Adaptive Receiver Window Scaling: Minimizing MPI Communication Memory over InfiniBand

SAYANTAN SUR, MATTHEW J. KOOP AND D. K. PANDA

Technical Report
OSU-CISRC-12/06-TR82

Adaptive Receiver Window Scaling: Minimizing MPI Communication Memory over InfiniBand *

Sayantan Sur Matthew J. Koop Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University
{surs, koop, panda}@cse.ohio-state.edu

Abstract

Commodity clusters are rapidly scaling up to tens-of-thousands of processors. With the increasing scale, providing scalable design and implementation of the Message Passing Interface (MPI) is a challenge. Current generation MPI implementations provide scalable modes of operation, but they lack adaptive mechanisms to dynamically adapt resource consumption as per application requirements. As a result, intimate knowledge of application characteristics is required to minimize resource consumption by the MPI layer. In this paper, we present a mechanism which dynamically adapts resource consumption according to application runtime characteristics enabling the MPI layer to consume only minimum amounts of resources. We evaluate this mechanism in the context of MVAPICH, a high-performance, open-source MPI implementation over InfiniBand. Experimental evaluation of our proposed design with NAS Parallel Benchmarks and NAMD (apoa1 dataset) reveals that the adaptive mechanism can provide the best available performance while consuming around factor of two to three lesser amount of communication memory than the original non-adaptive design. In addition, our experimental analysis shows that the proposed adaptive mechanism can achieve memory usage very close to the minimum required by MVAPICH to deliver the best possible performance. Further, we provide analysis of of our proposed design in combination with the effect of of low-level InfiniBand flow-control timers on end-application memory usage. To the best of our knowledge, this is the first research work which provides this kind of design and analysis for high-performance MPI over InfiniBand.

1 Introduction

Over the past decade, cluster computing has become quite popular, owing largely to its cost benefits from commodity components. The size of clusters is increasing rapidly to tens-of-thousands of processors. Generally clusters used for high-performance computing are connected with a high-performance, low-latency interconnect. InfiniBand [2] is a popular high-performance interconnect, offering low latency (1.5-3.0 μs) and high bandwidth (multiple GigaBytes/second). In addition to high-performance, InfiniBand also provides many advanced features such as Remote Direct Memory Access (RDMA), atomic operations, multi-cast, and QoS. Several top clusters

*This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco systems, Linux Networx and Sun Microsystems; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, Dell, Microway, PathScale, IBM, SilverStorm and Sun Microsystems.

in the Top500 list [14] have employed InfiniBand as their primary interconnect. Sandia Thunderbird [10] and NASA/Ames Columbia [6] are two such examples. The Message Passing Interface (MPI) [5] is the de-facto standard in writing parallel scientific applications on large-scale clusters. Hence, the performance and scalability of MPI libraries is crucial for end-application performance on large clusters.

Recently, memory consumption within MPI implementations over InfiniBand have been subject to much research [12, 13, 11]. Memory scalability has been achieved primarily by using the "Shared Receive Queue" (SRQ) feature of InfiniBand. MVAPICH [7], a high-performance and open-source MPI implementation is designed to leverage the SRQ feature. However, current mechanisms are not adaptive. As a result, in order to reduce communication memory, the user has to know up-front how much communication memory is required for attaining the best possible performance. This is often hard for the end user to know. Thus, two open questions remain:

- Can we design an adaptive mechanism which dynamically adjusts the amount of memory required for an application during runtime, using only minimal amounts of memory?

- How close is the communication memory consumption in the adaptive scheme compared to the minimum required to attain the best recorded performance?

In this paper, we aim to answer the above two questions. We present a mechanism which dynamically adapts (at runtime) the amount of communication memory used according to application demands. We have evaluated our mechanism with MVAPICH on a 32-node, dual-processor cluster equipped with InfiniBand. Our experimental evaluation with the NAS Parallel Benchmarks [1] (Class B) and NAMD [9] (apoa1 dataset), shows that our new mechanism can achieve the best possible performance with almost a factor of two to three lesser amount of communication memory than the original design. In addition, our experiments reveal that our mechanism consumes very close (within 500KB) to the *minimum* amount of memory required to run applications with the best possible performance. Further, we provide analysis of our proposed design in combination with the effects of low-level InfiniBand flow control timers on end-application memory usage.

The rest of the paper is organized as follows: Section 2 provides the required background about InfiniBand and MVAPICH. Section 3 describes our adaptive mechanism. Section 4 presents our experiments and their analysis. Related research is described in Section 5, and finally, the paper concludes in Section 6.

