Token and Acks

We use the remote direct memory access (RDMA) operation of Infiniband to do the token transfer. In our approach, we have used a one byte flag for both the token and ack in the case of barrier. For allreduce, the token as well as the ack additionally carry a data vector apart from the flag.

By using the RDMA operation the transfer of the token becomes straight forward. All we have to do is to RDMA write into an appropriate location in the remote memory buffer when we want to transfer the token or an ack. But prior to this, we have to exchange the necessary information about the buffers among all the participating nodes. Each node polls on the local memory to check for the token or the acks.

In the allreduce case, each node apart from the leaves has to do some computation before it can pass along the token or the ack. The leaf only includes its data vector in the ack and passes it to its parent. However, if an intermediary node has to pass an ack to its parent or a token to its child, it first computes a data vector from the acks received from the children. It then includes it in an ack or token and passes it to the appropriate node.

4.3.2 Avoiding Race conditions

Race conditions are possible in the implementation of the token-based approach. This is because many combinations of the token-ack transfers are possible based on the skew of the system. Take for example a simple case where both the node holding the token and its last child have arrived at the same time. In this case, the parent passes the token to its child and similarly the child acks the parent. If the child arrived a little late, it would have got the token and it need not have acked the root. The same is the case with the parent.

We take care of this race condition by using a simple technique. We make the child poll both for the token and the release message at the same time even though it has passed its ack to its parent. The parent on the other hand ignores the ack from its child and waits only for the release message. The child after it finds out that it has the token posts a release message immediately.

4.3.3 Reliability

Reliability is another issue which has to dealt with as we are using hardware multicast of Infiniband to post the release message in our implementation. We cannot be sure that all the nodes have received the message unless we add some mechanism to make the multicast reliable.

Reliability can be added by making use of a timeout and retransmission mechanism. The root stores all the posted release messages indexed by the count of the barrier or allreduce operation. It stores them until it receives an ack from all the other nodes. A sliding window mechanism can be employed at the root so that it doesn't block waiting for the acks. For more details refer to the paper on MPI_Bcast using hardware multicast support [5]

4.3.4 Flow control

Since we use RDMA based approach to transfer the acks and the token we need not post any extra descriptors on the remote node in this case. However hardware multicast requires that descriptors be preposted on all the nodes for the message to be received.

The way we do this is initially we post a certain number of descriptors. After every barrier operation we post an extra descriptor to make sure that there are sufficient number of descriptors available always.

5 Performance Evaluation

In this section we evaluate the different designs of doing barrier and allreduce. We compare the results of our adaptive scheme with the existing approaches for barrier and allreduce. We have used synchronization delay as an important metric in most of our micro-benchmark tests.

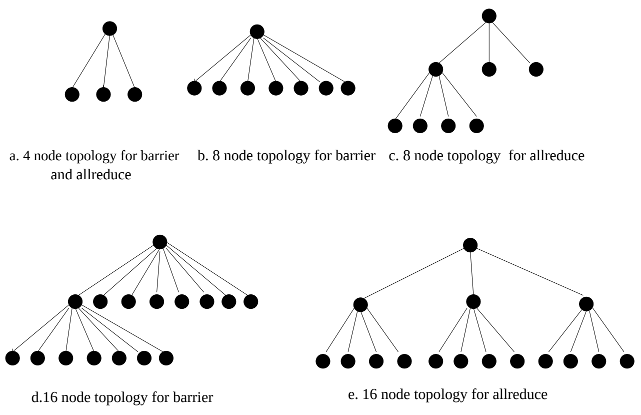
a. 4 node topology for barrier and allreduce
b. 8 node topology for barrier
c. 8 node topology for allreduce
d.16 node topology for barrier
e. 16 node topology for allreduce

Figure 3. Combining trees for barrier and allreduce

The different schemes considered for barrier are as follows:

- adaptive: Our implementation of an adaptive barrier using a combining tree of degree 8. Please refer to the Figure 3
- nonadaptive: a nonadaptive barrier using a combining tree of degree 8.
- dissemination: a dissemination based barrier

The different schemes considered for allreduce are as follows:

- adaptive: Our implementation of an adaptive allreduce. The exact topology used is as shown in the Figure 3
- nonadaptive: a nonadaptive using the same combining tree as above.
- pair-wise: allreduce based on the pair-wise exchange algorithm

