High Performance Broadcast Support in LA-MPI over Quadrics[*]

Weikuan Yu[‡] Sayantan Sur[‡] Dhabaleswar K. Panda[‡] Rob T. Aulwes[§] Rich L. Graham[§]

Network-Based Computing Laboratory[‡]
Dept. of Computer and Information Science
The Ohio State University
{yuw,surs,panda}@cis.ohio-state.edu

Los Alamos National Laboratory[§]
Advanced Computing Laboratory
Los Alamos, NM 87545, USA
{rta,rlgraham}@lanl.gov

Abstract

LA-MPI is a unique MPI implementation that provides network-level fault-tolerant message passing. This paper describes the efficient implementation of a scalable MPI broadcast algorithm. LA-MPI implements a generic version of the broadcast algorithm using a spanning tree method built on top of point-to-point messaging. However, the Quadrics network, with it's hardware broadcast support, provides an opportunity for a much more efficient implementation of this collective. We describe the design challenges encountered while making use of the hardware broadcast capability, explore design alternatives and describe the approach taken to design a low-latency, highly scalable, fault-tolerant broadcast algorithm. Our evaluation shows that this implementation reduces broadcast latency and achieves higher scalability relative to the generic version of this operation. In addition, we observe that performance of the implementation is comparable to that of the high performance implementation by QSW [13] for MPICH, and HP's for Alaska MPI, while providing fault tolerance to network errors not provided by these.

1. Introduction

LA-MPI [6] is a high performance thread-safe implementation of the Message Passing Interface (MPI) [9] library. It is designed for fault-tolerant message passing over terascale clusters. The current implementation of LA-MPI provides end-to-end network-level fault tolerance to the failures such as I/O bus errors, network interface errors and wire-transmission errors [1]. Its lower-level network transport has been implemented over a number of different transport protocols, including UDP over IP, HIPPI-800 OS bypass, Quadrics Elan3 RDMA [13], Myrinet GM [10], and shared memory.

A high performance MPI implementation requires that attention be paid to the performance of the collective routines. Broadcast is an important collective operation, which can be used to implement other collective operations, such as allreduce, barrier and scatter.

Efficient support of broadcast operation is an active research topic [2, 3, 14]. Message passing standards, such as MPI [9], often have it included as a part of their specifications. The Quadrics network provides a hardware broadcast primitive, which can be used to build efficient, reliable and scalable implementation of

[*]This research is supported in part by a grant from Los Alamos National Laboratory. Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36.

the MPI broadcast operation. Using this hardware broadcast primitive, a high performance broadcast operation has been implemented by QSW [13] for MPICH [7], a widely used implementation of MPI standard, and by HP's for Alaska MPI. However, these implementations ignore the issue of end-to-end reliable data delivery, this is left to the Quadrics hardware.

The current LA-MPI implementation is generic, using point-to-point messaging and a spanning tree algorithm to implement the broadcast. Fault tolerance to data transmission errors is already provided by the point-to-point messaging layer. In this paper we describe how we use the hardware broadcast primitive provided by the Quadrics network to implement an efficient MPI broadcast function, while providing fault tolerance to data transmission errors. We start by describing the benefits and limitations of Quadrics hardware broadcast support, the associated challenges in incorporating this functionality into LA-MPI. Then we explore the design challenges to use Quadrics hardware broadcast to achieve low latency and high scalability while providing network-level fault tolerance. Accordingly, we have implemented the broadcast algorithm with the best suitable strategies. Our evaluation shows that the new algorithm achieves a latency of 8.9μs when broadcasting a 4-byte message over 256 processes on a 64-node system.

The rest of the paper is organized as follows. We give an overview of Quadrics network, its programming libraries and the hardware broadcast primitive in Section 2, and an overview of LA-MPI in Section 3. In Section 4 we then describe the design challenges for the incorporation of the hardware broadcast into LA-MPI. In Section 5, we describe our broadcast algorithm and its implementation, and in Section 6, we describe the performance evaluation of this algorithm. Finally, in Section 7 we summarize our research and discuss future work directions.

2. Overview of Quadrics

Quadrics network (QsNet) [11] provides low-latency, high-bandwidth communication with its two building blocks: a programmable Elan network interface card and the Elite switch, which are interconnected in a fat-tree topology.

