Efficient Reliable Broadcast for Commodity Clusters*

Kwan–Po Wong and Cho–Li Wang
Department of Computer Science and Information Systems
The University of Hong Kong
Pokfulam, Hong Kong
kp.wong@graduate.hku.hk, clwang@csis.hku.hk

Abstract

High–performance collective communications for commodity clusters have been recently addressed. In the past, broadcast operation is implemented by using unicast operations. With our novel architecture, EQA (Enhanced Queue Architecture), and the hardware broadcast facility in network switches, an efficient reliable broadcast operation is implemented. As the size of a cluster doubles, the latency of the broadcast operation was only increased by less than 16% while the latency of the MPI broadcast was increased by at least 45%. We found that EQA could efficiently utilize computation and memory resources. It could reduce the memory consumption by almost 40%. To further test the performance of the system, we evaluated the performance of the broadcast operation. We found that Push–Pull Broadcast from DP–SMP with EQA outperforms the MPI broadcast operation. DP–SMP with EQA has proven to be a high–performance communication mechanism for commodity SMP clusters.

Keywords: Commodity Cluster, SMP, Communication Protocol, Bandwidth, Buffer Management, Low–Latency Communication.

1. Introduction

Commodity clusters had proven to be a viable solution to provide high computing power. To achieve such high performance, the cluster communication played a critical role. High–performance communication for commodity clusters has been addressed recently. Most research studies are focused on the performance of the point–to–point communication [1,3,4,5,8,10,11,13,17,18]. Although the current studies can make the communication performance between two cluster nodes almost comparable to the raw network performance, the optimization of the performance of the collective communication has not been addressed properly.

Collective communication plays a very important role in many parallel applications. The most fundamental collective operation is the broadcast operation. This operation can be used to build other important collective operations like barrier synchronization, which synchronizes all cluster nodes. The performance of broadcast operations can determine the performance of parallel applications.

Theoretical broadcast studies have focused on the delivery strategy of packets [2,7,9,14]. These studies have all assumed that the underlying implementation of the algorithm uses point–to–point communication. In fact, the hardware broadcast feature, which is commonly available in network switches and hubs, can help improve the performance of collective communication. With the incorporation of hardware broadcast into the design of broadcast algorithm, computation, memory and network resources can possibly be utilized efficiently when broadcasting a message using hardware broadcast packets. Furthermore, if the size of a cluster is smaller than the size of a switch, injecting one single packet can broadcast to all cluster nodes. However, broadcast packets, like ordinary packets, are subjected to lose in communication. This may because of insufficient buffer entries at the outgoing ports of the switch or at the network interface cards (NICs).

To avoid losing packets, a reliable protocol should be adopted. In point–to–point communication, each channel maintains its reliability independent of other channels. Each channel may keep a set of data structures, used as transmission and reception buffers, for the purpose of reliable transmission. As the size of the cluster increases, the number of transmission and reception buffers in a node increases. The aggregated size of all data structures may consume a large portion of memory spaces. This becomes a vital problem.

We have developed DP–SMP [17] which supports Pentium–based SMP clusters. It used Push–Pull Messaging as the messaging mechanism for point–to–point communication between SMP nodes. This mechanism focused on the scheduling of the data transfer between cluster nodes. To incorporate the facility in DP–SMP while maintaining the reliable delivery feature for user-level DP–SMP applications, a novel reliable mechanism, Enhanced Queue Architecture (EQA), is introduced. This mechanism has been implemented along with Push–Pull Messaging in our DP–SMP system. The mechanism leverages the memory consumption problem

*The research was supported by Hong Kong Research Grants Council grant 10201701 and HKU CRGC grant 10200544.