5.1 Experimental Testbed

Our testbed cluster consists of 8 SuperMicro SUPER X5DL8-GG nodes with ServerWorks GC LE chipsets. Each node has dual Intel Xeon 3.0 GHz processors, 512 KB L2 cache, and PCI-X 64-bit 133 MHz bus. We have used InfiniHost MT23108 DualPort 4x HCAs from Mellanox. All nodes are connected to a single Mellanox InfiniScale 24 port switch MTS 2400, which supports all 24 ports running at full 4x speed. The kernel version we used is Linux 2.4.22smp. The InfiniHost SDK version is 3.0.1 and HCA firmware version is 3.0.1. The Front Side Bus (FSB) of each node runs at 533MHz. The physical memory is 1 GB of PC2100 DDR-SDRAM. The compilers we used were GNU GCC 2.96 and GNU FORTRAN 0.5.26.

In the following parts of the section we talk in detail about the various tests performed and discuss the results.

In the first subsection we use the standard micro-benchmark, *Average latency* to compare the different schemes used to implement the collective operation.Note that measuring the average latency of a collective is meaningful in situations where the skew is negligible. In the following subsection we use the *Synchronization Delay* as a metric to evaluate the above schemes in the presence of varying skew conditions.

5.2 Average latency for Barrier and Allreduce

To obtain the average latency of a barrier operation we measure the time taken for each node to perform barrier for a large number of iterations. We compute the average across all the nodes and iterations to obtain the average barrier latency. Average latency of allreduce is computed in the same manner.

In the latency test for the dissemination and pair-wise exchange cases, we can safely assume that all the nodes call barrier or allreduce at same time and also exit the collective call at the same time. However in the tree-based schemes, there is one node which exits earlier than the rest. This could result in skew, but on large clusters this is very small since the latency is averaged across all the nodes.

Figure 4 shows the average latency results for the barrier operation. As seen from the figure, the adaptive scheme latencies are slightly higher than nonadaptive ones though both schemes use the same tree topology. This is because we pay a penalty of one extra rdma write operation for some iterations. This case happens if the root node has passed the token to the last arriving child but it receives its ack immediately after. This results in one extra hop towards the child before it posts the release message. But, on large number of nodes, this penalty is too small to be observed.

Though the tree based schemes perform badly for small system sizes, they perform better as the size increases.The reason is the use of a higher degree fan-in for the combining tree and use of hardware multicast which scales very well over large number of nodes.

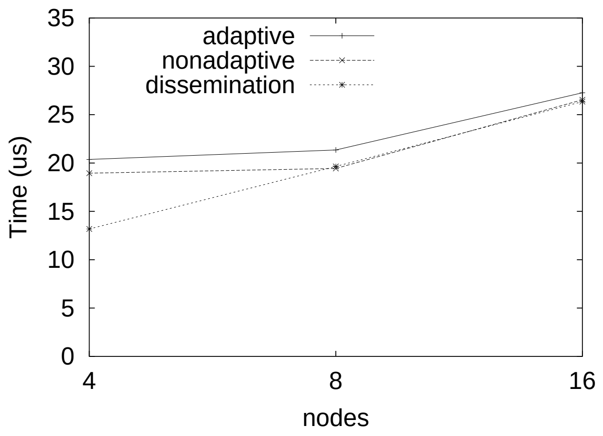

Figure 4. Average Barrier Latency across different schemes

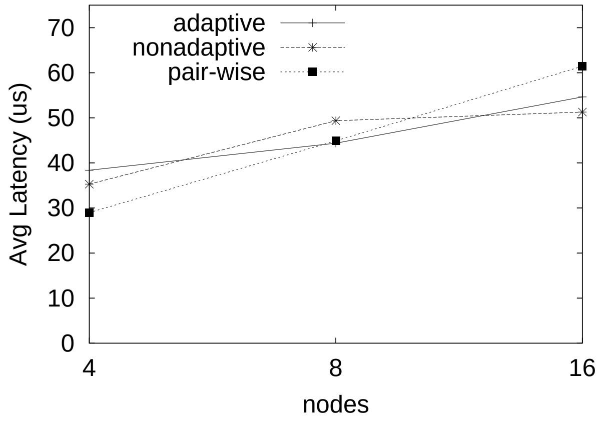

Figure 5. Average Allreduce Latency across different schemes

Figure 5 shows the average latency results for allreduce.

Notice that for node sizes of four and sixteen the adaptive scheme gives higher latencies. This is because for these topologies, the tree is totally balanced unlike the case with size eight. In balanced tree topologies, it is usually the case that we pay one extra hop of the token as the penalty. This is not the usual case in the unbalanced topology with eight nodes.