QsNet Communication and Hardware Broadcast – A parallel program over QsNet consists of processes with different virtual process identifiers (VPIDs). Interprocess communication is supported by an efficient model, remote direct memory access (RDMA). To enable RDMA, each process maps a portion of its virtual address space into the network interface, allowing access from remote processes via the communication processor. RDMA can then transfer data over the network at a low latency without requiring the involvement of remote processes. An efficient, reliable and scalable hardware broadcast is also supported over QsNet [13]. A sender process performs this by addressing a set of processes with a single preallocated VPID. As shown in Fig. 1, a hardware broadcast packet takes a predetermined path to reach all the recipients. It is successfully delivered only when all the recipients send an acknowledgment. The top Elite switch in the path takes care of broadcasting the packets to and combining the acknowledgments from the recipients [12]. However, the hardware broadcast has its own restrictions. The processes being addressed must have contiguous VPIDs in order for the switch to perform the broadcast. In addition, the destination addresses in a hardware broadcast have to be the same across all the processes being addressed.

QsNet Programming Library – QsNet provides the Elan and Elan3 libraries as an interface for its Elan3 network [13], with the Elan3 programming library (libelan3) being the lowest-level programming library. At the Elan3 level, a process in a parallel job is allocated a virtual process id (VPID) and maps a portion of its address space into the Elan interface. These processes can then communicate with each other through an abstraction of distributed virtual shared memory, formed by the mapped address space. A higher-level programming library, libelan, extends the libelan3 with point-to-point, tagged message passing primitives (called Tagged Message Ports or Tports) and support for collective operations. This programming interface

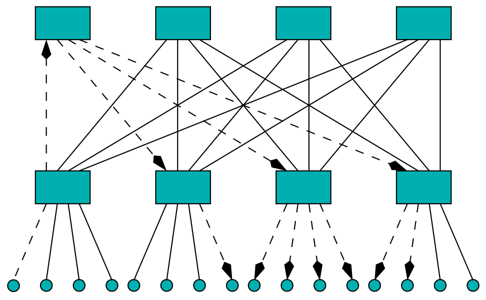
Figure 1. Hardware Broadcast

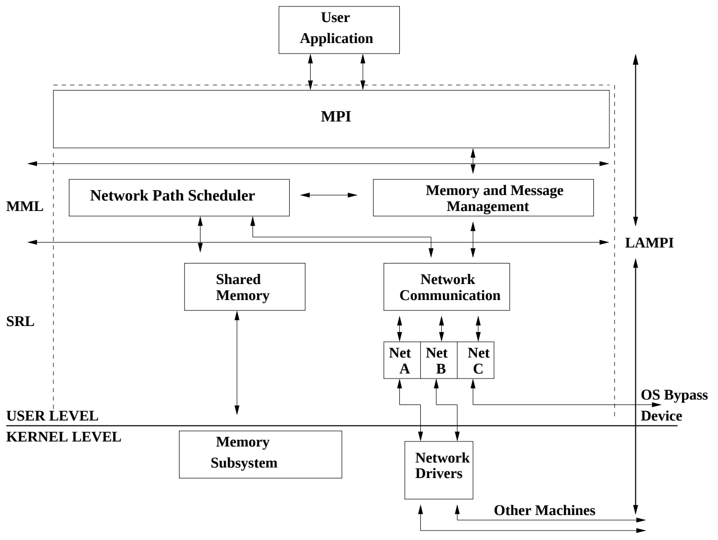

Figure 2. LA-MPI Architecture

is very similar to MPI [7]. Both the collective support and the Tport programming interface are used by QSW [13] to implement the high performance MPI support of MPICH [7].

Implementation of Broadcast in libelan – A broadcast operation in a parallel program can involve an arbitrary set of processes, which may not have contiguous VPIDs. The destination address in each process may not be the same, or may not even be elan addressable. Currently, the communication processor in the elan network interface supports only a 32-bit virtual address space [13]. These limitations impose challenges to developers who want to make use of the benefits of the hardware based broadcast communication. The libelan library provides elan_bcast() function which takes advantage of Quadrics hardware broadcast while overcoming its limitations. To use the hardware broadcast, the current implementation of elan_bcast() provides two broadcast buffers, which are used alternatively by the broadcast operations. To avoid the incoming broadcast messages clobbering the existing message with RDMA, elan_bcast() forces synchronization before the start of every broadcast operation. Consecutive broadcast operations are then throttled by this inserted synchronization, which can lead to a low broadcast throughput. Although it is not possible to eliminate the synchronization, it would be beneficial if its frequency can be reduced. Since network-level fault-tolerance is the goal, LA-MPI is developed from the lowest elan3 programming library and provides a different communication protocol from libelan. Thus the libelan collective support cannot be used by LA-MPI for its collective communication.

