HIGH PERFORMANCE BROADCAST SUPPORT IN LA-MPI OVER QUADRICS

Weikuan Yu[1]
Sayantan Sur[1]
Dhabaleswar K. Panda[1]
Rob T. Aulwes[2]
Rich L. Graham[2]

Abstract

LA-MPI is a unique MPI implementation that provides end-to-end reliable message passing between application processes. LA-MPI collective operations are implemented on top of its point-to-point operations, using generic spanning tree-based collective algorithms. The performance of the collective operations scales in a logarithmic order over that of the point-to-point operations. Thus, it is desirable to provide more efficient and more scalable collective operations while maintaining the end-to-end reliability. To this end, we investigate the feasibility of utilizing Quadrics hardware broadcast in this paper. We explore several challenging issues such as broadcast buffer management, broadcast over arbitrary processes, retransmission and reliability. Accordingly, a low-latency, highly scalable, fault-tolerant broadcast algorithm is designed and implemented over Quadrics hardware broadcast. 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 the performance of our implementation is comparable to that of the high performance implementation by Quadrics Supercomputers World for MPICH, and HPs for Alaska MPI, while providing fault tolerance to network errors not provided by these.

Key words: Quadrics, Broadcast, LA-MPI

The International Journal of High Performance Computing Applications, Volume 19, No. 4, Winter 2005, pp. 453–463
DOI: 10.1177/1094342005056145

1 Introduction

LA-MPI (Graham et al., 2002) is a high performance thread-safe implementation of the message passing interface (MPI; MPI Forum, 1993). 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 (Aulwes et al., 2003). 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 (see http://web1.quadrics.com/onlinedocs/Linux/Eagle/html/), Myrinet GM (see http://www.myri.com/scs/GM-2/doc/html/), and shared memory.

The current LA-MPI implementation of collective operations is based on generic spanning tree-based algorithms over its point-to-point operations. Fault tolerance of the collective operations is achieved through the underlying point-to-point messaging layer. However, this direct layering of collective operawtions over point-to-point operations leads to a typical scalability problem. That is to say, the performance of collective operations scales in a logarithmic order over that of point-to-point operations. On the other hand, a high performance MPI implementation requires that attention be paid to the performance of collective operations.

Broadcast is one of the most important collective operations, which can also be used to implement other collective operations, such as allreduce, barrier, and allgather. Figure 1 shows the performance of LA-MPI broadcast operation over an eight-node Quadrics cluster, compared with two other MPI implementations available over Quadrics interconnect, MPICH-Quadrics and HPs Alaska MPI. The LA-MPI broadcast implementation does not perform as well as that of MPICH-Quadrics or HPs Alaska MPI. It also has a logarithmic scalability with respect to the number of processes, compared to the $O(1)$ scalability of the other two.

Both MPICH-Quadrics and HPs Alaska MPI provide high performance broadcast operations. As a matter of fact, MPI broadcast operation in both of them is implemented on top of the broadcast operation available in a communication library, `libelan`, released by Quadrics. However, compared to the raw performance of Quadrics hardware broadcast, `libelan` broadcast operation introduces more than 150% overhead. Figure 2 shows the performance comparison between the two. It remains to be

[1]NETWORK-BASED COMPUTING LABORATORY DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING OHIO STATE UNIVERSITY, COLUMBUS, OH 43210, USA (YUW@CSE.OHIO-STATE.EDU)

[2]LOS ALAMOS NATIONAL LABORATORY ADVANCED COMPUTING LABORATORY LOS ALAMOS, NM 87545, USA

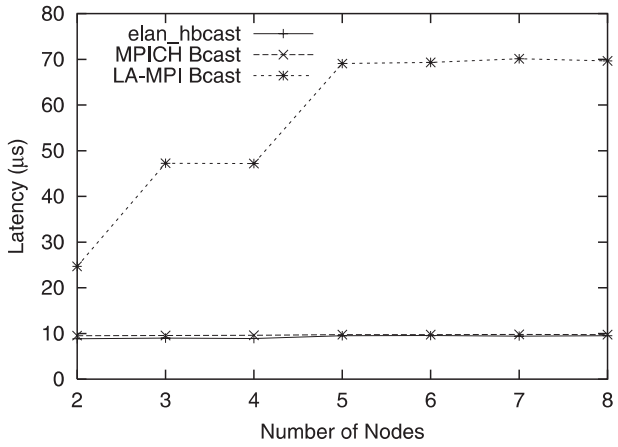


Fig. 1 Broadcast performance in different MPI implementations.