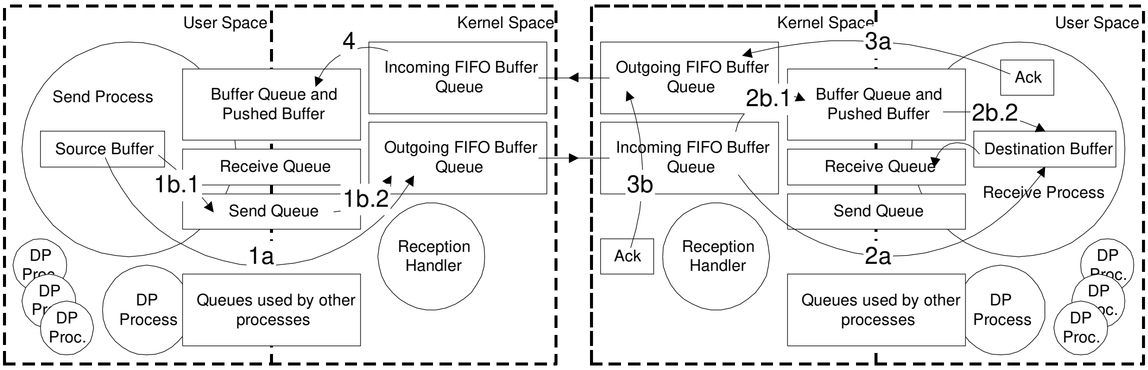

FIGURE 1. DP−SMP architecture.

in communication buffers as the size of the cluster in−creases. It also enables the incorporation of hardware broadcast facility in the DP broadcast operation to im−prove the performance of broadcast communication, which would tightly couple the cluster nodes.

For the rest of the paper, we discuss Push−Pull Mes−saging in section 2. We then introduce the efficient reli−able broadcast algorithm using hardware broadcast ca−pability in section 3. In section 4, we study the perform−ance of the broadcast algorithm. Conclusion is given in section 5.

2. Push−Pull Messaging

Directed Point [10], as known as DP, is a high−speed communication subsystem originally developed by the Systems Research Group at the University of Hong Kong. It aimed at providing a low−latency and high−bandwidth communication system for cluster network−ing. It supports network driver modules for Digital DEC 21140A Fast Ethernet, Hamachi Gigabit Ethernet, and FORE PCA−200E ATM [18], for connecting commodity nodes.

Push−Pull Messaging is a messaging mechanism to enable high−speed communication in SMP clusters based on the DP concept [17]. The mechanism is to lower latency and increase bandwidth. Figure 1 illus−trates the communication architecture of DP−SMP. In DP−SMP, each DP contains a set of queues and buffers, which are used for data transmission and reception. For transmission, a queue called *send queue* stores pending send requests. Packets are stored in a buffer called *send buffer*. Similarly for reception, a queue called *receive queue* stores pending receive requests. Packets received from the NIC are stored in a buffer called *receive buffer*. The *buffer queue* and *pushed buffer* store pending in−coming packets where their destinations in memory are not determined yet. Three queues can be accessed by both user and kernel threads.

Push−Pull Messaging consists of two distinct push and pull phases in data communication. With dynami−cally changing phases during communication, messages can be transmitted efficiently. As shown in Figure 1, each send or receive process has its own application−al−located buffer, source buffer and destination buffer re−sided in the user space.

In Push−Pull Messaging, the send process first pushes a part of a message to the receive party as shown in ar−row 1a. The size of the first pushed message is deter−mined by the parameter BTP (Bytes−To−Push). This important parameter BTP defines the number of bytes to be pushed by the sender at the beginning of the messag−ing process. This number is chosen based on the speed balance of the network and the memory system [16]. BTP is further divided into two values BTP_1 and BTP_2 where $BTP = BTP_1 + BTP_2$. These two values can fur−ther finely tune the performance of the cluster.

The pushed message is then handled by a reception handler at the receive party. The rest is registered in the send queue through arrow 1b.1. Depending on the timing of the receive operation performed by the receive proc−ess, the pushed message will be stored in the pushed buffer if the receive operation is not yet started as shown in arrow 2b.1. Otherwise, the message will be copied to the destination as shown in arrow 2a by the registered information in the receive queue.

Once the receive operation is started, either the re−ception handler at the receive party or the receive proc−ess itself will start the pull phase to pull the rest of the message from the send process. The pull phase will be started by sending an acknowledgment (or "Ack" in the figure), which implicitly contains request information, through arrow 3a or arrow 3b. The reception handler in the send party processes the acknowledgment. If the re−quest is granted, the send handler will send the request part of the message in the send queue through arrow 1b.2 to the receive party. The reception handler in the receive party handles the message and directly copies the mes−sage from NIC to the destination buffer without buffer−ing in the pushed buffer through arrow 2a using the reg−istered information in the receive queue.

With the mechanism, Push−Pull Messaging guaran−tees buffers to be properly managed. It can also reduce the handshaking delay for short messages. In addition, the sequence of communication events makes it possible to apply various optimizing techniques to remove those unexpected overheads in the critical path in communica−tion. Optimization techniques include Address Transla−tion Overhead Masking, Cross−Space Zero Buffer and Push−and−Acknowledge Overlapping [16].