3. Overview of LA-MPI

In this section, we give some background information of LA-MPI. As shown in Fig. 2, the implementation of LA-MPI has its MPI interface layered upon a set of User Level Messaging (ULM) interface primitives, which itself consists of two layers: the Memory and Message Layer (MML) and the Send and Receiver Layer (SRL) [6]. The MML provides message management services including message routing (i.e., network path selection), message tag matching, buffer allocation for uniform and non-uniform memory access machines (NUMA), message retransmission, and message status tracking. The SRL is reponsible for sending and receiving message fragments through different network adapters and shared memory, and is highly network dependent.

3.1. The Memory and Message Layer

The memory and message layer is a set of common abstractions to ensure reliable message delivery and matching while minimizing the overhead of memory management for LA-MPI data structures and buffers [6]. The MML layer is composed of a memory manager, a set of network paths, and a path scheduler. The memory manager controls all memory (physical and virtual), including the process private memory, shared memory, as well as "network memory", such as memory on the NIC. A network path is a homogeneous transport abstraction used to encapsulate the properties of different network devices and protocols. A path controls access to one or more network interface cards (NICs), within a path there may be several independent "routes". The path scheduler "binds" a specific message between a pair of source and destination processes to a particular path, so that different messages between the same end-points may use different paths. In short, all three components of the MML architecture, together, provide support to MPI functionalities with network transmission-specific primitives (i.e., the SRL).

3.2. The Send and Receiver Layer

The Send and Receive Layer (SRL) is responsible for sending and receiving messages. It consists of multiple network path implementations and a highly optimized shared memory communication implementation. The SRL layer supports message fragmentation and reassembly. Messages that do not require the network (on-host) are simply copied through shared memory. Those that do require the network (off-host) are handled by the Network Communication module, where the message fragments are sent via physical resources associated with the path, to which the message is bound. In addition, this layer handles the in-order delivery required by the MPI standard. Data that arrives out-of-order is queued for later processing as unexpected data.

4. Design Challenges

Designing a general purpose MPI broadcast routine to take advantage of the hardware broadcast capabilities presented by the Quadrics interconnect presents several design challenges. These include global buffer allocation and management, broadcast to an arbitrary set of processes which may not have contiguous VPIDs, acknowledgment and synchronization, retransmission, and reliability. In this section, we explore each of these issues and discuss several strategies to overcome them.

4.1. Global Buffer Allocation and Management

Hardware broadcast can only address a destination address that is global across the set of receiving processes. This necessitates the use of global memory. In order to do this, an identical elan3 address space across all processes is allocated and then mapped to the main virtual memory by each process. The consistency of the global memory has to be maintained. Two alternatives can be employed to maintain the consistency. One is to create a global allocator that enforces the consistency and each memory request will use this global allocator as a handle. The other is to divide the global memory into an array of global buffers and use different buffers for broadcast operations sequentially. The first alternative can allocate memory for larger number of requests. Correspondingly, the NIC has to perform the increasing number of address translation for these requests. Since Quadrics communication is very sensitive to the pattern of address translation [8], it would be better to choose the second alternative if one wants to minimize the number of address translation requests and, hence, reduce their effects. It is also easier to manage the memory by dividing them into a small number of buffers. The maintenance of the global memory consistency is also discussed later in Subsection 4.3.

4.2. Broadcast to Arbitrary Set of Processes

MPI broadcast operation can involve an arbitrary set of processes while Quadrics hardware broadcast can only address a set of processes with contiguous VPIDs. It is a design challenge to broadcast to an arbitrary set of processes. There are several design alternatives for such a broadcast pattern. One alternative is to partition the processes into disjoint subsets of contiguous processes and perform a broadcast operation to each of these sets. This allows the utilization of the existing hardware broadcast potential while paying the cost of a linearly increased latency with respect to the number of multicast subgroups. At its extreme, the broadcast latency within a group of sparsely distributed processes can increase linearly with respect to the group size. Another alternative is to partition the processes into disjoint sets of processes with contiguous VPIDs. Then a tree-based algorithm can be used to perform the broadcast with each unit being a set of processes with contiguous VPIDs. After receiving the data, processes in one set can forward the data to other sets. For fast expansion of the number of processes that can broadcast the data, the sets of processes can be chosen to receive the data in the order of their sizes. This alternative has been explored by Coll et. al. [4]. The tree topology, and so its corresponding data structure, is rather complex, compared to the first alternative. In addition, the resource management framework (RMS) of Quadrics network tends to allocate processes for a parallel program from the first available network nodes. This approach usually leads to a good contiguity in the base group (the group of processes MPI_COMM_WORLD starts with). Moreover, the use of the base group (MPI_COMM_WORLD) by collective operations is very common in MPI programs. Thus the first alternative can be sufficient for the purpose of exploiting the advantages of the hardware broadcast.