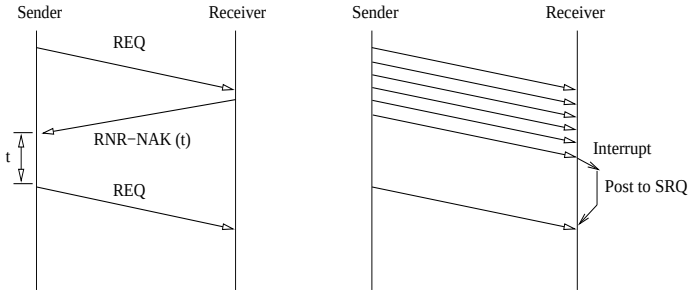
2 Background

In this section we provide the required background for the work done in this paper. First, we discuss the features and properties of InfiniBand as applicable to this research work. Second, we describe the design of the MPI layer (namely, MVAPICH [7]) on top of these InfiniBand primitives.

2.1 InfiniBand Overview

InfiniBand is a high-performance and feature-rich network architecture. In this section we discuss a subset of these features which are directly applicable to our research work. Details relating to other InfiniBand features can be obtained from IBA specification [2].

InfiniBand Communication Model: The InfiniBand Architecture [2] (IBA) defines a switched network fabric for interconnecting compute and I/O nodes. In an InfiniBand network, hosts are connected to the fabric by Host Channel Adapters (HCAs). A queue based model is used in InfiniBand. A Queue Pair (QP) consists of a send queue and a receive queue. Communication operations are described in the Work Queue Requests/Entries (WQR/WQE), or descriptors, and submitted to the work queue. It is a requirement that all communication buffers be posted into receive work queues before any message can be placed into them. In addition, all communication buffers need to be registered (locked in physical memory) before InfiniBand can either send from or receive data into that



(a) RNR-NAK Protocol (b) SRQ Limit Reached Event

Figure 1. IBA Flow Control Mechanisms for Shared Receive Queues

memory location. This restriction is imposed to ensure that memory is present when HCA accesses the memory. Finally, the completion of WQRs is reported through Completion Queues (CQ).

InfiniBand Transport Services: IBA provides several types of transport services: Reliable Connection (RC), Unreliable Connection (UC), Reliable Datagram (RD), and Unreliable Datagram (UD). On current-generation hardware, the RC transport provides the best performance and most features. It supports two-sided send/receive operations as well as one-sided RDMA Read/Write operations. In addition, it supports one-sided atomic operations and QoS features. In this paper, we focus on the RC transport exclusively.

Shared Receive Queue (SRQ): SRQ is a feature of IBA which allows sharing of one receive queue for multiple QPs. Using this feature, QPs have their individual send queues, but have only one receive queue. This feature allows efficient sharing of receive buffers amongst multiple QPs, especially when the incoming message rate on any QP cannot be predicted in advance. This feature maps very well to the requirement of MPI implementations, and is currently used by both MVAPICH [7] and OpenMPI [11].

Flow Control Mechanism for SRQ: Although the SRQ feature provides very efficient buffer sharing mechanism amongst multiple QPs, it suffers from inherent flow control related issues since shared buffers are consumed in a First-come-first-served (FCFS) fashion. For example, a sending process can no longer be guaranteed to find a receive buffer once its message reaches the receiver. IBA provides a NAK based flow-control mechanism to handle the case where there are incoming messages and no WQEs in the SRQ. The receiver side HCA sends a Receiver not Ready NAK (RNR-NAK) to the sender indicating that this message should be retried after a certain time interval t. The value of t can be user supplied at the time of QP initialization. Possible values of t range from 0.1ms to around 655ms. This protocol is shown in Figure 1(a). Another mechanism provided to avoid keeping the SRQ empty for extended periods of time is the `SRQ_LIMIT_REACHED` event. When the number of available SRQ buffers drops below a configurable limit, then the HCA generates an interrupt which can be handled by upper level software to refill the SRQ. This is illustrated in Figure 1(b). Both the RNR-NAK and `SRQ_LIMIT_REACHED` event mechanisms are complementary and can be utilized appropriately by upper level software.