For point−to−point intranode communication, DP−SMP can achieve a very low single−trip latency of 7.5 µs for sending a 8−byte message. The peak bandwidth is

350.9 Mbytes/s when sending 4000 bytes. For point–to–point internode communication, the system can achieve a single–trip latency of 30.1 µs for sending a 8–byte message. The peak bandwidth is 12.1 Mbytes/s under 100–Mbits/s Ethernet connection.

3. Broadcast with EQA

Conceptually, a reliable broadcast operation is established by multiple pairs of reliable send and receive operations, which is originated from the same source endpoint to a set of destination endpoints. Each endpoint maintains its own communication channel independently. When a broadcast operation is issued, the broadcast message is transmitted across a number of the channels. With the traditional high–level implementation, a sequence of point–to–point communication will be used to broadcast a message. This approach simplifies the implementation of the broadcast operation but it cannot fully optimize for the performance.

Since the reliable channels are maintained independently, each channel may keep a set of data structures, used as transmission and reception buffers, for the purpose of reliable transmission. As the size of the cluster increases, the number of transmission and reception buffers in a node increases. The aggregated size of all data structures may consume a large portion of memory spaces. This becomes a vital problem.

To optimize the use of resources, we consider the broadcast operation as a single "fat" operation. In this low–level approach, broadcasting a packet can be done through one packet with an encoded hardware broadcast operation. This would minimize the use of memory and network resources. We further introduce a reliable delivery mechanism to guarantee that all packets are delivered from the source node to all other participating nodes. All these novel mechanisms are relied on Enhanced Queue Architecture (EQA).

Two different implementations of the broadcast operation in DP–SMP are discussed in this section. One is called Simple Broadcast. Another one is called Push–Pull Broadcast.

3.1. Enhanced Queue Architecture (EQA)

EQA is a queue architecture for DP–SMP to utilize memory and network resources efficiently. EQA allows multiple senders to share one single queue and buffer properly. An entry produced in a queue and buffer could be retrieved by many senders which linked to the queue and buffer. Memory resources can be efficiently utilized.

EQA provides a reference count feature to keep track on the use of the queue and buffer. It further provides medium–grained locking mechanism so that multiple DP processes could share resources safely.

In EQA, there are three basic data structures as shown in Figure 2. One is Buffer (BUF). The other one is Home Management Structure (HMS). The last one is Local Management Structure (LMS). BUF is a fundamental data structure, which stores all buffer entries. HMS is a data structure for producers to maintain the integrity of BUF. This structure is very similar to the producer in the ordinary SPSC (Single–Produced–Single–Consumer) queue except it also contains a pointer which points to the head of the list of consumers. LMS is a data structure for consumers to keep track BUF. This structure is also very similar to the data structure of the consumer in the ordinary SPSC queue. The structure also contains a pointer, which points to the next consumer structure. This pointer is used for the EQA update algorithm.

In EQA, a lightweight directed point (LDP) is a data structure, which is aimed to minimize the size of a DP. The data structure of a LDP is the same as the structure of a DP except LDPs have no buffer entries. LDPs only store the buffer management structure. Buffer entries are stored in the form of pointers, which point to appropriate destinations.

To create a queue, a DP–SMP process needs to take a 2–step procedure. The first step is to create a LDP. The second step is to create a link from the LDP to a DP. Once the link is created, EQA features will be enabled. The linked LDPs can retrieve packets created in the DPs. With this organization, broadcast packets generated at the DPs can be transmitted to a set of destination DPs through a set of LDPs. The way to access the LDP will be exactly the same as the way to access DPs.

To set up a broadcast environment, a set of send–only LDPs is created. Correspondingly, a set of DPs is created at each destination node. For example in Figure 2, all LDPs at NID1 share buffers with DP1.

To perform a broadcast operation, root and leaf nodes start different procedures. At the root node, the broadcast DP is created by defining a set of LDPs which point to a set of destination DPs. Thus, one single memory space is used as the retransmission buffer. Whenever a send operation is issued to the DP, a sequence of send operations will be issued at the same time virtually. Physically, this sequence of operations are implemented by one hardware broadcast operation. When the DP issues the send operation, the operation is wrapped and the hardware broadcast operation is triggered instead.