4.3. Acknowledgment and Synchronization

Since the data can be written to a remote process without its knowledge, it must be ensured that a new message does not clobber the previous message that is still in the global buffer. Thus synchronization across the processes is needed before a broadcast operation reuses a global buffer.

One alternative for the synchronization is to use a tree-based algorithm to have a process collect all the acknowledgments and then update others with the combined results. Another is to use a special form of Quadrics network transactions, test-and-set, to perform a test on a global value [5, 13]. This transaction tests a value against the global value at various processes, and it optionally writes a new value to the global address [12]. Using this transaction for the synchronization, one can have all processes write a number to a dedicated global address, and wait for the number to be updated. This synchronization is very efficient when all the processes reach the synchronization fairly close to each other. However, an exponential back-off scheme must be in place to avoid too much broadcast traffic incurred by the test-and-set transaction [12, 13]. In addition, the timing of the synchronization is also important. The synchronization can be delayed if there are still global buffers available. Then the delayed synchronization cost can be amortized over a set of broadcast operations. Using the first alternative along with a sufficiently large set of global buffers can be a better choice, since it avoids potentially too much broadcast traffic and its cost can be insignificant when amortized.

4.4. Retransmission and Reliability

Unlike many MPI libraries that consider all underlying communication perfectly reliable, LA-MPI is designed to tolerate the failures of the PCI bus and the network [6]. The errors can be propagated and the effect of these errors can be amplified in long running parallel programs over terascale clusters, because of the sheer number of their components. To achieve reliable end-to-end message passing, LA-MPI optionally supports sender-side retransmission when the messages have exceeded their timeout periods for point-to-point messages. A similar sender side retransmission can be employed to achieve reliable broadcast. There are two different cases to be considered depending on whether Quadrics hardware broadcast is still available for the retransmission of the broadcast messages or not.

When the hardware broadcast is available, every broadcast message is transmitted along with a main memory-to-main memory 32-bit additive checksum or 32-bit cyclic redundancy code (CRC) (if it is needed). This checksum/CRC protects the message against network and I/O bus corruption. When the message is received by the recipients, its checksum/CRC is validated before the recipients acknowledge the arrival of data. If the verification is successful, the recipients generate their positive acknowledgment, ACKs. Otherwise, negative acknowledgments, NACKs, are generated. The ACKs and NACKs are combined during the synchronization and the corresponding sender processes are notified after the synchronization. A timestamp and the number of retransmission times are also generated with every broadcast message. The loss of a message can be detected by the sender or the receiver when it has exceeded its timeout period. An exponential back-off scheme is used to avoid flooding the network with broadcast traffic.

When the hardware broadcast is no longer available or its failure is suspected, a fail-over function must be provided to detect this and rebind all the outstanding broadcast operations to another broadcast operation on top of point-to-point operations. The fault tolerant point-to-point operations in LA-MPI can then ensure the completion of the outstanding broadcast operations.