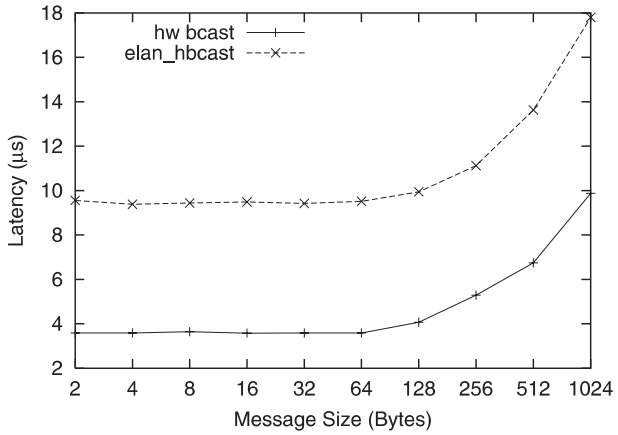


Fig. 2 Performance comparison of `libelan` broadcast and hardware broadcast.

investigated how a broadcast algorithm can be designed over Quadrics hardware broadcast without incurring as much performance overhead. Furthermore, MPICH-Quadrics and Alaska ignore the issue of end-to-end reliable data delivery, which is left to Quadrics hardware. It is desirable if a new broadcast algorithm can attain LA-MPI end-to-end reliability while leveraging the performance advantages of Quadrics hardware broadcast.

In this paper, we take on the challenge to design a low-latency, highly scalable, fault-tolerant broadcast algorithm, and explore the associated design issues such as broadcast buffer management, broadcast over arbitrary processes, retransmission and reliability. By overcoming these challenges, a broadcast algorithm is designed and implemented over Quadrics hardware broadcast. Our evaluation shows that this new algorithm achieves a latency of 8.9 µs when bwroadcasting a four-byte message over 256 processes on a 64-node system.

The rest of the paper is organized as follows. We provide an overview of Quadrics and its hardware broadcast in Section 2 and an overview of LA-MPI in Section 3. In Section 4, we describe the challenges in designing a new LA-MPI broadcast algorithm over Quadrics hardware broadcast. In Sections 5 and 6, we describe the implementation and the performance evaluation of this algorithm. Finally, we conclude this paper and discuss future work in Section 7.

2 Overview of Quadrics

Quadrics interconnect (see http://www.quadrics.com/onlinedocs/Linux/html/index.html) provides low-latency, high-bandwidth communication with its two basic building blocks: the programmable Elan3 network interface card and the Elite switch, which are interconnected in a fat-tree topology. Quadrics provides basic programming libraries, `libelan` and `libelan3`, and an MPI implementation MPICH-Quadrics over Quadrics. The Elan3 programming library (`libelan3`) is the lowest-level programming library.

2.1 QUADRICS COMMUNICATION AND HARDWARE BROADCAST

A parallel program over Quadrics consists of processes with different virtual process identifiers (VPIDs). Interprocess communication is supported by an efficient model, remote direct memory access (RDMA). An efficient, reliable and scalable hardware broadcast is also supported over Quadrics. A sender process performs this by addressing a set of processes with a single preallocated VPID. As shown in Figure 3, a hardware broadcast packet takes a predetermined path to reach all the recipients. It is successfully delivered when all the recipients return an acknowledgment. The top Elite switch in the path takes care of broadcasting the packets to and combining the acknowledgments from the recipients (Petrini et al., 2001). However, Quadrics hardware broadcast has its own restrictions. The destination addresses in a hardware broadcast have to be the same across all the processes. In addition, the processes being addressed must have contiguous VPIDs in order for the switch to perform the broadcast. Recently, the second generation of Quadrics interconnect ($QsNet^{II}$; Beecroft et al., 2003) has removed this restriction. This makes the utilization of hardware broadcast relatively easier. However, the main objective of this work is how to provide efficient end-to-

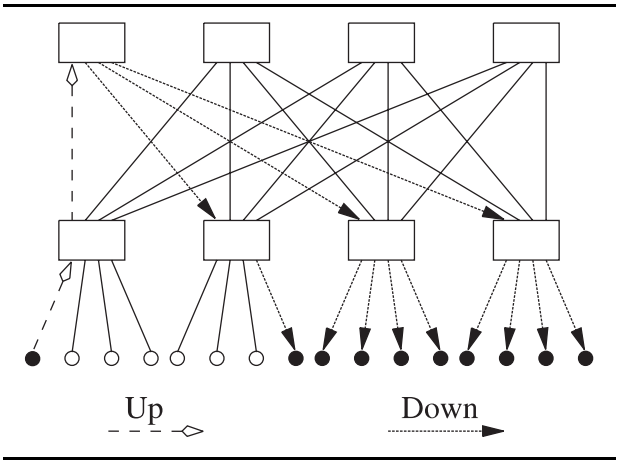


Fig. 3 Hardware broadcast.

end reliability support over Quadrics hardware broadcast. In this context, our work can be applied to QsNet^{II} hardware broadcast as well.

