Adaptive Transport Service Selection for MPI with InfiniBand Network

Masamichi Takagi
RIKEN Advanced Institute for Computational Science
masamichi.takagi@riken.jp

Norio Yamaguchi
RIKEN Advanced Institute for Computational Science
norio.yamaguchi@riken.jp

Balazs Gerofi
RIKEN Advanced Institute for Computational Science
bgerofi@riken.jp

Atsushi Hori
RIKEN Advanced Institute for Computational Science
ahori@riken.jp

Yutaka Ishikawa
RIKEN Advanced Institute for Computational Science
yutaka.ishikawa@riken.jp

ABSTRACT

We propose a method which adaptively selects InfiniBand transport services used for source and destination peers to improve performance while limiting memory consumption of the MPI library. There are two major choices of IB transport services available, i.e., Reliable Connection (RC) and Dynamically Connected (DC), each of which is selected for each pair of source peer and destination peer. RC is faster than DC for all communication patterns except for the case where there are many active RCs. It also consumes a lot of memory when there are many processes. DC, on the other hand, consumes less memory than RC but its performance drops when sending messages to different destinations or when many DCs sends a message to the same destination DC. Therefore, the library should find the best mapping of RCs and DCs to pairs of source peer and destination peer according to the communication pattern of the application. Our method finds a good mapping by comparing potential latency benefits for candidate mappings. It achieves 13%-19% latency reduction when compared to the methods using only DCs in micro-benchmarks representing communication patterns problematic to RC or DC with 64 processes.

Categories and Subject Descriptors

C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*High-speed*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Software libraries*

1. INTRODUCTION

Clusters, built of commodity hardware components, are the most prevalent architecture for High Performance Computing (HPC) at present. Commodity components provide significantly better cost-performance benefits over architectures targeted at smaller markets. With respect to networking, InfiniBand (IB) [4] has been increasingly deployed as the de-facto interconnect solution in commodity clusters because of its high bandwidth and low latency benefit over other network technologies. At the same time, the Message Passing Interface (MPI) [7] is the most widely used programming model in these distributed memory machines.

As systems sizes increase to exascale and beyond, scalable message passing faces two important issues. One requirement is to reduce communication latency because it can take a substantial portion of the execution time of applications. Additionally, memory consumption of the run-time software as the function of the growing number of compute nodes in the system must be limited not to hinder execution of massively parallel distributed programs. We focus on designing an MPI library, especially how to select IB transport services to meet these requirements. There are two major choices of transport services, i.e. Reliable Connection (RC) and Dynamically Connected (DC) (DC is an extension done by Mellanox [2]). IB provides a concept of transport entity called Queue Pair (QP). An RC-QP is used on the source peer and another is used on the destination peer. A DC-QP is used in the similar manner. One RC-QP can send to one specific RC-QP and therefore it cannot be shared with multiple peers, whereas one DC-QP can send to any DC-QPs and therefore one DC-QP can be shared with multiple peers. Communication using RC-QP is faster than that of DC-QP in all communication patterns except when there are many active QPs, but RC-QP consumes more memory than DC-QP because $N \times M$ RCs per process are needed in a configuration with N processes times M nodes. On the other hand, DC-QP consumes less memory than RC-QP because only one DC-QP is needed in a configuration with N processes times M nodes but it performs poorly when a DC-QP sends many messages to different DC-QPs in a short period of time or when many DC-QPs send a message to the same DC-QP at the same time. Therefore, the MPI library should find the best mapping of RC-QPs and DC-QPs to pairs of source peer and destination peer, or in other words, the best binding between RC-QPs and DC-QPs and peers, according to the communication pattern of the application. Conventional methods fail to solve the issue. For example, neither the method proposed by Subramoni et al. [9] nor the one proposed by Koop et al. [5] mixes RCs and DCs.

We propose an IB transport service selection method that finds a good mapping. It compares potential latency bene-

ExaMPI2015, November 15-20, 2015, Austin, TX, USA
 ISBN 978-1-4503-3998-8/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2831129.2831132

fits of the candidate mappings and chooses the best one to find the good mapping. The memory consumption is limited by only including the mappings which satisfy the limit in the candidates. The benefit is calculated against the communication statistics for a certain historical time-window. Our method makes the time-window slide, as the time goes by, by comparing the benefits of different mappings with different time-windows and choosing the best one.

The remainder of this paper is organized as follows. In Section 2, we explain the background. In Section 3, we describe our IB transport selection method. In Section 4, we explain the evaluation results. In Section 5, we discuss related work. In Section 6, we conclude the paper.

2. BACKGROUND AND MOTIVATION

In this section we present an overview of the MPI library's organization and discuss the attributes of available transport services provided by the InfiniBand network. At the same time, we highlight the main issues to motivate our work.

The MPI library (assuming MPICH implementation) usually uses the following three layers of abstraction for communication. The first layer, which we call the transport service, controls the network hardware and provides low-level communication primitives. The second layer, called the software message channel, is built on top of the transport service and it provides abstracted commands including buffering and message arrival detection. The third layer, which we call the software message protocol, is in turn built on top of the software message channel and it provides commands following the MPI API.