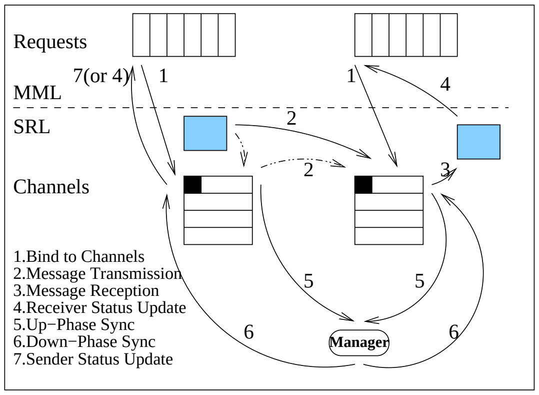

Figure 3. Broadcast Flow Path Using Quadrics Hardware Broadcast

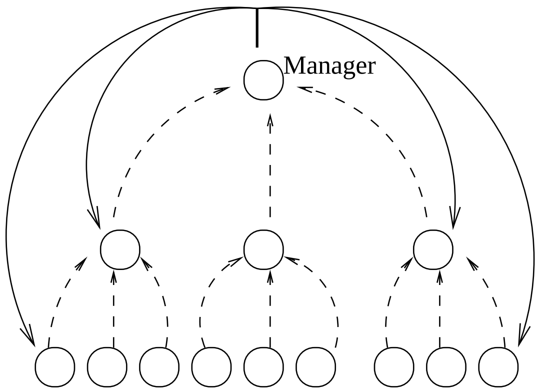

Figure 4. Synchronization of the Broadcast Operation

5. Our Implementation

Memory Allocation and Request Binding – As shown in Fig. 3, we implemented the broadcast operation with added support in both the MML and the SRL in LA-MPI. To make use of the hardware broadcast, we chose the second alternative described in Subsection 4.1 to allocate global memory and divide it into a number of channels. As shown in Fig. 3, a broadcast operation binds to one of the channels when they are available. This is shown as Step 1 in the MML for both the senders and receivers.

Message Transmission – The SRL is extended with functionalities to take care of the broadcast of messages. At the sender side, the sender generates CRC/checksum and creates a message header at the beginning of the channel. If the message is not elan addressable or if it is less than 16KB, it is copied over and appended to the header. Then this message is transmitted over the network with hardware broadcast. To avoid the copying overhead, when the message is large than 16KB and also elan addressable, it is transmitted as a separate message before the header. This is shown as Step 2 in Fig. 3. The dashed line shows the separate data message. When the recipients do not have contiguous VPIDs, the sender partitions them as disjoint groups and allocates a broadcast VPID for each group. The broadcast message is transmitted to each group in a chained manner, using chained DMA provided by Quadrics [13].

Message Reception and Status Update – When the broadcast message arrives at the receiver end, its tag is matched against the corresponding broadcast request at the receiver' side. If it is the expected message, the data is copied into the destination buffer, and CRC/checksum is generated and compared to the CRC/checksum contained in the header. If the received data is not corrupted, the request status is updated as completed, as shown in Fig. 3 (Step 3 and 4 at the receiver side). If the message is not expected or its data is corrupted, the message is dropped and a NACK is generated to signify the error. The NACK is detected and corrected later during the synchronization. At the sender side, if the message is buffered in the channel buffer, it also updates the MML request status as completed, shown in Fig. 3, Step 4 at the sender side. Otherwise, it has to wait until it is notified that all receivers have received the data through synchronization.

Synchronization and Status Update – As discussed in the subsection 4.3, we chose to take the tree-based algorithm for the synchronization, and leave the hardware global test-and-set for later optimization. We use a delayed synchronization scheme, in which the synchronization is only triggered when the system is running out of broadcast channels. The use of broadcast channels is also synchronized. As shown in Fig. 4, we use a balanced tree to perform UP phase synchronization following the approach described in [12, 13]. Every process updates its status to its parent with an ACK or a NACK, and the manager process at the root of the tree combines all the acknowledgments. This is shown in Fig. 3 as Step 5. During the DOWN phase of synchronization, the manager process broadcasts its result using the same broadcast approach described in Subsection 4.2. This is shown in Fig. 3 as Step 6. The two-phase synchronization is also shown in Fig. 4 in more detail. With successful synchronization, each process updates its channel state and progresses to the next round of broadcast operations and synchronization, and so on.

Retransmission and Reliability – When a NACK is received, the manager process makes no distinction about which operation has been successfully transmitted and it notifies all the processes that there is a transmission error since last synchronization. A process then retransmits a message, for which it is the sender. A receiving process that has received data correctly will ignore the repeated message based on its channel states. But it still cannot reuse the channel because the synchronization fails to update its channel state and releases the channel. This approach implies that the retransmission process will be very time consuming and possibly repeat some successfully completed broadcast operations. Taking into account of the high reliability of the Quadrics interconnect, this is acceptable because it can speed up the successful transmission by freeing processes from frequent synchronization. And since the possibility for the retransmission is very rare, the cost for a possible retransmission can very well be contained by the benefits from saved synchronization. When a message happens to be lost, the sender (or the receiver) detects it with its timestamp, or the manager can detect it when it fails to make progress through synchronization. In either case, the retransmission gets triggered for the broadcast operation. If there is a failure of the hardware broadcast communication, a generic broadcast operation on top of point-to-point operations takes care of the failed broadcast operations.