2.2 BROADCAST IMPLEMENTATION IN LIBELAN

A broadcast operation in a parallel program can involve arbitrary processes, which may not have contiguous VPIDs. The destination address in each process may not be identical, which is required for Quadrics hardware broadcast. The Quadrics `libelan` library overcomes these limitations inside an `elan_bcast()` function. Two broadcast buffers are provided for double buffering of messages so that one broadcast operation can start while another one is still in transit. In addition, `elan_bcast()` imposes a synchronization before the start of every broadcast operation. This is to synchronize the completion of an early message so that it is not overwritten by an incoming broadcast message. A side effect is that consecutive broadcast operations are throttled by this inserted synchronization, which can lead to a low broadcast throughput. Although it is necessary to have the synchronization, it would be beneficial if its frequency can be reduced. LA-MPI is developed from the lowest `libelan3` programming library and provides a different communication protocol from `libelan`. The `libelan` collective support is also not end-to-end reliable. Thus it cannot be used directly by LA-MPI for its collective communication.

3 Overview of LA-MPI

In this section, we give some background information about LA-MPI. As shown in Figure 4, the implementation of LA-MPI has its MPI interface layered upon the user-level messaging (ULM) interface, which itself consists of two layers: the memory and message layer (MML) and the send and receive layer (SRL; Graham et al., 2002). 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 responsible for sending and receiving message fragments through different network adapters and shared memory, and is highly network dependent.

3.1 MEMORY AND MESSAGE LAYER

The MML is a set of abstractions to ensure reliable message delivery while minimizing the overhead of memory management for LA-MPI data structures and buffers (Graham et al., 2002). 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 the memory on the network interface card (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 NICs, and within a path there may be several independent routes. The path scheduler binds a specific message to a particular path. Different messages between the same end-points may use different paths. Together all three components of the MML architecture provide support to MPI functionalities with network transmission-specific primitives (i.e. the SRL).

3.2 SEND AND RECEIVE LAYER

The SRL is responsible for the sending and receiving of messages. It consists of multiple network path implementations and a highly optimized implementation of shared memory communication. 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 paths, where the message fragments are sent via associated physical resources. In addition, this layer handles the in-order delivery required by the MPI standard. Messages that arrive out-of-order are queued for later processing as unexpected data.

4 Design Challenges

There are several challenges in order to design MPI broadcast over Quadrics hardware broadcast. These include broadcast buffer management, broadcast to arbitrary proc-