There are two major choices regarding transport services in case of InfiniBand networks, which we will discuss in detail below. They both have strengths and weaknesses and the right combination of the two depends on the application's communication pattern. Therefore, it is important to select the right mapping of transport services to pairs of source peer and destination peer to hide weaknesses. The selection can be performed by using communication statistics over a time-window in application execution time, which reflects the communication pattern of the given interval. Additionally, because communication pattern changes over time, another issue is that we need to decide whether or not we should calculate and apply a new mapping using more recent statistics that reflects the corresponding changes.

We will now explain the characteristics of IB transport services. Subsequently, communication patterns of applications are observed to design the selection for the right mapping of transport services to pairs of source peer and destination peer.

2.1 Transport Service

IB provides many transport services and we focus on Reliable Connection (RC) and Dynamically Connected (DC) for the following reasons. Our method is a framework and other transport services can be added later. And therefore our strategy is first to evaluate the framework with small number of representative transport services with minimum effort. Unreliable Datagram (UD) can be represented by DC because DC provides almost the same performance [9] and consumes almost the same memory as UD. Extended RC (XRC) can be represented by RC because XRC provides almost the same performance as RC and it reduces memory consumption by the number of processes per node, which can be reduced to one by using hybrid parallelization, when compared to RC. DC is an extension proposed by Mellanox. IB provides the concept of an event queue for packet arrival detection and command completion notification, which is called a Completion Queue (CQ). The IB terminology for a transport service entity is a Queue Pair (QP). We explain their performance degradation issues caused by problematic communication patterns.

Reliable Connection (RC).

RC has two send methods. The first one is called *sendrecv* where a receiver side posts a receive command which specifies the address to which the arriving message is stored and a sender side posts a send command to send a message. The arrival of message is detected by consulting its CQ. The second one is called RDMA where a sender side posts a send command which specifies the remote address to which the message is written. The arrival of message is detected by polling on the message. RDMA is faster than *sendrecv* because of the difference in the arrival detection since consulting CQ incurs an additional cache-miss for a CQ entry and adds the overhead of the IB library's involvement when compared to polling.

An RC-QP can send messages only to the RC-QP specified at its initialization time. In other words, one sender RC-QP is paired with one receiver RC-QP. Therefore, RC's memory consumption is proportional with the number of nodes (M) times the number of MPI processes (N) per node, requiring $N \times M$ RC-QPs per process. Reportedly, RDMA of RC is faster than RDMA of DC by up to approximately ten percent [9] under various communication patterns. However, it experiences performance degradation when there are many active RC-QPs. This is because the IB Host Channel Adapter (HCA) has an on-board cache of QPs and cache slashing occurs when the number of active QPs are above the hardware threshold [5].

Dynamically Connected (DC).

A QP of Dynamically Connected (DC) can perform both *sendrecv* and RDMA. A DC-QP can send packets to any DC-QPs at the expense of connection/disconnection request overhead. It specifies the network name of the receiver when sending a message and sends a connection request when the receiver is a new one and the receiver creates a temporal communication context. Therefore, N MPI processes times M nodes configuration requires only one DC-QP per process if all processes use the DC-QP and hence scalability issue in memory consumption is less severe than RC. On the other hand, DC is slower than RC because of the connection transaction overhead and two performance degradation issues. Fig. 1 explains the overhead and the first performance degradation issue [9]. All the three processes use DC-QPs and Process 2 sends a packet to Process 1 and then Process 2 sends a packet to Process 3. Process 2 needs to send a connection-request packet to Process 1 when trying to send a packet to Process 1, which incurs overhead when compared to RC-QP. Process 2 needs to disconnect from Process 1 when trying to send a packet to a new destination, in this case Process 3, because a DC-QP remembers only one connection context. And hence it sends a disconnection-request packet to Process 1, which incurs another kind of overhead when compared to RC-QP. Process 2 is not allowed to send the disconnection-request packet until it receives the last ac-

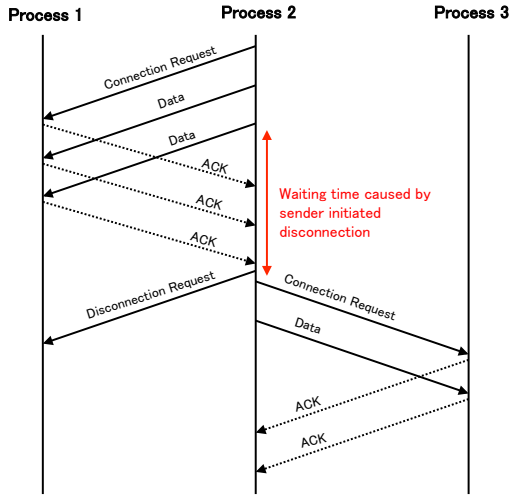

Figure 1: Waiting time caused by sender initiated disconnection in Dynamically Connected (DC) Queue Pairs (QP).