Shared Memory Support – LA-MPI has a highly optimized shared memory communication implementation. Processes on the same node exchange messages through the shared memory. For a broadcast operation, one process on an SMP node first participates in the off-host communication that requires the network. Once the off-host communication is completed, these processes perform an on-host communication with the other processes on the same node through the shared memory. This shared memory support is also incorporated into the hardware-based broadcast algorithm.

6. Performance Evaluation

In this section, we describe the performance evaluation of the broadcast algorithm. We have evaluated the implementation on a TRU64 quad-1.25GHz alpha cluster, which is equipped with Quadrics interconnect, composed of a dimension four switch, Elite-256, and QM-400 cards. On the same system, we have also measured the performance of the broadcast implementation by QSW for MPICH [7], and HP's for Alaska MPI. For the experiments in the subsections 6.3 and 6.4, we have used an eight-node cluster of quad-700MHz Pentium-III. An Elan3 QM-400 card is attached to each node and links to a quaternary fat tree switch of dimension two, Elite-16.

Our tests are performed by having the processes first warmed up with 20 broadcast operations. Then a sample size of 1000 broadcast operations are performed after a barrier synchronization. This is repeated for

100 samples, each again performed after a barrier. The time for performing each sample is recorded. With statistical analysis on these samples, we derive the average time for a broadcast operation as the latency. The same test is used to measure the performance of MPICH and Alaska MPI broadcast operation, which uses the hardware broadcast communication.

6.1. Broadcast Latency

We measured the broadcast latency over 128 processes on a contiguous 32-node cluster. As shown in Fig. 5, the new algorithm in LA-MPI (LA-MPI HW) significantly reduces broadcast latency, compared to the original algorithm. This is to be expected because the new broadcast algorithm takes advantage of the hardware broadcast. As also shown in Fig. 5(a), this broadcast algorithm outperforms the algorithms implemented for MPICH by QSW [13] and HP's for Alaska MPI. This is because, for messages $\leq$ 16KB, the new algorithm reduces the frequency of global buffer synchronization to only once every 16 broadcast operations. Hence, the synchronization cost is amortized across different broadcast operations.

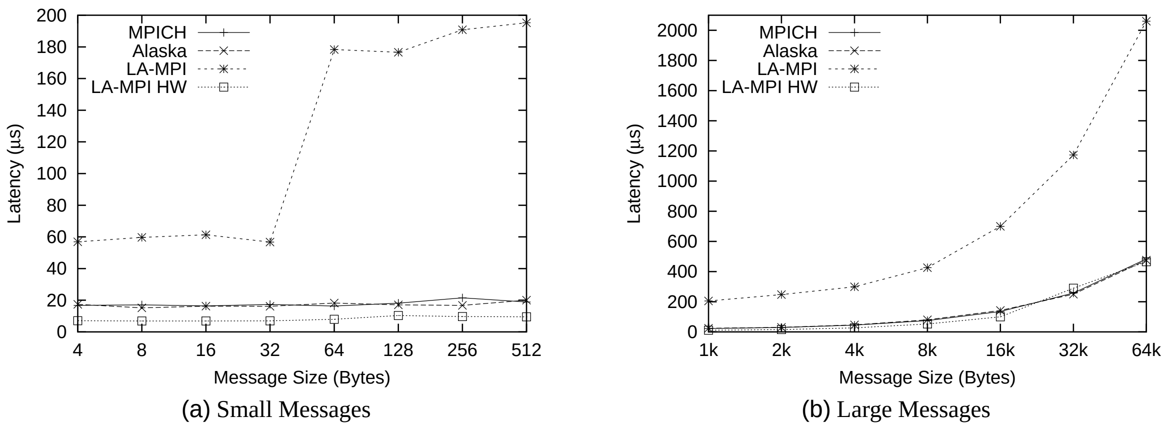

Figure 5. Performance Comparison of Different Broadcast Algorithms

6.2. Broadcast Scalability

Scalability is an important feature of any collective operation. To evaluate the scalability of the broadcast operation with different system sizes, we performed the same test with varying number of processes. Fig. 6 shows the broadcast latency for an 8-byte message over different number of processes. Compared to the original generic algorithm in LA-MPI, the new broadcast algorithm scales better. When compared to MPICH or Alaska MPI, the similar scalability is also observed.

6.3. Impact of the Number of Broadcast Channels

As discussed in Subsection 4.3, the synchronization needed to maintain the consistency of the global buffers can be delayed with a large set of global buffers. With this approach, the synchronization cost can be amortized over a set of broadcast operations.