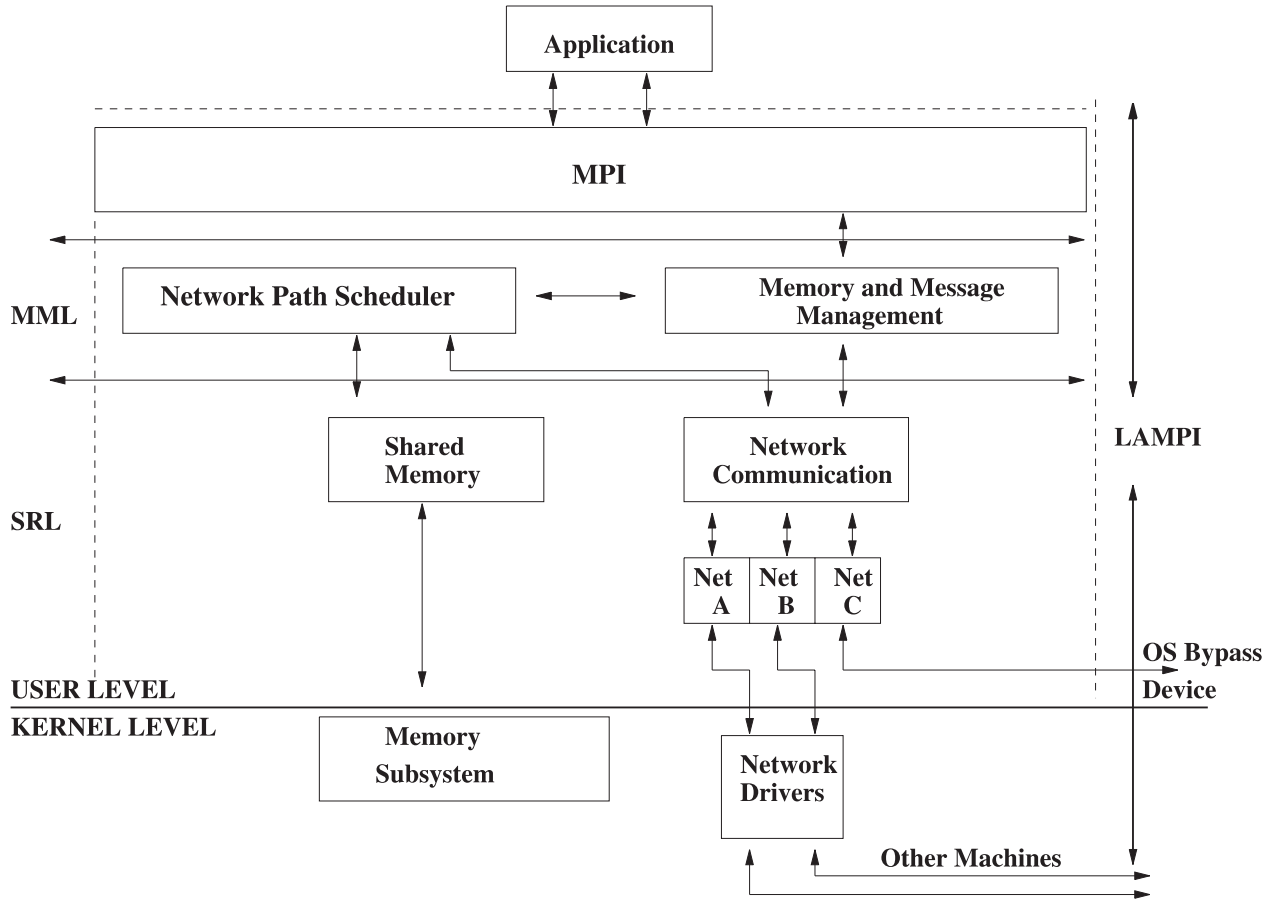


Fig. 4 LA-MPI architecture.

esses, acknowledgment and synchronization, retransmission and reliability. In this section, we explore each of these issues and discuss several strategies to overcome them.

4.1 BROADCAST BUFFER MANAGEMENT

Quadrics hardware broadcast can only address a destination address that is global across all receiving processes. This demands the presence of a global address space. The global address space is created by having each process map a part of its virtual main memory to an identical address space on the Elan3 network interface. The use of this global address space has to be consistent among all processes. Two alternatives can be employed to maintain the consistency. One is to use a global allocator to enforce the consistency and have every memory request go through it. The other is to divide the global memory into an array of small buffers and use different buffers for broadcast operations sequentially. The first alternative essentially allocates broadcast buffer on-the-fly and has to invoke the global allocator in the communication critical path. It would be better to choose the second alternative to avoid the penalty on going through the global buffer allocator. It is also easier to manage the global buffer with them divided into a small number of buffers, referred to as broadcast channels hereafter. The global memory consistency is discussed more later in Section 4.3.

4.2 BROADCAST TO ARBITRARY 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. There are several design alternatives for broadcasting over arbitrary processes. One alternative is to partition the processes into disjoint subsets of contiguous processes and perform a broadcast operation to each set. This allows the utilization of the existing hardware broadcast 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 using a tree-based algorithm to perform the broadcast with each node being a set of processes with contiguous VPIDs. After the processes in one node receiving the data, they can initiate parallel broadcast operations to processes in the remaining nodes. For fast expansion on 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. (2003). Compared to the first alternative, the second alternative with a tree topology and the associated data structure is rather complex. 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 inside MPI_COMM_WORLD). Moreover, the use of the base group by collective operations is very common in MPI programs. Thus, the first alternative can be sufficient for the purpose of exploiting the advantages of Quadrics hardware broadcast.

4.3 ACKNOWLEDGMENT AND SYNCHRONIZATION

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

One alternative is to use a tree-based algorithm to collect all the acknowledgments to one process, and then update others with the combined results. Another is to use a special form of Quadrics network transactions, test-and-set, to poll on a global value (Frachtenberg et al., 2002; see http://www.quadrics.com/onlinedocs/Linux/html/index.html). This transaction tests the value of a global variable across all processes against a provided value, and then, if they are the same, writes a new value into the same global address (Petrini et al., 2001). Quadrics hardware broadcast is used for polling. Synchronization is achieved by having all processes waiting for a new value to be available. This is very efficient when all the processes reach the synchronization fairly close to each other, otherwise the polling broadcast traffic is rather intrusive to the network. Thus, an exponential back-off scheme needs to be in place to avoid too much broadcast traffic incurred by the test-and-set transaction (Petrini et al., 2001; see http://www.quadrics.com/onlinedocs/Linux/html/index.html). In addition, the timing of the synchronization is also important. The synchronization can be delayed if there are still broadcast channels 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 broadcast channels 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 across the PCI bus or the network (Graham et al., 2002). Errors can be propagated and the effect of these errors can be amplified in long-running parallel programs, especially over terascale clusters because of the sheer number of their components. To achieve end-to-end reliable message passing, LA-MPI optionally supports sender-side retransmission when the messages have exceeded their timeout periods. 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 so desired). This checksum/CRC protects the message against network and I/O bus corruption. When this 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 return their positive acknowledgment (ACKs), otherwise negative acknowledgments (NACKs) are returned. The ACKs and NACKs are combined during the synchronization and the corresponding sender processes are notified after the synchronization. A time-stamp and the number of retransmission are also recorded 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.