Notice that for large number of nodes, both the tree based algorithms perform better or comparably to the pair-wise algorithm in allreduce and dissemination in the case of barrier. More importantly, in the case of barrier, the adaptive scheme performs comparably to the nonadaptive one. In the case of allreduce, for an unbalanced topology the adaptive allreduce fares better.

5.3 Synchronization Delay

In this section we focus on the evaluation of different schemes where unlike the previous case, all the nodes do not arrive at the same time. We artificially delay some of the nodes by making these *target* nodes loop for the specified amount of time. We outline below our approach of measuring *Synchronization Delay* which we use to benchmark different schemes. Following that we present the results of different test cases showing the performance of the schemes under the presence of skew.

Measuring Synchronization Delay.Synchronization delay is the time difference between the last node entering the collective to the last node leaving the collective. The test to measure this delay consists of a loop where each of the nodes take turns to send back an acknowledgment using MPI_Send to the node arriving last. Each of the nodes can determine who is the last arriving node from the input parameters given to the test program. The last node starts a timer before calling the operation and stops it as soon as it receives the ack. It computes this time for all the nodes and computes the maximum. This maximum minus the latency of the MPI_Send gives the Synchronization delay for the test.

We now give an outline of the different kind of tests considered in this section. In the first test, we show the scalability of adaptive design compared to the other approaches. In the second test, we consider the scenario where one node always arrive late at the collective. In the third test, we extend this to multiple nodes arriving late. In the fourth and final test, we show instances of skew where the adaptive design does not give optimal performance and explain the approach to solve the problem.

5.3.1 Scalability study

To show the scalability of the adaptive design, we choose one node to be the *target* node and delay this node by the amount equal to twice the average latency of the collective considered. We perform this test for different system sizes. Also, we have chosen the highest rank node as the *target* node.

Figure 6 shows the results for barrier. From the figure we observe that the synchronization delay is constant across different node sizes for adaptive scheme. For nonadaptive case we see every extra level added to the tree increases this delay by one hop. This is because in the former scheme,the token arrives at the *target* node by the time it called barrier. The combining of acks would have been done and the node only has to post the release message. Since we are using hardware multicast, this phase of the barrier is a fast and scalable operation. In the case of barrier with dissemination, the synchronization delay is proportional to log(n) where n is the number of nodes involved in the barrier.

Figure 7 is the graph for allreduce. The same reason described for the barrier applies to allreduce as well. However, the synchronization delay, though remaining constant is higher. This is because, the release message now carries the final result vector of the allreduce operation and hence is greater in size. Moreover, the synchronization delay also includes costs of copying and the computation of the final data vector. We have used a vector of size 128 and of type MPI_DOUBLE in all our test cases. The operator used was MPI_PROD.

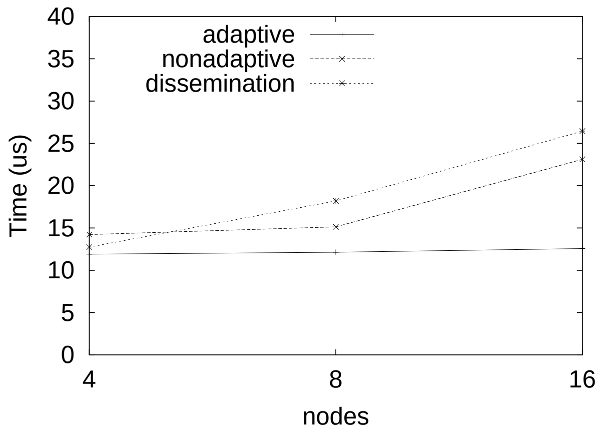

Figure 6. Synchronization Delay, Barrier

5.3.2 Single node arriving late

In this test, the *target* node is delayed by different amounts and the synchronization delay is calculated. All the tests are run on 16 nodes. Once again we choose rank 15 as the target node. This helps us to understand the behavior of adaptive scheme better.