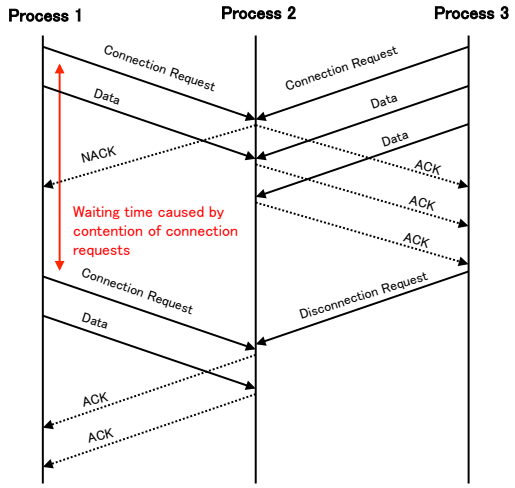

Figure 2: Waiting time caused by contention of connection requests in Dynamically Connected (DC) Queue Pairs (QP).

knowledge packet from Process 1. Therefore, DC slows down when sending many messages to different DC-QPs. Fig. 2 explains the second performance degradation issue [2]. All the three processes use DC-QPs and Process 3 sends a packet to Process 2 and then Process 1 sends a packet to Process 2. Process 1 sends a connection-request packet to Process 2 but it contends with the connection-request packet sent from Process 3 and Process 1 gets a NACK. Process 1 needs to wait until Process 2 receives a disconnection request from Process 3. The wait can be performed by back-off and retry algorithm. Therefore, DC slows down when many DC-QPs try to send a message to the same DC-QP at the same time.

2.2 Communication Pattern

Communication patterns of applications are observed with the following key questions in mind.

1. Do any problematic communication patterns appear? Let us define the communication pattern problematic to IB transport services. (A) One rank [1] sends many messages to different destination ranks. (B) Many source ranks send messages to one rank. (C) Many destination or source ranks are involved from the viewpoint of one rank.

2. What kind of mapping of IB transport services (i.e., RC-QP or DC-QP) to source ranks and destination ranks would be effective for the problematic patterns?

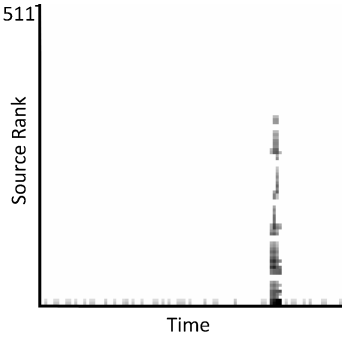

Figure 3: Source rank distribution over time observed from rank zero in SMG2000 with 512 processes.

Fig. 3 shows the source rank distribution over time observed from rank zero in SMG2000[1] using 10 processes × 63 nodes. The entire execution time is divided into one hundred periods and the source ranks are grouped into one hundred groups. Message receive events are counted in each period and source rank group. The X-axis shows the periods, the Y-axis shows the source rank group. The darkness of the dots indicates receive counts, which is calculated by a normalized logarithm of the counts.

Narrow horizontal line in the figure implies that a small number of source ranks are involved and a large number of messages are received and therefore it is not a problematic communication pattern. It is beneficial to use RC-QPs for those source ranks because combination of RC and RDMA is the fastest and the benefit of latency reduction outweighs the overhead of preparing RC-QP and RDMA buffer since the receive count is large. The vertical line indicates that the application performs a communication involving many ranks but it does not use an MPI collective function. This involves hundreds of source ranks and therefore it is a problematic communication pattern (A) or (B) or (C). There are three strategies to handle the pattern. The first one is to create the limited number of RC-QPs and DC-QPs on one rank and make them handle source ranks and destination ranks (make DC-QPs handle the large portion) when the pattern (C) is happening. In this way, the number of active RC-QPs is reduced and the performance degradation issue of RC is avoided. The second one is to create many RC-QPs and DC-QPs on one rank and make them handle many destinations when the pattern (A) is happening. In this way, the number of destinations for one DC-QP on one rank is reduced and the performance degradation issue of DC is avoided. The third one is to create many RC-QPs and DC-QPs on one rank and make them handle many sources when the pattern

[1] peer and rank are used interchangeably.

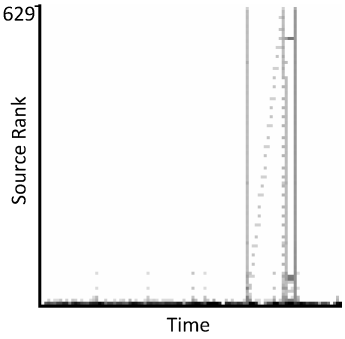

Figure 4: Source rank distribution over time observed from rank zero in DL-POLY with 630 processes.

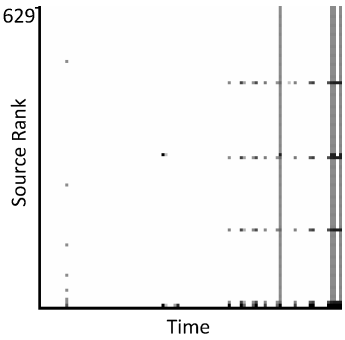

Figure 5: Source rank distribution over time observed from rank zero in NAMD with 630 processes.

(B) is happening. In this way, incoming messages are distributed over multiple destination QPs on the rank and the performance degradation issue of DC is avoided.

Fig. 4 shows the source rank distribution over time observed from rank zero in DL-POLY [10] using 10 processes $\times$ 63 nodes. We can see a new pattern, i.e., a short series of vertical dots in which the 2^i-th ranks are involved. This appears to be the result of an MPI collective function. It is not a problematic communication pattern because the number of source ranks is usually the order of logarithm of the number of processes. It is beneficial to use RC-QPs when the message count is large. The application shows the new pattern, that is, the diagonal dots. It might be the pattern (A) or (B) or (C) if many source ranks are involved. One of the three strategies mentioned previously can be used. Selecting RC-QPs would be beneficial when the number of source ranks is small and the overhead of preparing RC-QPs and RDMA buffer is smaller than the benefit which is determined by the message counts.