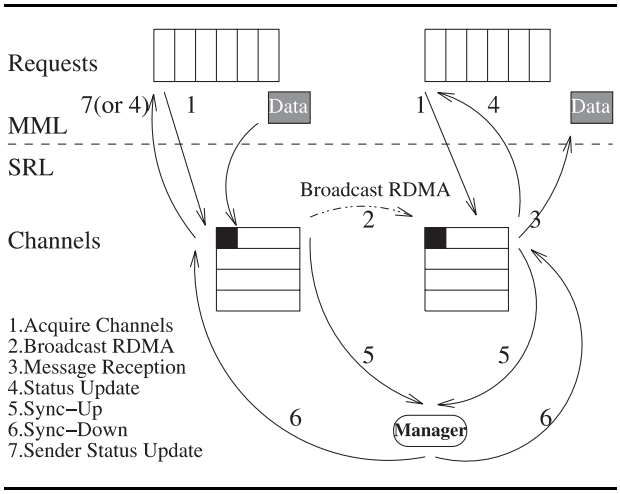


Fig. 5 Broadcast flow path using Quadrics hardware broadcast.

When the hardware broadcast is no longer available or its failure is suspected, a fail-over function is provided to detect this and binds all the outstanding broadcast operations to the broadcast algorithm 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.

5 Our Implementation

As shown in Figure 5, we have implemented the broadcast operation with added support in both the MML and the SRL in LA-MPI. In this section, we provide the details on the broadcast flow path and its reliability.

5.1 BROADCAST FLOW PATH

The flow path of a broadcast operation can be divided into four major steps: request binding, send-side message transmission, receiver-side message delivery, and synchronization.

1. **Broadcast buffer management and request binding**

   To make use of Quadrics hardware broadcast, we choose the second alternative described in Section 4.1 to allocate global memory and divide it into a number of channels. As shown in Figure 5, a broadcast operation binds to one of the channels when they are available. This is shown as step 1 in the MML on both the sender and receiver sides.

2. **Sender-side message transmission**

   The sender generates CRC/checksum and creates a message header at the beginning of the channel. If the message is not addressable by the Elan3 network interface 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 16 KB and also elan addressable, it is transmitted as a separate message before the header. This is shown as step 2 in Figure 5. 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.

3. **Receiver-side message delivery**

   When a broadcast message arrives at the receiver side, its tag is matched against the corresponding broadcast request. If it is an expected message, the data are copied into the destination buffer and CRC/checksum is generated and compared to the CRC/checksum contained in the header. If the received data are not corrupted, the request status is updated as completed, as shown in Figure 5 (steps 3 and 4 at the receiver side). If the message is not expected or its data are 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, it also updates the MML request status as completed, shown in Figure 5, step 4 at the sender side. Otherwise, it has to wait until it is notified that all receivers have received the data through synchronization.

4. **Synchronization**

   As discussed in Section 4.3, synchronization is needed before reusing a broadcast channel. We choose to take a tree-based algorithm for the synchronization, and leave the test-and-set based approach 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 Figure 6, we use a balanced tree to perform the `up`

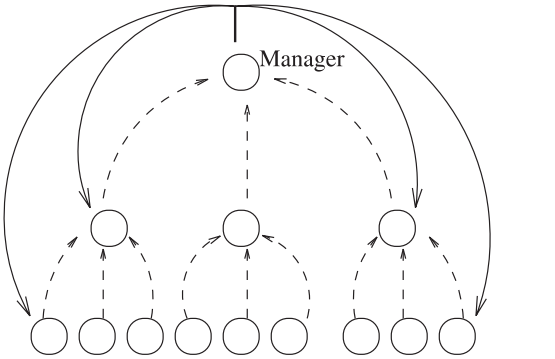


Fig. 6 Synchronization of the broadcast operation.

phase synchronization following the approach described in Petrini et al. (2001); see http://www.quadrics.com/onlinedocs/Linux/html/index.html. 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 Figure 5 as step 5. During the `down` phase of synchronization, the manager process broadcasts its result using the same broadcast approach described in Section 4.2. This is shown in Figure 5 as step 6. The two-phase synchronization is also shown in Figure 6 in more detail. With successful synchronization, each process updates its channel state and progresses to the next round of broadcast operations and synchronization.

5.2 BROADCAST RELIABILITY

When a NACK is received, the manager process notifies all the processes that there is a transmission error since the last synchronization. A process then retransmits all messages for which it is the sender since the last synchronization. A receiving process that has received data correctly will ignore the redundant message. However, it still cannot reuse the channel because the synchronization stage has not finished updating its channel state and releasing the channel. This approach implies that the retransmission is time-consuming and will possibly repeat some successfully completed broadcast operations. Taking into account the high reliability of Quadrics interconnect, this is acceptable because it can speed up the common cases of successful transmission by freeing processes from frequent synchronization. Since the error rate is low, the cost for the rare retransmission cases can be contained by the benefits from saved synchronization. When a message happens to be lost, the sender (or the receiver) detects it with its time-stamp, or the manager can detect it when it fails to make progress through synchronization. In either case, the broadcast retransmission is triggered. 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.