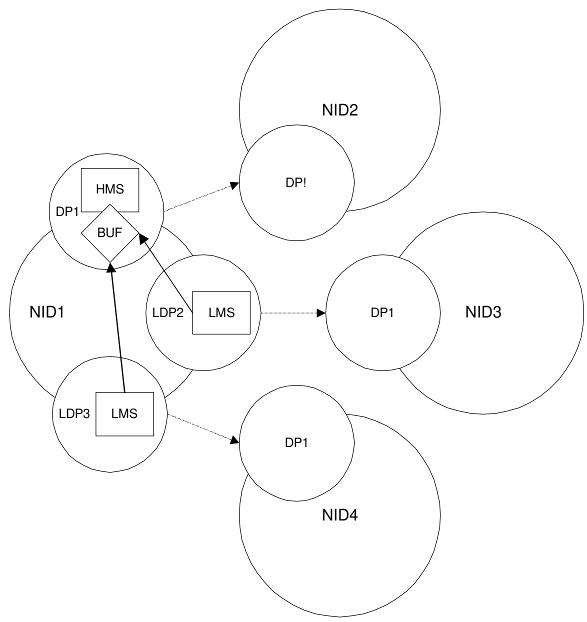

FIGURE 2. The organization of DPs and LDPs.

At leaf nodes, each node creates a receive–only DP. This DP is the same as the ordinary DP. The only difference is that each point needs to know the communication channel used at the source node. Since each hardware–type broadcast packet only contains shared information, it does not contain any private information associated to every single destination DP. Therefore, each receive–only DP should have the information about the source node whenever it receives a broadcast packet. After setting up the broadcast environment, the broadcast operation can be carried out.

3.2. Simple Broadcast

Simple Broadcast does not utilize the Push–Pull Messaging feature. Packets are broadcasted one by one immediately. Packets may be lost if the destination buffers are not allocated due to the late receive operation.

When Simple Broadcast is started, the message is copied to the shared retransmission buffer. At the same time, packets generated from the message are broadcasted onto the network. Depending on the availability of the destination DPs, some packets may not be able to reach the destination endpoints. Lost packets can be detected by examining sequence numbers embedded in each packet at the destination DPs. In our implementation, we adopted the go–back–n protocol with a window size of 16. When the source DP receives no acknowledgment packet, the DP sends the lost packets based on the LMS. So, each channel can maintain its reliability individually.

Simple Broadcast favors short message broadcast since the probability of getting not enough buffer spaces at destination nodes is low. The broadcast operation can be executed efficiently without retransmission.

3.3. Push–Pull Broadcast

Push–Pull Broadcast uses Push–Pull Messaging as the packet delivery mechanism. Because of this, the behavior of the broadcast operation changes as the size of message increases. Like Push–Pull Messaging, a portion of message is pushed to the destination DPs at the beginning. The rest of the message is pulled depending on the decision of the destination DPs.

Since LDP possessed all the DP features, Push–Pull Messaging can be employed in each LDP. Therefore, during broadcast communication, each LDP can maintain its own communication using Push–Pull Messaging. When broadcasting a long message, buffers at the destination nodes can be managed properly.

Like Push–Pull Messaging, Push–Pull Broadcast divides the broadcast operation into two phases – push and pull. In the push phase, a portion of the broadcast message is pushed to all the destination nodes, similar to Simple Broadcast.

The push phase at the source DP is started after receiving the requests. At all the destination DPs, we only assign one DP to send a Push–Pull acknowledge packet after the DP finishes the push phase since normally, the length of the pushed message is short and packets received at the destination DPs are either saved in the pushed buffer or copied to the destination buffers directly. Instead of requesting all DPs to send acknowledge packets after finishing the push phase, asking one DP would help reduce network traffic.

In case of packets lost in the push phase which rarely happens, the broadcast operation will be suspended temporarily since some DPs were not able to receive all the packets due to losing packets. The operation will be resumed as soon as these DPs receive all the packets through retransmission.

In the pull phase, the source DP broadcasts the remaining packets to all DPs one by one. When broadcasting a long message, buffers at some destination DPs will be drained quickly. This is because buffers may become full or some of the linked LDPs have not consumed buffer entries as fast as possible. Therefore, the broadcast operation may be stopped and thus some DPs cannot receive packets. To efficiently handle retransmission, we allow retransmission to be triggered at each DP conditionally to maintain the reliability of their channels. Only those DPs which lose packets will request for retransmission using the go–back–n protocol. Until all nodes get packets, a Pull–Push acknowledge packet will be sent from an assigned node to resume the broadcast operation. In other words, the operation synchronizes when the buffer becomes full.