Fig. 5 shows the source rank distribution over time observed from rank zero in NAMD [8] using 10 processes $\times$ 63 nodes. We can see more clear case of the 2^i-th ranks pattern. We can see the new pattern, multiple horizontal dotted lines. It might be the pattern (A) or (B) or (C) if many source ranks are involved and thus one of the three strategies mentioned previously can be used.

3. DESIGN

The mechanism which selects a proper mapping of IB transport services consists of the following four components. (1) Software message channels which uses the transport services to provide abstracted messaging mechanisms. (2) Finding the best mapping of software message channels given a historical time-window. (3) Establishing software message channels to instantiate the mapping. (4) Evaluating different mappings using different time-windows to adapt the change in communication pattern.

Table 1: IB transport service and send method used for different MPI protocols with different types of software message channel

Software message channel	Eager Protocol	Rendezvous Protocol	
DC-sendrecv	DC, sendrecv	**Control Message**	DC, sendrecv
		Data transfer	DC, RDMA
DC-RDMA	DC, RDMA	DC, RDMA	
RC-RDMA	RC, RDMA	RC, RDMA	

3.1 Software Message Channels

We construct the following three software message channels for the MPI library on top of IB transport services.

DC-sendrecv.

A DC-sendrecv channel uses a pool of DC-QPs and performs communication using *sendrecv*. Multiple DC-QPs are used to alleviate the connection transaction overhead when sending to different ranks in a contiguous manner [9]. $((i+j)$ mod $n)$-th DC-QP is selected for the source rank of j and the destination rank of i (n denotes the number of DC-QPs). DC-sendrecv is slower than DC-RDMA and RC-RDMA.

DC-RDMA.

A DC-RDMA channel uses a pool of DC-QPs and performs communication using RDMA. Multiple DC-QPs are used for the same reason and in the same manner as DC-sendrecv. A ring-buffer for eager protocol is prepared and control messages are exchanged via a dedicated DC-sendrecv channel and the type of software message channel is agreed with the source rank when replacing a DC-sendrecv channel with a DC-RDMA channel Control messages are exchanged via the dedicated DC-sendrecv channel and the type of software message channel is agreed with the source rank when replacing a RC-RDMA channel with a DC-RDMA channel. DC-RDMA has shorter latency for sending message than DC-sendrecv[6].

RC-RDMA.

An RC-RDMA channel uses a RC-QP and performs communication using RDMA. A control message exchange similar in the DC-RDMA case is performed when replacing DC-sendrecv with a RC-RDMA or replacing DC-RDMA with a RC-RDMA. It provides the shorter latency than DC-RDMA[9].

The use-scenario of the software message channels is as follows. An MPI process prepares one DC-sendrecv for data transfer, one DC-sendrecv for sending credit messages, one DC-sendrecv for making source ranks prepare DC-RDMA or RC-RDMA channels at the startup time and all the communication uses the first DC-sendrecv. And then the MPI

process calculates the best mapping of software message channels to pairs of source rank and destination rank. And then it prepares DC-RDMAs or RC-RDMAs and replaces channels with the DC-RDMAs or RC-RDMAs for the source ranks according to the mapping. Table 1 describes which IB transport service and send method is used for a rank to perform different protocols, i.e., eager protocol or rendezvous protocols, with different software message channels. Eager protocol over DC-sendrecv channel uses DC and *sendrecv* and rendezvous protocol over DC-sendrecv channel uses DC and *sendrecv* for control messages and DC and RDMA for data transfer. Both eager and rendezvous protocol over RC-RDMA use RC and RDMA. Both eager and rendezvous protocol over DC-RDMA use DC and RDMA.

3.2 Finding the Best Mapping

Finding the best mapping is performed by solving a combinatorial optimization problem which fulfills the following targets.

- Allocate distinct RC-RDMA or DC-RDMA channels for destination ranks to alleviate the issue when there are many destination ranks.
- Allocate distinct RC-RDMA or DC-RDMA channels for source ranks on an MPI-process to alleviate the issue when there are many source ranks.
- Offload source ranks to DC-RDMA channels to limit the number of active RC-QPs to prevent the problem of on-HCA QP-cache slashing.
- Use as many RC-RDMAs as possible to make the most of the fastest RC-RDMA.
- Limit the number of RC-RDMA channels to limit the memory consumption.

Fig. 6 shows the algorithm. It is also used to calculate the latency reduction amount when using the mapping. The latency reduction amount is calculated relative to the case where using only DC-sendrecv. It calculates the latency reduction amounts when giving RC-RDMA channels to m source ranks and DC-RDMA channels to the following $l-m$ source ranks with different l and m values to find the best value of l and m by the doubly nested loop. RC-RDMA channels previously allocated are not destroyed when it is decided that they are not needed anymore. The total number of RC-RDMA channels is limited by changing the value of h according to the number of RC-RDMA channels created so far. The calculation of the overheads mentioned in the algorithm is explained in Section A.