2.2 MVAPICH Design Overview

MVAPICH [7] is a popular implementation of MPI over InfiniBand. It uses several InfiniBand services like Send/Receive, RDMA-Write, RDMA-Read, and Shared Receive Queues to provide high-performance and scalability to MPI applications. There are two major protocols used. The first is the *Eager Protocol*, which is used to transfer small messages. The second protocol used is the *Rendezvous Protocol*, which is used for large messages. In order to avoid buffering large messages inside the MPI library, the Rendezvous protocol negotiates the availability of receive buffer by using control messages. After the

negotiation phase, the messages are sent directly to receiver user memory with the use of RDMA. These control messages used by the Rendezvous protocol are small in size and are sent over the Eager protocol. Thus, the Eager protocol can be used for MPI application generated small messages as well as Rendezvous control messages.

The Eager protocol requires the presence of "pre-allocated" communication buffers, in order to avoid any runtime costs and achieve low latency. The Rendezvous protocol does not require any additional buffer space other than the control messages sent over the Eager protocol. Hence, only the Eager protocol consumes communication memory in a MPI process. Since, in this paper, we are studying the impact of communication memory usage on performance, we will focus on the Eager protocol.

Of the several implementations of the Eager protocol in MVAPICH, the most scalable one is over SRQ. The MVAPICH SRQ channel has been described in detail in [12, 13]. This channel relies on the `SRQ_LIMIT_REACHED` event provided by IBA. When the number of available receive requests on the SRQ drops below a certain limit, an interrupt generated by the HCA is handled by MVAPICH to post more receive descriptors to the SRQ. The current implementation is not adaptive in nature. i.e. the user can only specify a specific receive window size for the entire run of the application (for all the processes in the job). This is not very convenient, since the user should know up-front, the minimum window size required to achieve the best performance. Also, the user has no way of controlling window sizes on different processes based on application requirement. Thus, the current scheme enforces a global static policy on the size of the receive window.

3 Design of Adaptive Receive Window Scaling

In this section we discuss the design of our proposed adaptive receiver window scaling mechanism.

As mentioned in Section 2.2, MVAPICH relies on the `SRQ_LIMIT_REACHED` event to detect if the SRQ is low on receive WQEs. If there are no `SRQ_LIMIT_REACHED` events, then it can be assumed that enough receive operations are already made available to the network than send operations. However, if there are `SRQ_LIMIT_REACHED` events, then it indicates the MPI application is trying to send more messages than the receiver initially estimated. Whenever the event occurs, our design increases the available receiver window size according to the following equation:

$$W_{current} = min(2^{nlimit} \times W_{initial}, W_{max})$$

Where,
$W_{current}$ = current receive window,
$nlimit$ = total number of limit events,
$W_{initial}$ = initial window size,
W_{max} = maximum allowable window size

The threshold (given by $Limit_{current}$) for the next `SRQ_LIMIT_REACHED` event is set as follows:

$$Limit_{current} = W_{current}/2$$

We choose an exponential function to grow the window size, so that our scheme can very quickly adapt to the application characteristics. Currently, our design does not reduce the window size. This is because, it is hard to estimate for the MPI library when the application will again choose to send a burst of messages. Even if the application does not utilize the entire window (allocated after the burst), the buffers allocated could still be used for keeping unexpected messages, etc.

In our implementation, the user can vary the $W_{initial}$ and W_{max} values. For example, a user may select a lower W_{max} value to set a limit on the amount of memory the MPI layer uses for communication. However, when the W_{max} is set much lower, one factor which comes into play is the RNR-NAK timeout values. As mentioned in Section 2.1, the RNR-NAK timeout value impacts the rate at which the sender retries, in case send operations do not find corresponding receive requests on the receiver side. A very low RNR-NAK value may lead to significantly

increased traffic in the fabric as well as more SRQ_LIMIT_REACHED events, thus leading to increased memory usage. On the other hand, a very high RNR-NAK value may lead to very delayed retries, leading to degraded performance.

4 Experimental Evaluation

In this section we present experimental results for the design alternatives described in the previous section. We evaluate both the performance and memory used while executing the NAS Parallel Benchmarks [1] (Class B) and NAMD [9] (apoa1 dataset).

In order to obtain the communication memory usage, we inserted profiling code inside MVAPICH. The information collected by the profiling code was collected using MPI_Reduce inside MPI_Finalize. The profiling code does not impact the performance of the application (verified with latency/bandwidth micro-benchmarks). Our experimental results are divided into two categories. In the first category, we measure the impact of various values of W_{max} on both application performance and memory consumption. In the second category, we measure the impact of choosing several different RNR-NAK timer values for different values of W_{max}.