4. Performance evaluation

The proposed system, DP–SMP with EQA, was implemented and evaluated on eight Intel MP1.4–complaint SMP machines. Each computer consisted of two Intel Celeron 450 MHz processors with 128 Mbytes of main memory. Each Intel processor had 8–Kbyte L1 instruction cache and 8–Kbyte data cache. The size of the unified L2 cache is 128 Kbytes. The computers were connected by Fast Ethernet with the peak theoretical bandwidth of 100 Mbits/s. Each computer attached one D–Link Fast Ethernet 500TX card with Digital 21140 controller. Linux 2.2.1 was installed on each machine with symmetric interrupt enabled.

The benchmark routines used hardware time–stamp counters in the Intel processor, with the resolution within 100 ns, to time the operations. Each test performed 500 iterations. Among all timing results, the first and last 10% (in terms of execution time) were neglected. Only the middle 80% of the timings was used to calculate the average.

We first evaluated the performance of these communication systems. We then compared the memory consumption of DP–SMP without EQA and DP–SMP with EQA.

4.1. Broadcast latency test

Broadcast latency tests have been carried out for three different communication systems. The first target system (denoted by PPBCAST) uses the implemented Push–Pull Broadcast in DP–SMP with EQA. DP–SMP uses 80 and 780 bytes as the value of BTP_1 and BTP_2 respectively. The second system (denoted by SBCAST) uses Simple Broadcast in our MPI library ported for DP–SMP with EQA. The last system (denoted by MPICH) uses the MPI broadcast operation of MPICH, which adopts a tree–

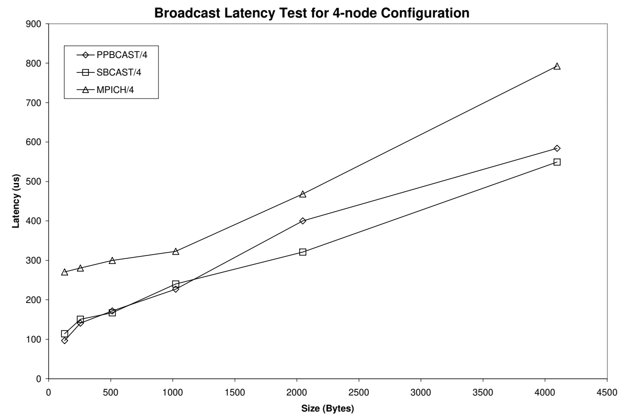

FIGURE 3. Broadcast latency comparison for 4–node configuration.

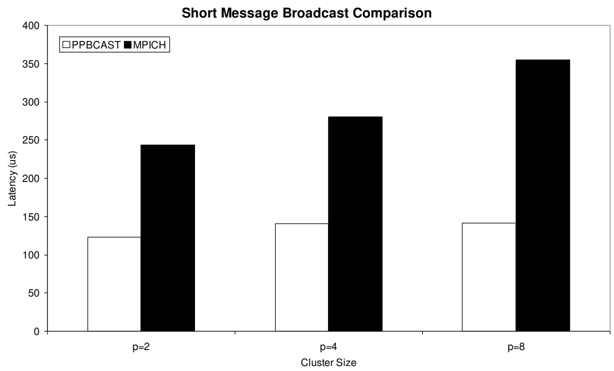

FIGURE 4. 256–byte message broadcast latency comparison.

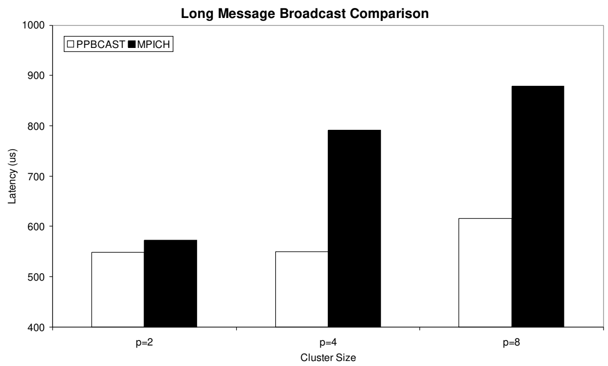

FIGURE 5. 4096–byte message broadcast latency comparison.

based algorithm. In all tests, we further varied the number of nodes in the cluster. We used "/n" to denote the performance of broadcast operation using n nodes in the testing environment. For example, "PPBCAST/4" denotes Push–Pull Broadcast on a cluster size of 4 nodes.