3.3 Establishing Software Message Channels

The component establishes RC-RDMA or DC-RDMA channels by the following steps. One thread which runs in the background of the application thread is prepared to handle these steps.

J1 The initiator suspends its MPI send-command processing for the pair of source rank and destination rank, prepares a ring-buffer for eager protocol, send a control message to the responder and creates RC-QP when allocating an RC-RDMA channel.

```
FIND_MAP(W, S)
Input:
  W: A historical time-window
  S: Communication statistics of W
     which are the number of receives for source ranks.
     The source ranks are sorted in the descending order of
     number of receives.
Output:
  Mapping of DC-RDMAs or RC-RDMAs to source ranks
  and the amount of latency reduction when compared to
  using DC-sendrecv

Definitions:
  n_i: The number of receives from the i-th source rank
  d (r): Amount of the latency reduction for receiving messages
    when replacing DC-sendrecv with DC-RDMA (RC-RDMA)
  c: Overhead on the receiver side by sending one credit message
  s: Number of entries of each ring-buffer
  δ (ρ): Overhead for control message exchange to establish
    a DC-RDMA (RC-RDMA) channel
  p: Latency for one polling on one ring-buffer entry which fails
    to detect a message arrival
  h: Number of RC-QPs available
  τ: Max number of RC-RDMA or DC-RDMA channels

/* Calculate the latency reductions when giving RC-RDMA
  or DC-RDMA channels to l source ranks */
for l in {0, 1, 2, ..., τ} do
  /* Calculate the latency reductions when giving RC-RDMA
    channels to m source ranks and DC-RDMA channels to
    the following l − m source ranks */
  for m in {0, 1, 2, ..., h} do
    /* Latency reduction by replacing DC-sendrecv
      channels with RC-RDMA or DC-RDMA channels */
    x_{l,m} = Σ_{i=1}^{m} n_i r + Σ_{i=m+1}^{l} n_i d
    /* Overhead of sending credit messages */
    x_{l,m} = x_{l,m} − Σ_{i=1}^{l} n_i c/s
    /* Overhead of performing polling */
    x_{l,m} = x_{l,m} − (max_i n_i) l/2 p
    /* Overhead of control message exchanges to establish
      RC-RDMA or DC-RDMA channels */
    x_{l,m} = x_{l,m} − (mρ + (l − m)δ)
  end for
end for
Pick up the l and m value which give the maximum x_{l,m}
  and return the corresponding mapping of RC-RDMA and DC-RDMA
  channels to the source ranks.
```

Figure 6: Algorithm to find the best mapping of software message channels and calculates its latency

J2 The responder records the remote address of the ring-buffer and jumps to the step J4 when allocating a DC-RDMA channel. Otherwise, the responder creates RC-QP and responds to the initiator.

J3 The initiator makes the RC-QP transition to ready-to-send state and responds to the responder.

J4 The responder makes the RC-QP transition to ready-to-send state when allocating an RC-RDMA channel, sends the id (memory location for DC-RDMA or RC-RDMA, sequence number for DC-sendrecv) of the last message sent via the old channel and resumes its send-command processing.

J5 The initiator waits until the last message arrives to make sure that the arrival of the first message sent through the new channel is not detected before the arrival of the last message sent through the old chan-

nel. And then it starts polling for the new channel and resumes its MPI send-command processing.

3.4 Evaluating Different Mappings

This component first calculates the latency reduction amounts for different historical time-windows by using the component of finding the best mapping. Communication statistics for the time-windows are gathered at run-time and given to the component. And then it decides whether or not it is beneficial to move from the current mapping to the new one. It is assumed that the latency reduction amount calculated by using the historical time-window well predicts the latency reduction amount for the future time-window. And then it instructs the component of establishing software message channels to apply the mapping when it has decided to move to the new mapping. One thread running in the background of the application thread is used for the component. It is woken up periodically and the period is in the order of one second.

Let us call a part of execution time of an application a time-window. Let us denote by u the unit time and t the current time. Let us denote by $[b, e)$ a time-window begins at b and ends at e. The component uses the following steps.

T1 Find the best channel mapping for the first time-window $[0, u)$ and calculate the latency reduction amount by using the component of finding the best mapping, when one unit of time elapsed (let us denote by s_{cur} the mapping). Set the time-window as the current time-window (Let us denote by β and ϵ the start time and the end time of the current time-window, respectively). Record the latency reduction amount for the channel mapping (call it $b_{\mathrm{cur,cur}}$). Instruct the component of establishing message channels to apply the channel mapping.

T2 Find the best channel mapping for the extended time-window $[\beta, t)$ (let us denote by w_{ext} the time-window and let us denote by s_{ext} the mapping) and record the latency reduction amount (let us denote it by $b_{\mathrm{ext,ext}}$). Evaluate the latency reduction when s_{cur} is applied to $[\beta, t)$ and record it (let us denote it by $b_{\mathrm{cur,ext}}$). Note that the latency reduction amount does not include the overhead of establishing new channels because we don't need to perform them.

T3 Find the best channel mapping for the shrunk time-window $[t - \frac{\epsilon-\beta}{2}, t)$ and record the latency reduction amount (let us denote by w_{shr} the time-window and by $b_{\mathrm{shr,shr}}$ the latency reduction amount). Calculate the latency reduction when s_{cur} is applied to w_{shr} and record it (let us denote by $b_{\mathrm{cur,shr}}$ the latency reduction amount). Note that the latency reduction does not include the overhead of establishing new channels because we don't need to perform them.