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 Quadrics for MPICH (Gropp et al., 1996), and HPs for Alaska MPI. For the experiments in Sections 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.

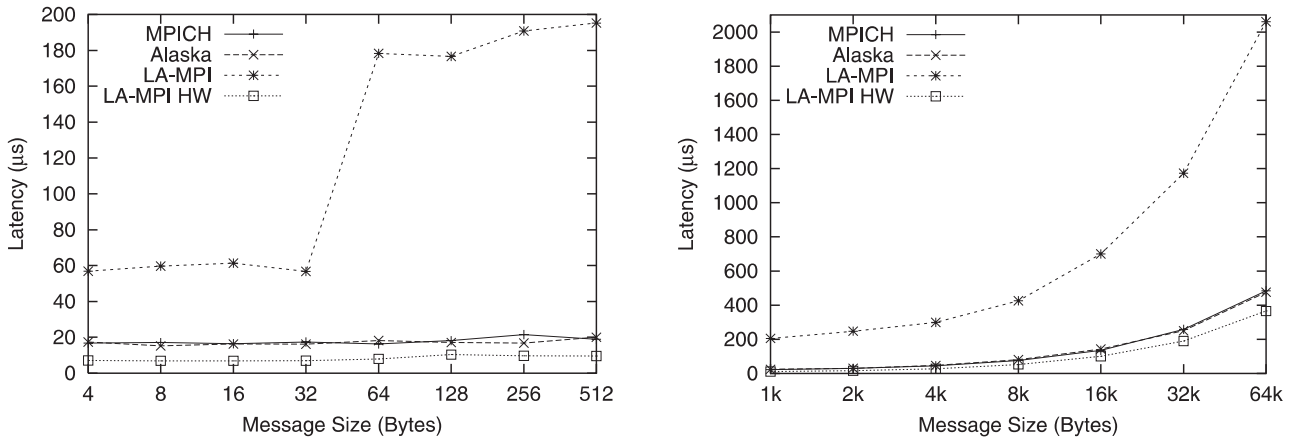
Our tests are performed by having the processes first warmed up with 20 broadcast operations. Then a sample size of 1000 broadcast operations is performed after a barrier synchronization. This is repeated for 100 samples, each again performed after a barrier. The duration of each sample is recorded. 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-Quadrics and Alaska MPI broadcast operation, which uses the hardware broadcast communication.

6.1 BROADCAST LATENCY

We measured the broadcast latency of four-byte messages over 128 processes on 32 contiguous nodes. As shown in Figure 7, 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 Quadrics hardware broadcast. As also shown in Figure 7(a), our broadcast algorithm outperforms the algorithms implemented for MPICH by Quadrics and HPs for Alaska MPI. This indicates that the new algorithm is able to take advantage of the hardware broadcast with much less overhead compared to the broadcast algorithm provided in `libelan`. These performance gains are due to the pipelining of broadcast messages and the reduction on synchronization overhead.

6.2 BROADCAST SCALABILITY

Scalability is an important feature of any collective operation. To evaluate the scalability of the broadcast opera-



(a) Small Messages

(b) Large Messages

Fig. 7 Performance comparison of different broadcast algorithms.

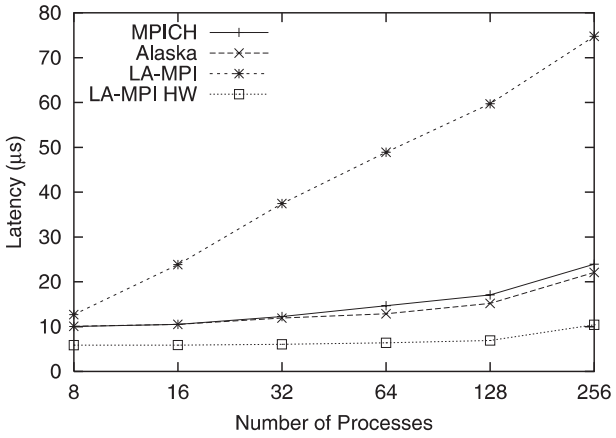


Fig. 8 Broadcast scalability with different system sizes.

tion with different system sizes, we have performed the same test with varying number of processes. Figure 8 shows the broadcast latency for an eight-byte message over different number of processes. Compared to the original generic algorithm in LA-MPI, the new broadcast algorithm achieves a scalability close to $O(1)$, up to 128 processes. When compared to MPICH or Alaska MPI, a better scalability is also observed. This indicates that the new broadcast algorithm is also able to provide a better scalability while taking performance advantage of Quadrics hardware broadcast.