As shown in Figure 3, PPBCAST and SBCAST can achieve shorter latency than MPICH. We further evaluated the broadcast performance for 256–byte and 4096–byte messages in Figure 4 and 5. We found that for 256–byte messages, the degradation of PPBCAST is much less than the degradation of MPICH when the number of nodes increases. Broadcasting a short message for 8–node cluster only uses 142 µs, which is faster than MPICH by 60%. The increase percentage of the latency of broadcasting from 2–node to 8–node is only 16% for PPBCAST while the percentage for MPICH is 45%. The performance results of PPBCAST under different cluster sizes are very similar. But for MPICH, it requires more computation and memory resources to deliver broadcast messages, thus increases the latency time.

For 4096–byte messages, PPBCAST broadcast messages faster than MPICH by 30%. The increase percentage of the latency of broadcasting from 2–node to 8–node is only 12.3% for PPBCAST while the percentage for MPICH is almost 54%.

With the high–speed broadcast operation, SMP nodes can be tightly–coupled together to form a high–performance cluster.

4.2. Memory consumption measurement

We first calculated the size of each data structure used by each DP. Then we calculated the size of the memory that each DP used. We further compared the memory consumption of two broadcast mechanisms – Point–to–Point (P2P) and Hardware Broadcast (HB). Broadcast operation in P2P was implemented based on the point–to–point messaging. Therefore, the size of all DP pairs is,

$\left(s_{SDP}+s_{RDP}\right)\cdot n$ where s_{SDP} is the size of a send–only DP, s_{RDP} is the size of a receive–only DP and n is the size of the cluster.

Broadcast operation in HB was implemented using hardware broadcast facility. Since EQA was utilized for reliability, LDPs were created to minimize the amount of memory consumption. Therefore, the size of all DPs and LDPs is,

$s_{SDP}+(n-1)s_{LDP}+n\cdot s_{RDP}$ where s_{LDP} is the size of a LDP.

In DP–SMP, each send–only DP uses 4 Kbytes as the send queue and 4 Kbytes as the pushed buffer. Each receive–only DP uses 4 Kbytes as the receive queue and 12 Kbytes as the pushed buffer. Each DP uses 24 Kbytes as the retransmission buffer queue. LDP uses 4 Kbytes as the send queue only. Thus, $S_{SDP}=33.8\,Kbytes$, $S_{RDP}=41.4\,Kbytes$ and $S_{LDP}=4.4\,Kbytes$.

Figure 6 shows the memory consumption comparison of two broadcast mechanisms. We compared the fraction of memory space saved.

$$M_{save}(n)=\left(\frac{(s_{SDP}+s_{RDP})\cdot n-(s_{SDP}+(n-1)\cdot s_{LDP}+n\cdot s_{RDP})}{(s_{SDP}+s_{RDP})\cdot n}\right)$$

Thus, $M_{save}(n)=0.386\left(1-1/n\right)$

When the cluster contains 128 nodes, we can reduce 38.3% of memory usage by EQA.

5. Conclusion

The performance of collective communications determines the performance of medium to fine–grained parallel applications. However, collective communications use many computation and communication resources to distribute messages to all nodes in the cluster. As the size of the cluster increases, inadequate manage–

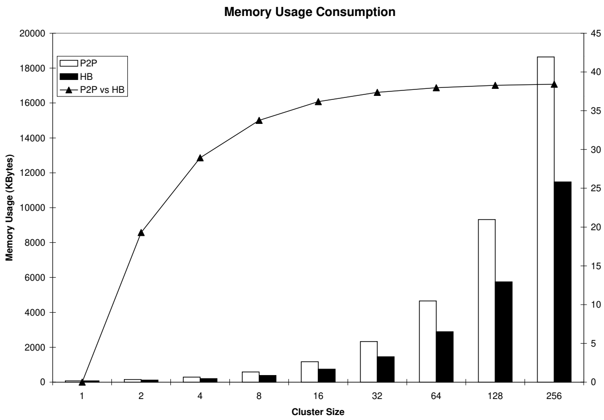

FIGURE 6. Memory usage consumption comparison.

ment of resources will reduce the scalability of the cluster. We introduced EQA, a queue architecture, which can minimize the use of computation and memory resources. Simple and Push–Pull Broadcast are two different variations of the broadcast operation in DP–SMP with EQA. Push–Pull Broadcast for DP–SMP with EQA showed the most promising result in most of the cases.