T4 Let $(b_{\mathrm{ext,ext}} - b_{\mathrm{cur,ext}})$ divided by the number of messages received in the time-window be g_{ext} and $(b_{\mathrm{shr,shr}} - b_{\mathrm{cur,shr}})$ divided by the number of messages received in the time-window be g_{shr}. Select the channel mapping with largest g as a new channel mapping if $g > \gamma$ (γ is a parameter).

There is an additional issue for this component, i.e., creating and destroying RC-QP takes thousands of microseconds while the latency reduction of one message send is in the order of 0.1 microsecond. We prepare a pool of π RC-QPs to deal with it (π is a parameter). The component keeps the created RC-QP instead of destroying it when it is decided that an RC-RDMA is assigned to a source rank and then the RC-RDMA channel is no longer needed to the source rank. A RC-QP is reused without the creation/destruction cost when it is decided that an RC-RDMA is assigned to the same source rank again. ρ is set to the smaller value than the measured overhead to reflect the expected reuse counts.

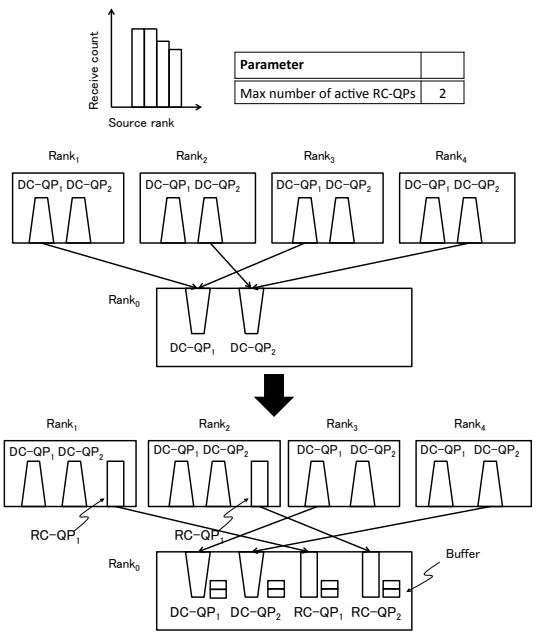

Figure 7: Example of the selection of software message channels.

Fig. 7 illustrates the steps. All ranks start with a DC-sendrecv channels with two DC-QPs. And then rank zero tries to find the best mapping, i.e., software message channel type for Rank one, Rank two, Rank three and Rank four using the statistics. It calculates latency reduction amount for possible mappings and compare the amounts and find the best. In this example, RC-RDMA for Rank one, RC-RDMA for Rank two, DC-RDMA for Rank three, DC-RDMA for Rank four is the best mapping. This is because the maximum number of RC-QPs is two and Rank one and two has the largest receive counts and therefore the latency reduction amount is maximized when allocating fastest RC-RDMAs to them and offloading the receive handling for Rank three and four to the DC-RDMA. Then the rank prepares two RC-QPs and buffers for them and instructs Rank one and Rank two to prepare RC-QPs and then the rank prepares buffers for the two DC-RDMA channels. Note that the mechanism changes the software message channel for receiving but does not change the channel for sending, e.g., it is not decided that Rank zero will use RC-RDMA when sending a message to Rank one.

4. EVALUATION

We compare the memory consumption of the MPI library with our method, called Adaptive Queue Pair Selection (AQPS), against that with existing methods and then compare the

execution times of microbenchmarks using AQPS against existing methods.

4.1 Methodology

The following three channel selection methods including AQPS are implemented. The first two are considered as the existing methods.

DC-RDMA-OD.

This method starts from a DC-sendrecv channel with 28 DC-QPs for sending messages. It uses another DC-sendrecv channel with two DC-QPs for credit messages. It uses yet another DC-sendrecv channel with two DC-QPs for establishing DC-RDMA channel. A process sends a control message to a destination peer via the channel to request a DC-RDMA channel and ring-buffer of 16 entries when it sends the first message to the peer. It uses a DC-RDMA channel for the destination peer after that.

RC-RDMA-OD.

This method starts from a DC-sendrecv channel with four DC-QPs for sending messages. It uses another DC-sendrecv channel with two DC-QPs for credit messages. It uses yet another DC-sendrecv channel with two DC-QPs for establishing RC-RDMA channel. A process sends a control message to a destination peer via the channel to request an RC-RDMA channel and a ring-buffer of 16 entries when it sends the first message to the peer. It uses a RC-RDMA channel for the destination peer after that. It creates up to as many RC-QPs as the number of source peers.

AQPS.

This method starts from a DC-sendrecv channel with four DC-QPs for sending messages. It uses another DC-sendrecv channel with two DC-QPs for credit messages. It uses yet another DC-sendrecv channel with two DC-QPs for establishing RC-RDMA or DC-RDMA channel. DC-RDMA channels with a ring-buffer of 16 entries or RC-RDMA channels with a ring-buffer of the same size are selected to source ranks according to the communication statistics. The parameters for the selection are shown in Table 2. These values are obtained through measurement or estimation. The selection decision is performed around every one second. This value is chosen to make AQPS perform the first selection fast enough and detect the following changes in communication pattern frequently enough while keeping the selection overhead low by observing communication traces of applications.