To gain insights into the performance of the broadcast algorithm, it is beneficial to find out the impact of providing different number of broadcast channels. We measured the broadcast latency of the broadcast operation with different number of channels over 8 processes on an 8-node system. As shown in Fig. 7, with two channels, the broadcast latency is 17.1μs for 8-byte messages. The latency is reduced when more channels are provided. When sixteen channels are provided, this latency is reduced to 4.3μs. Thus the synchronization cost is about 13μs. By using 16 channels, it is indeed amortized into multiple operations.

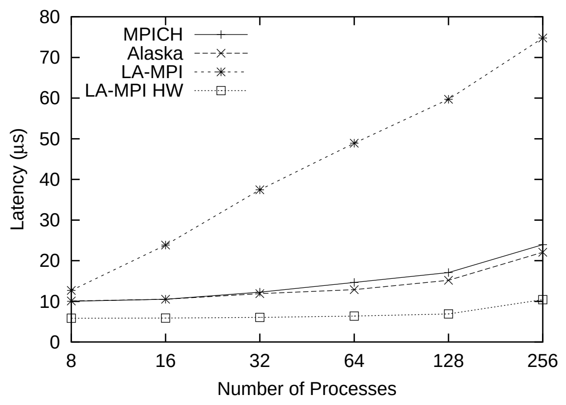

Figure 6. Broadcast Scalability with Different System Sizes

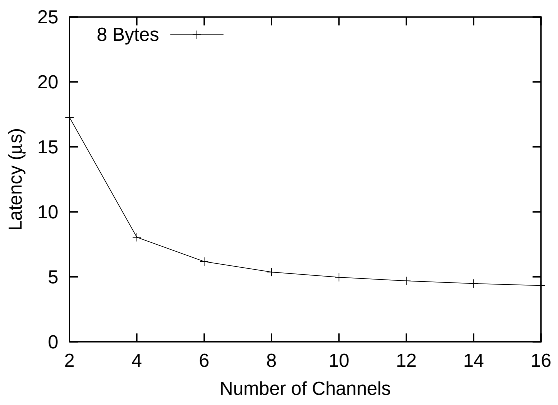

Figure 7. Impact of the Number of Broadcast Channels

6.4. Cost of Reliability

While the QSW [13] implementation for MPICH [7] and HP's for Alaska MPI leaves the issue of reliable data delivery to the Quadrics hardware, LA-MPI ensures end-to-end fault-tolerant message passing. This broadcast algorithm also allows the option to turn on the reliability. Under the reliable running mode, broadcast messages are retransmitted if any error occurs. The overhead of reliability can be measured when LA-MPI is running under the reliable mode. As shown in Fig. 8(a) the reliability cost is about 1μs for messages $\leq$ 256 bytes. This indicates that the added reliability has a little impact to the overall broadcast performance. But it has a significant impact to large message broadcast operations, as shown in Fig. 8(b). This is to be expected because the generation and validation of CRC/checksum for large messages takes a significant amount of time.

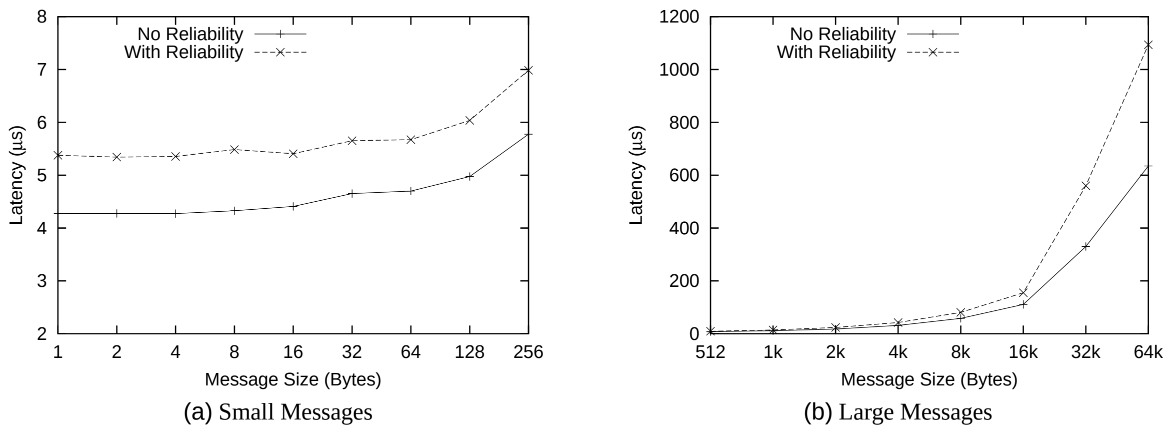

Figure 8. Cost of Reliability

7. Conclusions and Future Work

LA-MPI [6] is a high performance thread-safe implementation of the Message Passing Interface (MPI) library, designed for fault-tolerant message passing over terascale clusters. Currently, LA-MPI broadcast implementation is a generic spanning-tree based algorithm on top of point-to-point messaging, which does not exploit the benefits of Quadrics hardware broadcast communication.