Currently, the bandwidth of Fast Ethernet is still low compared with the peripheral bus bandwidth. We believe the next important step is to port DP–SMP for Gigabit Ethernet.

6. References

[1] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team. "A Case for NOW (Networks of Workstations)", IEEE Micro, 15(1), February, 1995.

[2] A. Bar–Noy and S. Kipnis. "Designing Broadcasting Algorithms in the Postal Model for Message–Passing Systems", Proc. of the ACM Symposium on Parallel Algorithms and Architectures, pp. 11–22, June, 1992.

[3] G. Ciaccio. "Optimal Communication Performance on Fast Ethernet with GAMMA", Proc. of International Workshop on Personal Computers based Networks Of Workstations 1998, Orlando, March 30/April 3, 1998.

[4] T. von Eicken, A. Basu, V. Buch and W. Vogels. "U–Net: A User–Level Network Interface for Parallel and Distributed Computing", Proc. of the 15th ACM Symposium on Operating Systems Principles, December, 1995.

[5] T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser. "Active Messages: A Mechanism for Integrated Communication and Computation", Report No. UCB/CSD 92/#675, Computer Science Division, University of California, Berkeley, CA 94720, March, 1992.

[6] W. Gropp, E. Lusk, N. Doss and A. Skjellum. "A High–Performance, Portable Implementation of the MPI Message Passing Interface Standard", Journal of Parallel Computing, Vol. 22, No. 6, pp. 789–828, September, 1996.

[7] M. Golin and A. Schuster. "Optimal Point–to–Point Broadcast Algorithms via Lopsided Trees", Proc. of the Fifth Israeli Symposium on Theory of Computing and Systems, pp. 63–73, June, 1997.

[8] S. S. Lumetta, A. M. Mainwaring and D. E. Culler. "Multi–Protocol Active Messages on a Cluster of SMP's", Proc. of Supercomputing '97 High Performance Networking and Computing, November, 1997.

[9] R. M. Karp, A. Sahay, E. E. Santos and K. E. Schauser. "Optimal Broadcast and Summation in the LogP Model", Proc. of the 5th Symposium on Parallel Algorithms and Architectures, June, 1993.

[10] C.–M. Lee, A. Tam and C.–L. Wang. "Directed Point: An Efficient Communication Subsystem for Cluster Computing", Proc. of the 10th International Conference on Parallel and Distributed Computing and Systems, pp. 662–665, Las Vegas, October, 1998.

[11] J. Shen, J. Wang and W. Zheng. "A New Fast Message Passing Communication System for Multiprocessor Workstation Clusters", Technical Report, Department of Computer Science and Technology, Tsinghua University, China, 1998.

[12] A. S. Tanenbaum. "Computer Networks", 3rd Edition, Prentice–Hall International, Inc., 1996.

[13] Y. Tanaka, M. Matsua, M. Ando, K. Kubota and M. Sato. "COMPaS: A Pentium Pro PC–based SMP Cluster and its Experience", Proc. of International Workshop on Personal Computers based Networks Of Workstations 1998, Orlando, March 30/April 3, 1998.

[14] Y.–C. Tseng and J.–P. Sheu. "Toward Optimal Broadcast in a Star Graph using Multiple Spanning Trees", IEEE Transactions on Computers, Vol. 46, Issue 5, pp. 593–599, May, 1997.

[15] A. Tam and C.–L. Wang. "Realistic Communication Model for Parallel Computing on Cluster", Proc. of International Workshop on Cluster Computing, pp. 92–101, 2–3 December, 1999.

[16] K.–P. Wong. "High–Speed Network Interface for Commodity SMP Clusters", Master Thesis, Department of Computer Science and Information Systems, The University of Hong Kong, 2000.

[17] K.–P. Wong and C.–L. Wang. "Push–Pull Messaging: A High–Performance Communication Mechanism for Commodity SMP Clusters", Proc. of International Conference on Parallel Processing, Fukushima, Japan, 21–24 September, 1999.

[18] W. Zhu, D. Lee and C.–L. Wang. "High Performance Communication Subsystem for Clustering Standard High–Volume Servers using Gigabit Ethernet", Proc. of the 4th International Conference on High Performance Computing in Asia–Pacific Region, Beijing, China, 14–17 May, 2000.