We implemented these methods by modifying the network device module of MPICH-3.1 [3]. That is, the proposed methods are implemented as a part of a communication library we have been developing, called Low-Level Communication library (LLC), and then the network device module which uses LLC as a library is developed and then the module is plugged into MPICH. LLC is compiled with Intel C Compiler version 15.0.3 20150407 with the option of `-O2`. MPICH is compiled with Intel C Compiler version 15.0.3 20150407 with the `configure` option of `-enable-fast=O2, nochkmsg, notiming, ndebug -enable-nemesis-dbg-nolocal`. The last option makes processes residing in the same node use IB, not shared memory. We used a cluster computer with the specifications listed in Table 3. The configuration of one MPI process per node is used because we assume that many applications use OpenMP plus MPI hybrid programming.

Table 3: Parameters for evaluation environment

Component	Parameters
Node processor	Intel Xeon E5-2680 v2, 2.801 GHz, 10-physical core, 20-logical core, 2-socket
HCA	Mellanox Connect-IB, 6.79 GB/s
I/O bus	PCI Express 3.0, 16-lane, 15.75 GB/s
Number of nodes	64

Table 4: Memory consumption components of different methods of software channel selection. "Common" part is included in the all methods.

Channel selection method	Type of software message channel	Purpose	Mem. cons. by QPs	Mem. cons. per one read command	Mem. cons. by one ring-buffer
(Common)	DC-sendrecv	Credit message	296KB	256B	-
	DC-sendrecv	Establishing channels	512KB	256B	-
DC-RDMA-OD	DC-sendrev / DC-RDMA	Data transfer	3464KB	4096B	64KB
RC-RDMA-OD	DC-sendrecv / DC-RDMA	Data transfer	1340KB	4096B	-
	RC-RDMA	Data transfer	1424KB	-	64KB
AQPS	DC-sendrecv / DC-RDMA	Data transfer	1340KB	4096B	64KB
	RC-RDMA	Data transfer	1424KB	-	64KB

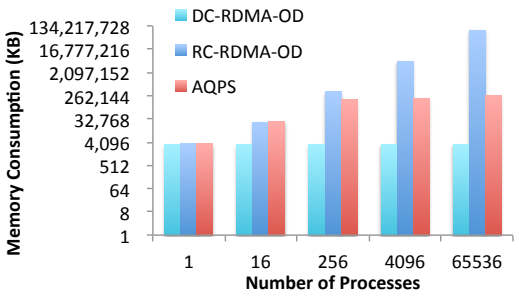

Figure 8: Memory consumption of the channel selection methods (excluding buffers) with different numbers of processes.

4.2 Memory Consumption

Memory consumption of a method of software channel selection mainly comes from its QPs and buffers. Table 4 describes the memory consumption components of the different methods. AQPS maintains a table of communication statistics with 2048 entries and the size of one entry is 8 byte times number of processes in addition to QPs and buffers.

Fig. 8 shows the memory consumption of the channel selection methods, including QPs and the statistics table but excluding buffers. Let us denote by n the number of processes. The values include QP but excludes buffers used by DC-sendrecv or RC-RDMA or DC-RDMA. The memory

Table 2: AQPS parameters used for evaluation. n denotes the number of processes.

Parameter name	Description	Value
d	Latency difference when replacing DC-sendrecv with DC-RDMA	-0.1 usec
r	Latency difference when replacing DC-sendrecv with RC-RDMA	-0.15 usec
p	Overhead for one failed polling	0.01 usec
c	Overhead for sending one credit packet	0.3 usec
γ	Threshold value of per message latency reduction	0.025 usec
δ	Overhead for establishing a DC-RDMA channel	10 usec
ρ	Overhead for establishing an RC-RDMA channel	10 usec
h	Max number of active RC-QPs	26
τ	Max number of software message channels allocated at one time	256
π	Size of RC-QP pool	128
σ	Size of communication statistics table	2048 ($n \leq 2^{10}$) $2048/((\log_2 n - 10) * 2)$ ($n > 2^{10}$)

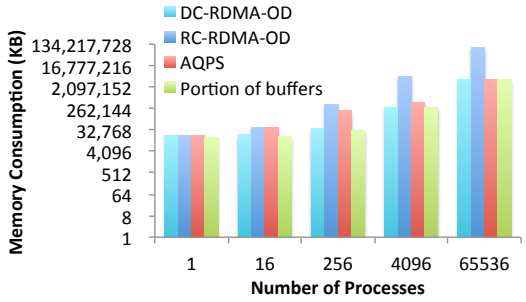

Figure 9: Memory consumption of the channel selection methods (including buffers) with different numbers of processes.