6.3 COST OF RELIABILITY

While MPICH-Quadrics and HPs for Alaska MPI leaves the issue of reliable data delivery to Quadrics hardware, LA-MPI ensures end-to-end reliable message passing. LA-MPI provides an option to turn the end-to-end reliability on or off. The users have the convenience to choose between high reliability and fast performance. Under the reliable running mode, broadcast messages are retransmitted if any error occurs. The cost of reliability can be measured by comparing the performance of LA-MPI under the reliable and unreliable modes. As shown in Figure 9(a) the reliability cost is about 1 µs for messages ≤ 256 bytes. This indicates that the added reliability has a little impact on the overall broadcast performance. However, it has a significant impact on large message broadcast operations, as shown in Figure 9(b). This is to be expected because the generation and validation of CRC/checksum for large messages takes a significant amount of time.

6.4 IMPACT OF THE NUMBER OF BROADCAST CHANNELS

As discussed in Section 4.3, synchronization is needed to maintain the consistency of broadcast channels. With the provision of multiple broadcast channels, the synchroni-

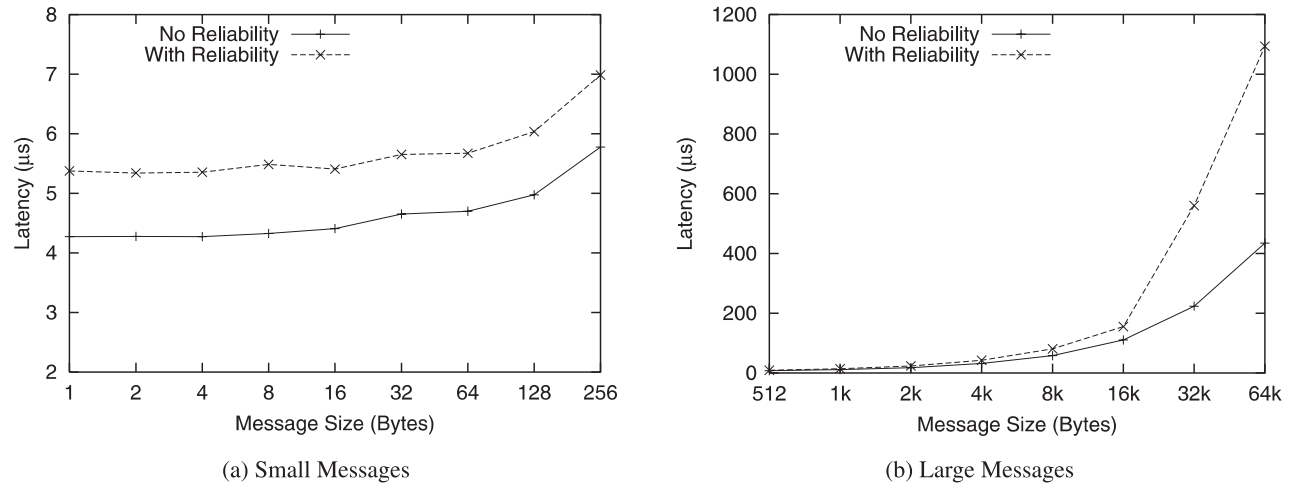


Fig. 9 Cost of reliability.

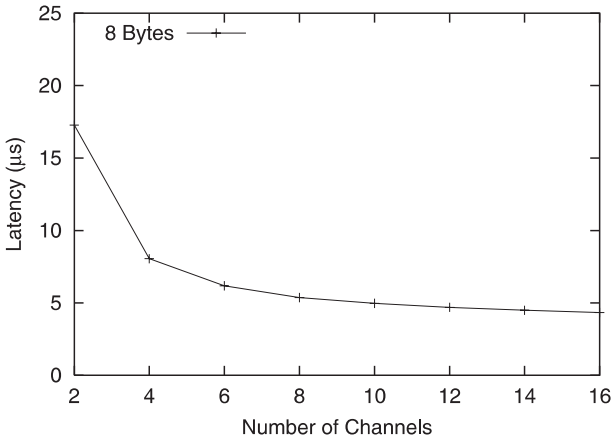


Fig. 10 Impact of the number of broadcast channels.

zation 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 a different number of channels over eight processes on an eight-node system. As shown in Figure 10, with two channels the broadcast latency is 17.1 µs for eight-byte messages. The latency is reduced when more channels are provided. When 16 channels are provided, this latency is reduced to 4.3 µs. Thus, the synchronization cost is reduced by 13 µs as the frequency of synchronization is dropped to every 16 broadcast operations. By buffering multiple messages with 16 channels, the cost is indeed amortized into multiple operations.

7 Conclusions and Future Work

LA-MPI (Graham et al., 2002) is a high performance thread-safe implementation of the 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.

In this paper, we take on the challenge to design an efficient LA-MPI broadcast operation over Quadrics hardware broadcast. We describe the benefits and limitations of Quadrics hardware broadcast and possible strategies to overcome them. Accordingly, a broadcast algorithm is designed and implemented with the best suitable strategies to achieve high performance, as well as end-to-end reliability. Our evaluation shows that the new broadcast algorithm achieves significant performance gains compared to the original generic broadcast algorithm in LA-MPI. It is also highly scalable as the system size increases. It outperforms the broadcast algorithms implemented by Quad-

rics Supercomputers World MPICH and HPs for Alaska MPI.