4.1 Experimental Platform

Our experimental platform is a 32 node dual Intel 3.7 GHz cluster. Each node is equipped with 2GB of main memory and PCI-Express interface. The nodes have MHGA28-1T Mellanox DDR HCAs with firmware version 5.1.4. The cluster is connected with a 144 port Flextronics DDR switch. The Verbs layer used is from OpenFabrics [8] (OFED 1.1). The Linux kernel version used is 2.6.17.

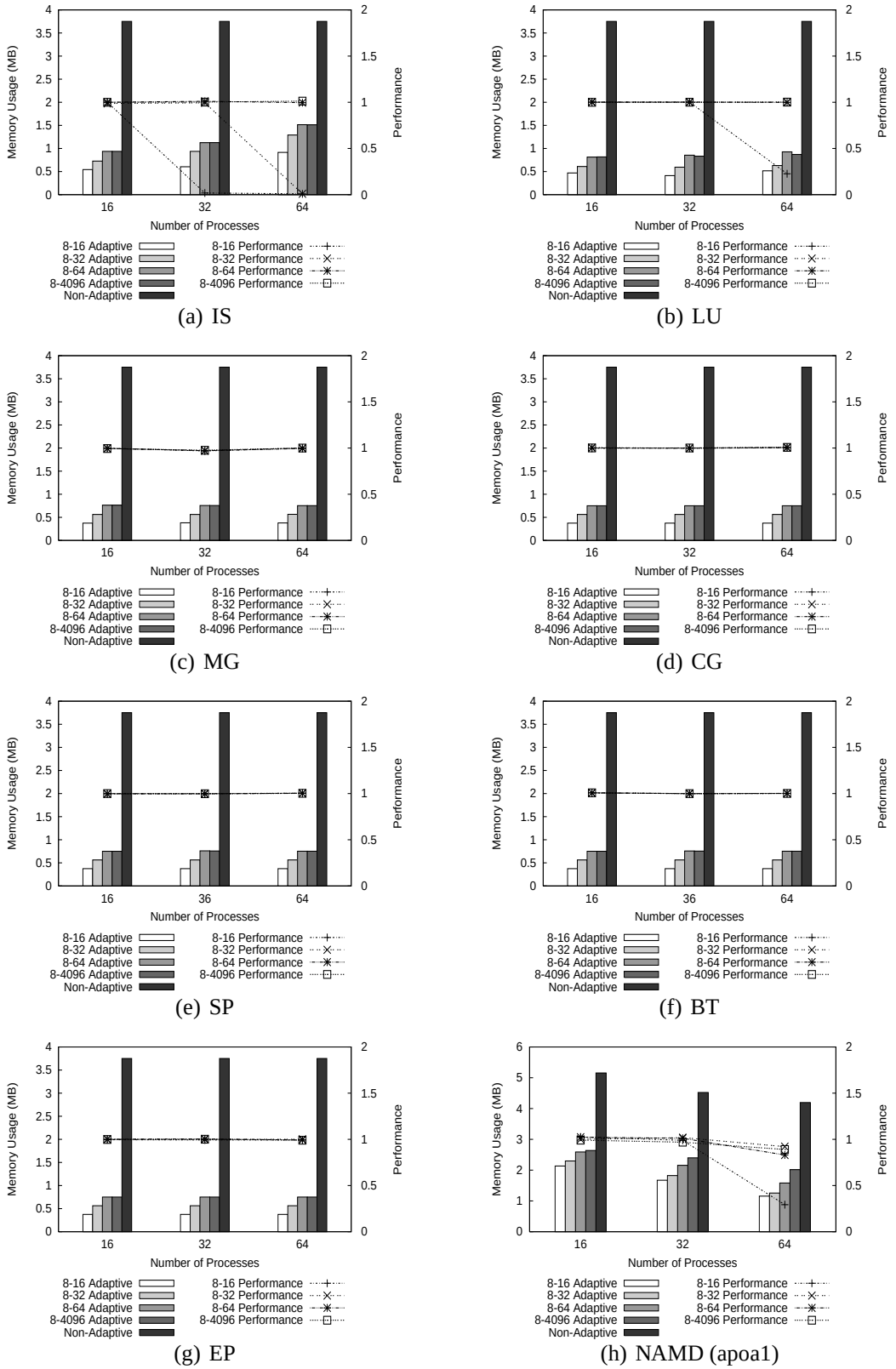
4.2 Impact of W_{max} on Performance and Memory Usage

In this section, we present our evaluation of our adaptive receive window scaling mechanism with various values of W_{max}. In our experiments, we vary W_{max} from 16 to 4096. The value of $W_{initial}$ is fixed at 8 for all the experiments.