Figure 8 shows the results for Barrier. In the adaptive case, as the delay is increased, we obtain smaller synchronization delays upto a point after which it remains constant. This is because initially, the token is not reaching the last

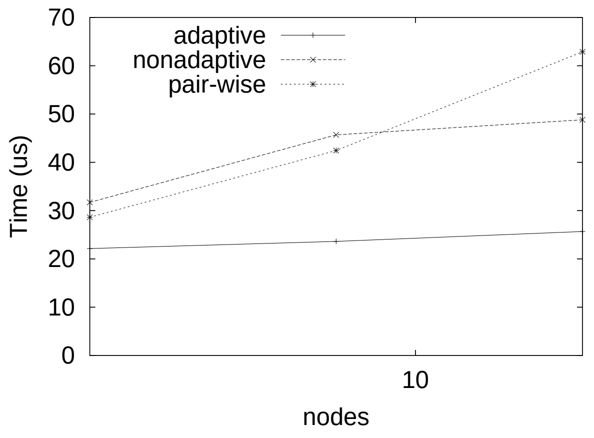

Figure 7. Synchronization Delay, Allreduce

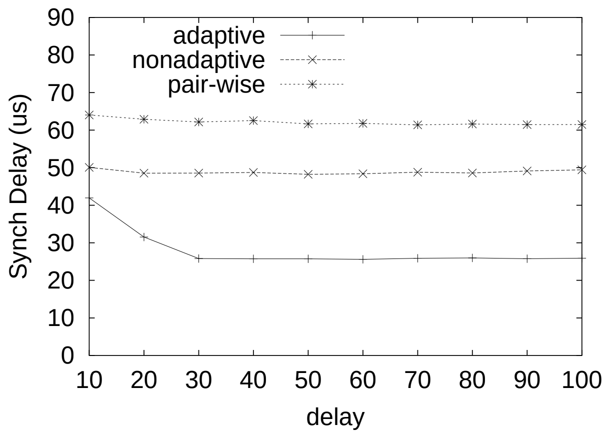

Figure 9. Single node arrives late, Allreduce

node as the delay is not sufficiently high. However, we perform better than the nonadaptive based scheme even in such cases. This is because, in the adaptive design, an intermediary node receives both the token from its parent and acks from its children. Thus, less time is spent by the ack of the *target* node to reach the node holding token unlike the non-adaptive design, where it has to travel all the way to the top. For larger delay we observe that the other schemes have the synchronization delay equal to the average latency. For the adaptive design, it is equal to the latency of hardware multicast.

Figure 9 shows the results for allreduce. The same trend is also observed in the case of allreduce too. However,the values we obtain are higher than in the case of barrier.We have already explained this observation in the previous test case.

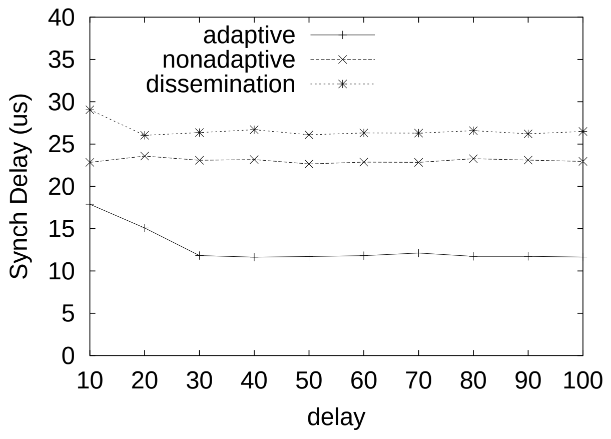

Figure 8. Single node arrives late, Barrier

5.3.3 Multiple nodes arrive late

In this section we concentrate on evaluating the different schemes when we use multiple *target* nodes. For a cluster of large size, there are many combinations of *target* nodes possible. Also, even after fixing the target nodes, these nodes may arrive in any order. It is not feasible to evaluate the schemes for all possible cases. However, in the case of barrier on 16 nodes we come with three representative cases which include all the skew patterns possible.

Observe that all the children under a given root are identical. That is, no matter what the pattern of arrival of these children is, the passing of the token is dependent upon the last two arriving children. The skew between these nodes decides who gets hold of the token irrespective of the arrival pattern of the other nodes. Thus we can choose two *target* nodes which are the last and second last arriving nodes.

In the topology used in barrier(Figure 3) we have two subtrees. Thus we have three cases to consider. Both the *target* nodes belong to the upper subtree, both belong to the lower subtree, one belongs to upper subtree and the other to the lower subtree.

Figure 8 can be used to demonstrate the the third and the second case. The way we obtained this results earlier was by skewing only one *target* node where as all the others arrived at the same time. The second *target* node can be assumed to be any one of these other nodes. However, in this case the second *target* node always arrives early. We consider the possibility of the second *target* node arriving early separately. From the figure, we conclude that if there is a reasonable amount of difference between the arrival times of the two *target* nodes, the adaptive barrier does significantly better than the other schemes.