In this paper, we take on the challenge of incorporating Quadrics hardware broadcast communication into LA-MPI to provide an efficient broadcast operation. We then describe the benefits and limitations of the hardware broadcast communication and possible strategies to overcome them. Accordingly, a broadcast algorithm is designed and implemented with the best suitable strategies. Our evaluation shows that the new broadcast algorithm achieves significant performance benefits compared to the original generic broadcast algorithm in LA-MPI. It is also highly scalable as the system size increases. Moreover, it outperforms the broadcast algorithms implemented by QSW [13] MPICH, and HP's for Alaska MPI. Furthermore, instead of leaving the reliability of message passing to the Quadrics hardware, this new algorithm can ensure the end-to-end reliable data delivery.

Although this broadcast operation is designed to be fault-tolerant over an arbitrary set of processes, the current evaluation only focuses on the basic performance in terms latency and scalability. In future, we intend to study its fault-tolerant capabilities in the presence of various errors. Also since QsNet provides programmable network interface support, we intend to offload some of the broadcast functionalities to the NIC, e.g., the synchronization phases, further improving its performance while maintaining its end-to-end message passing fault-tolerance. In addition, since the synchronization cost is considerably high, we intend to investigate possible ways to reduce this cost. Moreover, we plan to extend this broadcast algorithm so that it can take advantage of multiple Quadrics network interfaces and stripe messages over them. Furthermore, we intend to use the broadcast algorithm presented in this paper to support other collective operations in LA-MPI, such as scatter, barrier and allreduce.

References

[1] R. T. Aulwes, D. J. Daniel, N. N. Desai, R. L. Graham, L. Risinger, M. W. Sukalski, and M. A. Taylor. Network Fault Tolerance in LA-MPI. In *Proceedings of EuroPVM/MPI03*, September 2003.

[2] A. Bar-Noy and S. Kipnis. Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 13–22, 1992.

[3] Bhoedjang, Ruhl, and Bal. Efficient Multicast on Myrinet using Link-Level Flow Control. In *ICPP: 27th International Conference on Parallel Processing*, 1998.

[4] S. Coll, J. Duato, F. Petrini, and F. J. Mora. Scalable Hardware-Based Multicast Trees. In *Proceedings of Supercomputing '03*, November 2003.

[5] E. Frachtenberg, F. Petrini, J. Fernandez, S. Pakin, and S. Coll. STORM: Lightning-Fast Resource Management. In *Proceedings of the Supercomputing '02*, Baltimore, MD, November 2002.

[6] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. Minnich, C. E. Rasmussen, L. Dean Risinger, and M. W. Sukalski. A Network-Failure-tolerant Message-Passing system for Terascale Clusters. In *Proceedings of the 2002 International Conference on Supercomputing*, June 2002.

[7] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[8] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. P. Kini, P. Wyckoff, and D. K. Panda. Micro-Benchmark Level Performance Comparison of High-Speed Cluster Interconnects. In *Proceedings of Hot Interconnects 10*, August 2003.

[9] MPI Forum. MPI: A Message Passing Interface. In *Proceedings of Supercomputing '93*, November 1993.

[10] Myricom. Myrinet Software and Customer Support. http://www.myri.com/scs/GM/doc/, 2003.

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

[12] F. Petrini, S. Coll, E. Frachtenberg, and A. Hoisie. Hardware- and Software-Based Collective Communication on the Quadrics Network. In *IEEE International Symposium on Network Computing and Applications 2001, (NCA 2001)*, Boston, MA, February 2002.

[13] Quadrics Supercomputers World, Ltd. Quadrics Documentation Collection. http://www.quadrics.com/onlinedocs/Linux/html/index.html.

[14] W. Yu, D. Buntinas, and D. K. Panda. High Performance and Reliable NIC-Based Multicast over Myrinet/GM-2. In *Proceedings of the International Conference on Parallel Processing (ICPP'03)*, October 2003.