The experimental results are shown in Figure 2. We present both the average memory usage (i.e., memory used by an average MPI process) and the relative performance as compared to the best recorded performance of MVAPICH (using the "default" configuration). The default configuration consists of the settings used in MVAPICH version 0.9.8. The default configuration does not have the adaptive receiver window scaling mechanism, rather it uses a fixed window of 512. Relative performance of 1 indicates that performance remains the same, and < 1 indicates degraded performance. The configurations used in this experiment are named as "$W_{initial} - W_{max}$ Adaptive" and "Non-Adaptive" (the original design).

As seen in Figures 2(a) to 2(h), the memory consumed by $W_{max} = 4096$ (column "$8 - 4096$ Adaptive") is considerably lesser than that of the default combination, ranging from a factor of two or three better. In addition, from Figures 2(c) to 2(g), we can observe that all values of W_{max} perform equally well, as compared to the default "Non-Adaptive" combination. Further, the difference in memory consumption between $W_{max} = 16$ and $W_{max} = 4096$ is just around 500KB.

In Figures 2(a), 2(b) and 2(h), we see that with $W_{max} = 64$ and greater achieves the same performance as the default combination. For values of $W_{max} = 32$ and below (in case of Figure 2(h), $W_{max} = 16$), there is performance degradation. This indicates that window of at least that much is required to obtain the best available performance. $W_{max} = 4096$ consumes only around 500KB more of memory (in case of Figure 2(h), 1MB more), and is able to provide the best possible performance. Thus, we note that our adaptive receive window scaling can achieve the best possible performance with almost the *minimum* amount of memory required with $W_{max} = 4096$. Current generation Mellanox HCAs (like the one used for this paper) support up-to $W_{max} = 16,384$. It is our expectation that even on very large clusters, the adaptive receive window scaling mechanism can scale to consume only as much memory as applications require for attaining best performance.



(a) IS (b) LU

(c) MG (d) CG

(e) SP (f) BT

(g) EP (h) NAMD (apoa1)

Figure 2. Performance of NAS Benchmarks and NAMD with various values of W_{max}

4.3 Impact of RNR-NAK Timer on Memory Usage

In this section, we present the experimental results which show the impact of the choice of the RNR-NAK timer on the memory consumed by an application with varying W_{max}.

The experimental results are shown in Figure 3. The various values of RNR-NAK timers tried were from 0.01ms to 655.36ms. When a message arrives at the receiver and there is no SRQ WQEs available, the sender gets a RNR-NAK with a specific timer value. According to the IBA specification [2], the requester HCA (i.e. the HCA sending the message), should re-try the message after waiting for at-least this amount of time. There is no upper bound specified. Nevertheless, we found on the Mellanox HCAs, this RNR-NAK timer to be in general a good indicator of how often messages are re-tried.

In Figure 3(a), we can see the impact of reducing the RNR-NAK timer with very low maximum SRQ sizes (W_{max}). Even though the W_{max} value is low, due to very high frequency of retries, more memory is consumed. As soon as memory is made available to the MPI library, new messages are received and marked as "unexpected". Since unexpected messages also count towards the communication memory usage of the MPI library, the usage increases as a result. The other NAS benchmarks and NAMD show the trend as seen in Figure 3(b). These show that even though the W_{max} value is quite low, the RNR-NAK value has literally no impact on the overall performance. The reason for this can be obtained from observations in Section 4.2, where we see that MG performs equally well even for very low W_{max} values.

5 Related Work

Memory usage inside MPI libraries over InfiniBand is a well researched topic. Since the most popular InfiniBand transport is connection oriented, resources dedicated *per-connection* increase the overall memory usage on large scale clusters. Liu, et al performed a comparison of the earliest MPI implementation over InfiniBand (MVAPICH) with MPI implementations over Myrinet and Quadrics in [4], which showed this trend of increasing memory with the number of increasing processes. The Shared Receive Queue feature of InfiniBand was not available at that time. Yu, et al introduced an "adaptive" connection strategy in [15] which allowed InfiniBand connections to be set-up only after certain number of messages were exchanged between a process pair. The MVAPICH SRQ channel was described in [12], a much more detailed application level evaluation was carried out in [13] which revealed that the SRQ design required only around $5 - 10$ MB of communication memory while executing various applications and benchmarks on a 64-node InfiniBand cluster. Other MPI implementations, such as Open MPI also have Shared Receive Queue support [11]. While most of the above research works focussed on either reducing the connection memory or receive buffer and communication memory, Koop, et al showed in [3] the impact of reducing the number of outstanding send operations combined with message coalescing on the overall memory usage of the MPI library.