Now consider Figure 11 to show the case where both the *target* nodes belong to the upper subtree. This also includes the missed case explained above. Note that in this case, the margin of improvement between the nonadaptive based scheme and the adaptive scheme has reduced. This is because the target nodes are now closer to the root than the earlier case.

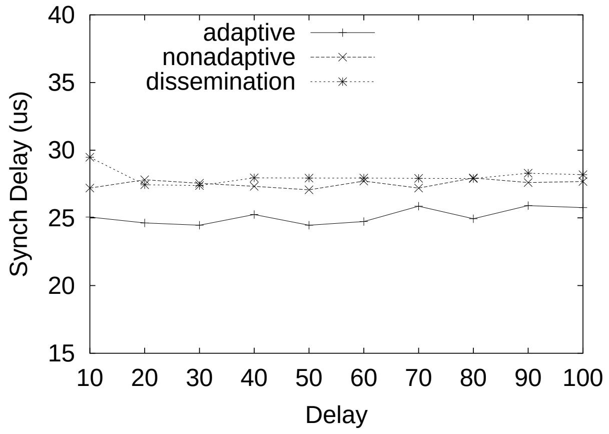

Figure 10. Two nodes arrive late, Barrier

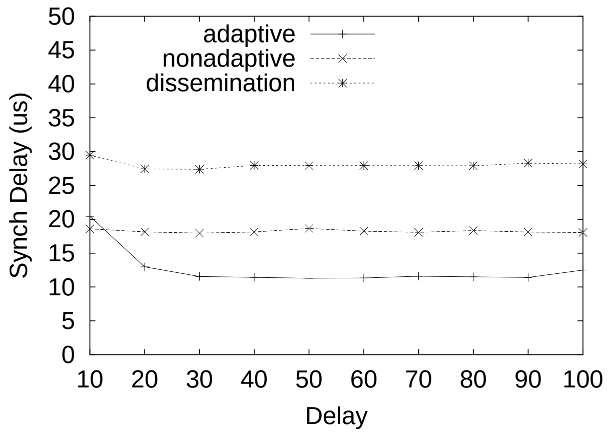

Figure 11. Single node arrives late, Barrier

The allreduce case is more complicated as there are more subtrees to be considered. In this paper we show only one set of combinations which we can conclude from Figure 9. It is the combination in which the one of the target nodes is a bottom node and the secondary node is any of the nodes in the tree.

5.3.4 Two nodes arriving equally late

We have taken this as a special case to show the adaptive scheme does not give optimal performance under this situation. We also show how this can be improved.

Figure 10 shows the results obtained for the barrier where the two nodes come late by the same amount of time.

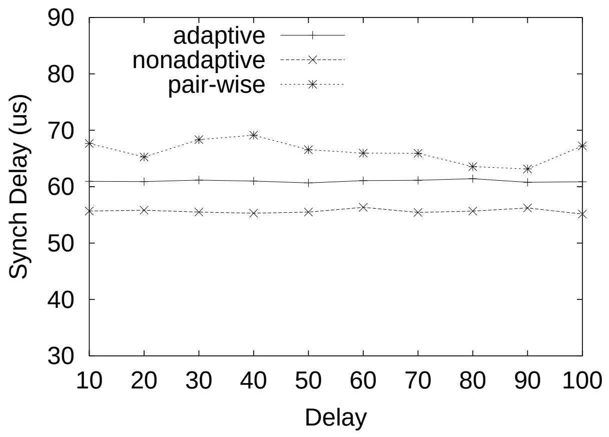

Figure 12. Two nodes arrive late, Allreduce

Also, the two nodes are the bottom most ones of the tree and belong to separate subtrees under the main root. We used nodes with ranks 1 and 15 as the *target* nodes for this case. The x axis shows the delay of both nodes. As shown in the figure, the synchronization delay remains the same in spite of the increase in the delays of these nodes unlike the earlier figures where it drops down to the hardware multicast latency.

The similar behavior is observed in allreduce case (Figure 12)

This behavior is because the token is immobile at the root when two of the nodes are arriving late. The root cannot decide to which branch it should pass along the token unless the second last child arrives. But in this case considered, the second last child arrives at the same time as the last child. So, the root is unable to make that decision. We can overcome this problem in our design, by introducing an extra token. Now the root can pass two tokens, one to the second last child and the other to the last child. This idea can be generalized to include multiple tokens. However, introducing too many tokens increases both the complexity of the design and adds more traffic into the system.