The current evaluation 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. We also plan to extend this broadcast algorithm so that it can take advantage of multiple Quadrics network interfaces and stripe messages across them. Moreover, since Quadrics 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.

ACKNOWLEDGMENTS

This research is supported in part by Grant 76521-001-03 4v 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.

AUTHOR BIOGRAPHIES

Weikuan Yu is a Ph.D. student in the Computer Science and Engineering Department at the Ohio State Univeristy. His research interests include parallel computing, parallel file systems, and bioinformatics. Yu has an M.S. in molecular, cellular and developmental biology from Ohio State University. He is a member of the IEEE Computer Society and the USENIX Association.

Sayantan Sur has been a Graduate Researcher at the Department of Computer Science and Engineering at the Ohio State University for the last two years. His research has concentrated on collective communication issues in high performance clusters of commodity processors. Prior to this, he was employed as Member of Technical Staff at Sun Microsystems, India. He obtained a Bachelors of Technology degree in electrical and electronic engineering from the National Institute of Technologyin Calicut India in 2001.

Dhabaleswar K. Panda is a Professor of Computer Science at the Ohio State University. He obtained his Ph.D. in computer engineering from the University of Southern California. His research interests include parallel computer architecture, high performance computing, user-level communication protocols, interprocessor communication and synchronization, network-based computing, and quality of service. He has published over 125 papers in major journals and international conferences related to these research areas. Dr. Panda has served on the Program Committees and Organizing Committees of several parallel processing and high performance computing conferences and on the editorial boards for several parallel processing journals. Dr. Panda is a recipient of the National Science Foundation Faculty Early CAREER Development Award, the Lumley Research Award (1997 and 2001) at the Ohio State University, and an Ameritech Faculty Fellow Award. He is a member of IEEE Computer Society and ACM.

Rob Aulwes has been employed at Los Alamos National Laboratory for over one year, where he is currently a Technical Staff Member. He is working with the Unified Data Model team whose goal is to provide an efficient parallel I/O library to support the manipulation of mesh data and data interchange between applications at the lab. He obtained his Ph.D. in applied mathematics from the University of Iowa in 1999 under the direction of Dr. William Klink and Dr. Tuong Ton-That.

Richard Graham has been a technical staff member in the Advanced Computing Laboratory (ACL) at Los Alamos National Laboratory since 1999, and is the Resilient Technologies team leader. Rich graduated from the Seattle Pacific University in 1983 with a Bachelors degree in chemistry and received a Ph.D. in theoretical chemistry from Texas A&M in 1990. Rich worked as a postdoctoral researcher at the University Of Chicago from 1990 to 1992, and worked at Cray/SGI for seven years before joining the ACL. Richs interests are primarily in developing low-level supporting technologies for high performance large-scale simulations. He has been a member of the team developing LA-MPI since the projects inception in 1999, and has led the project for the past four years, bringing it along from being a research project on high performance fault-tolerant communications, to its current stage of a production quality high performance, fault-tolerant MPI.

References

Aulwes, R. T., Daniel, D. J., Desai, N. N., Graham, R. L., Risinger, L., Sukalski, M. W., and Taylor, M. A. 2003. Network fault tolerance in LA-MPI. *Proceedings of EuroPVM/MPI03*, Venice, Italy, September 29–October 2.

Beecroft, J., Addison, D., Petrini, F., and McLaren, M. 2003. QsNet-II: an interconnect for supercomputing applications. *Proceedings of Hot Chips 03*, Stanford, CA, August.

Coll, S., Duato, J., Petrini, F., and Mora, F. J. 2003. Scalable hardware-based multicast trees. *Proceedings of Supercomputing 03*, Phoenix, AZ, November 15–21.

Frachtenberg, E., Petrini, F., Fernandez, J., Pakin, S., and Coll, S. 2002. STORM: lightning-fast resource management. *Proceedings of Supercomputing 02*, Baltimore, MD, November 16–22.

Graham, R. L., Choi, S-E., Daniel, D. J., Desai, N. N., Minnich, R., Rasmussen, C. E., Risinger, L. D., and Sukalski, M. W. 2002. A network-failure-tolerant message-passing system for terascale clusters. *Proceedings of the 2002 International Conference on Supercomputing*, New York, NY, June 22–26, pp. 77–83.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A. 1996. A high-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing* 22(6):789–828.

MPI Forum. 1993. MPI: a message passing interface. *Proceedings of Supercomputing 93*, Portland, OR, November 15–19.

Petrini, F., Coll, S., Frachtenberg, E., and Hoisie, A. 2001. Hardware- and software-based collective communication on the Quadrics network. *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA 2001)*, Boston, MA, February.