This research work complements other existing works by evaluating the effect of adaptively increasing the receiver side SRQ window on the application memory usage and performance. In addition, it evaluates the combined effect of different RNR-NAK thresholds and low SRQ window sizes on the end application performance. To the best of our knowledge, this is the first research work which provides this kind of design and analysis.

6 Conclusions and Future Work

With the increasing scale of clusters, it is essential that MPI implementations have scalable design and consume the minimum possible resources. In this paper, we proposed a mechanism for adaptively scaling the receive window to obtain the best possible performance with the least amount of memory resources. Our experimental evaluations have proved that not only does our mechanism consume a fraction of memory as compared to the default design, memory required by it is very close to the *minimum* required mem-

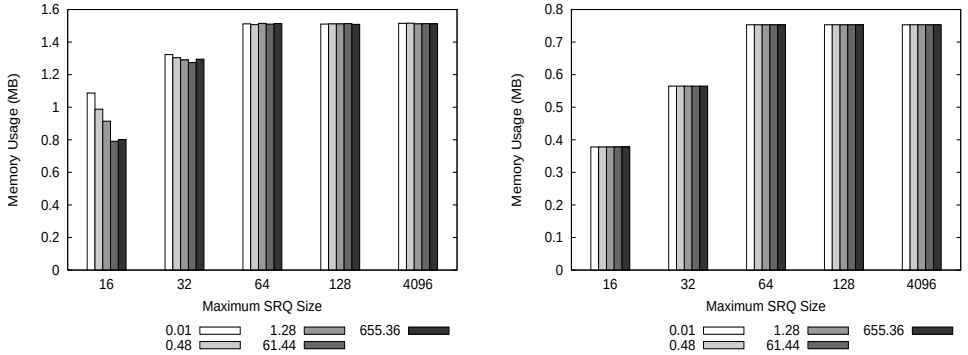


Figure 3. Impact of RNR-NAK Timer on Memory Usage

ory for attaining best performance. In addition, we have also evaluated our proposed mechanism in combination with the effect of low-level InfiniBand flow control timers to show the effect on end application communication memory usage.

In the future, we plan to evaluate this design on much larger scale clusters in conjunction with methods and heuristics to reclaim unused memory with least impact to end application performance. We also plan to study application characteristics in detail to understand the patterns of memory requirement from the MPI library.

References

[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. volume 5, pages 63–73, Fall 1991.

[2] InfiniBand Trade Association. InfiniBand Architecture Specification, Volume 1, Release 1.2. http://www.infinibandta.com.

[3] M. J. Koop, T. Jones, and D. K. Panda. Reducing connection memory requirements of MPI for InfiniBand clusters: A message coalescing approach. Technical Report UCRL-CONF-226304, Lawrence Livermore National Laboratory, November 2006.

[4] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda. Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics. In *Supercomputing(SC)*, 2003.

[5] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.

[6] NASA Project: Columbia. Columbia Supercomputer. http://www.nas.nasa.gov/About/Projects/Columbia/columbia.html.

[7] Network-Based Computing Laboratory. MPI over InfiniBand Project. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/.

[8] Open Fabrics Alliance. Open Fabrics Enterprise Distribution. http://www.openfabrics.org/.

[9] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale. NAMD: Biomolecular Simulation on Thousands of Processors. In *Supercomputing*, 2002.

[10] Sandia National Laboratories. Thunderbird Linux Cluster. http://www.cs.sandia.gov/platforms/ Thunderbird.html.

[11] G. Shipman, T. Woodall, R. Graham, and A. Maccabe. Infiniband Scalability in Open MPI. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[12] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared Receive Queue Based Scalable MPI Design for InfiniBand Clusters. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[13] S. Sur, M. J. Koop, and D. K. Panda. High-Performance and Scalable MPI over InfiniBand with Reduced Memory Usage: An In-Depth Performance Analysis. In *Super Computing*, 2006.

[14] The Top 500 Project. The Top 500. http://www.top500.org/.

[15] W. Yu, Q. Gao, and D. K. Panda. Adaptive Connection Management for Scalable MPI over InfiniBand. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.