6 Related Work

Work in [1] deals with designing software barriers when there is skew in the system caused by the load imbalances. It explains why such load imbalances occur in a system and how it leads to processes being skewed. In this paper, the authors have come up with a semi-adaptive approach where the nodes which arrive late are placed closer to the root. This approach is different from our design which is totally dynamic. In their approach they use a prediction based scheme based on the recent history of the barriers done. This scheme has certain drawbacks as it works well when the skew pattern remains the same across the barriers. The approach we have taken in this paper is to use a dynamic tree topology which adjusts itself to the skew. Changing the topology of the tree has also been discussed in [8]. But they have used a counter based combining trees for SMP systems. The scheme discussed in this paper is targeted for large scale clusters which use Infiniband as the interconnect. The standard algorithms like pair-wise exchange are explained in [4] Other algorithms like the gather-broadcast and dissemination are explained in [3]. We have shown in the paper the limitations of using such algorithms in systems having lot of skew. Hardware multicast is also evaluated in this paper and [5]. More details about the hardware multicast are found at [2],[6],[7]. Other interconnects like Quadrics provide hardware multicast[10]. In [12] this feature is used to implement broadcast. However hardware multicast in quadrics functions for nodes that are contiguous. NIC-level multicast is studied in [13]. This is different from the hardware multicast of Infiniband. In NIC-level multicast, the broadcast operation is handled by the NIC instead of the host.

7 Conclusions and Future Work

The standard algorithms like pair-wise exchange, dissemination and gather broadcast do not perform optimally when there is skew in the system. This is because the nodes participating in this algorithms are tightly coupled with each other in all the steps of the algorithm. The design presented in this paper removes this limitation by making the tree topology adapt dynamically to the changing skew scenarios. We have used an adaptive root mechanism where the last arriving node becomes the root of the tree if the skew is sufficiently large. We have used hardware multicast in the release phase of our algorithm. From the results we have showed that the design presented in this paper scales very well as the number of nodes increases. We obtained a synchronization delay of 12 us in the case of Barrier which is close to the hardware multicast latency. This number is constant for varying system sizes. Using our scheme we reduce the synchronization delay by a factor of 2.28 for Barrier and by a factor of 2.18 for allreduce. We also show that the adaptive design performs comparably when there is no skew. Different skew scenarios were discussed in the paper and we show that our adaptive design either performs better or comparable to the existing skew conditions.

In our future work, we would like to extend the idea of a single token to multiple token and evaluate its performance. Also we would like to evaluate our scheme over a larger test bed. We expect to significant performance gains for a large cluster. We would also like to take a closer look at the semi-dynamic algorithm and evaluate its performance.

8. Acknowledgements

We are thankful to Gopalakrishnan Santhanaraman and Sundeep Narravula for their valuable comments.

References

[1] A. E. Eichenberger and S. G. Abraham. Impact of load imbalance on the design of software barriers. In *Proceedings of ICPP*, pages 63–72, 1995.

[2] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.1. http://www.infinibandta.org, November 2002.

[3] S. P. Kini, J. Liu, J. Wu, P. Wyckoff, and D. K. Panda. Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters. In *EuroPVM/MPI*, Oct. 2003.

[4] V. Kumar, A. Grama, A. Gupta, and G. karypis.

[5] J. Liu, A. R.Mamidala, and D. K. panda. Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support. In *Proceedings of IPDPS*, 2004.

[6] Mellanox Technologies. Mellanox InfiniBand InfiniHost MT23108 Adapters. http://www.mellanox.com, July 2002.

[7] Mellanox Technologies. Mellanox VAPI Interface, July 2002.

[8] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM ToCS*, 9(1):21–65, 1991.

[9] NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[10] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, 2002.

[11] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI–The Complete Reference. Volume 1 - The MPI-1 Core, 2nd edition.* The MIT Press, 1998.

[12] W.Yu, S.Sur, D.K.Panda, R.T.Aulwes, and R.L.Graham. High Performance Broadcast Support in LA-MPI over Quadrics. In *Las Alamos Computer Science Institure Symposiun,(LACSI'03)*, Oct 2003.

[13] W. Yu, D. Buntinas, and D. K. Panda. High Performance and Reliable NIC-Based Multicast over Myrinet/GM-2. In *Int'l Conference on Parallel Processing, (ICPP 2003)*, Kaohsiung, Taiwan, October 2003.