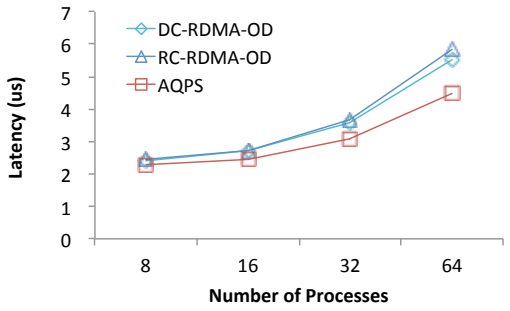

Figure 10: Communication latency of one send with different number of processes in *onetoall*.

consumption of DC-RDMA is a scalable amount and do not increase with n because they do not add QPs. The memory consumption of RC-RDMA is not scalable amount because it grows with n and reaches 93 GB with 2^{16} processes because it adds RC-QPs. Only the statistics history table part of the memory consumption of AQPS grows linearly with n because it stops creating RC-QPs after creating 128 of them. The pace can be decreased by reducing the number of the table entries by $(\log_2 n - 10) * 2$ when n is greater than 2^{10}. This optimization reduces the memory consumption of the table to 87 MB with 2^{16} processes and reduces the memory consumption of AQPS 273 MB with 2^{16} processes. Therefore, it can be said that the memory consumption of AQPS is scalable.

Fig. 9 shows the memory consumption of the channel selection methods, including QPs, the statistics table and buffers. It also shows the memory consumption of buffers which is common to all methods. The memory consumption of all the methods are not scalable when including the buffers because a part of the memory consumption of the buffers grows linearly with n. There are methods to reduce the buffers, for example, limiting the number of message channels using RDMA (i.e. RC-RDMA and DC-RDMA) by only satisfying first N allocation requests (N is a parameter).

4.3 Microbenchmarks

The effectiveness of AQPS is evaluated by using three micro-benchmarks representing problematic communication patterns and changing communication patterns. The first one corresponds to the communication pattern where one process sends to all the other processes. (call it *onetoall*). The size of messages is one byte and the number of messages sent is set to a multiple of 320000. Barriers are performed when a process receives more than 128 messages after a series of $N-1$ sends to avoid too much contention (N denotes the number of processes). The second one corresponds to the communication pattern where $N-1$ processes sends to one process (call it *alltoone*). The size of messages is one byte and the number of messages sent is set to a multiple of 48000. Barriers are performed when a process receives more than 128 messages after a series of $N-1$ sends to avoid too much contention. These communication patterns seem extreme and rare but they are found in real applications as we see in Section 2.2. The third one repeats *onetoall* and *alltoone* for five times (call it *repeat*). This program tests the capability of AQPS to adapt the change of communication patterns.

Fig. 10 shows the result of *onetoall*. The Y-axis shows the latency per one send. AQPS achieves the latency reduction of 19% with 64 processes when compared to DC-RDMA-OD. DC-RDMA-OD can alleviate the performance degradation issue when a source rank sends messages to different destination ranks in a short period of time because the latency for waiting for the acknowledge packet can be hidden by preparing 28 DC-QPs for the different destination ranks in the source rank and using them in a round-robin manner. AQPS can alleviate this issue in a similar way by using 4 DC-QPs plus 26 RC-QPs. In addition, AQPS can re-

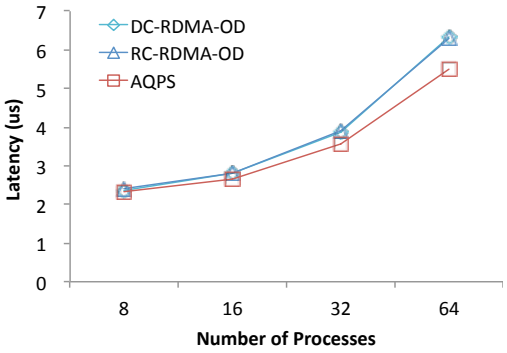

Figure 11: Communication latency of one send with different number of processes in *alltoone*.

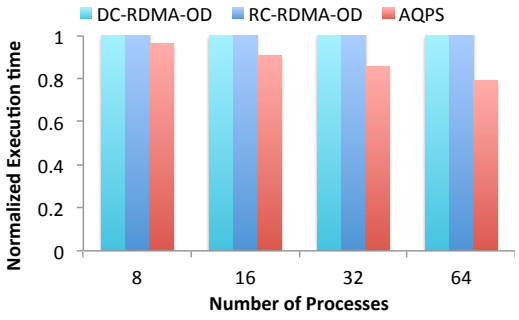

Figure 12: Execution time normalized to DC-RDMA with different number of processes in *repeat*.

duce both of the DC connection and disconnection overhead by preparing only 4 DC-QPs in the source rank and letting other messages handled by 26 RC-QPs. It is assumed that AQPS achieves shorter latency than DC-RDMA-OD because this additional effect is large since the overhead of the DC connection and disconnection packets occupying the network resources is large because the message size is small.

AQPS achieves the latency reduction of 24% with 64 processes when compared to RC-RDMA-OD. RC-RDMA-OD suffers from the performance degradation issue because there are more than 64 active QPs with 64 processes. On the other hand, AQPS limits the number of RC-QPs to 26 and DC-QPs to 8 to avoid the issue. In this way, AQPS achieves shorter latency than RC-RDMA-OD.

Fig. 11 shows the result of *alltoone*. The Y-axis shows the latency per one send. AQPS achieves the latency reduction of 13% with 64 processes when compared to DC-RDMA-OD. DC-RDMA-OD can alleviate the performance degradation issue when many source ranks send a message to the same destination rank because the contentions among the DC connection requests from the source ranks happen rarely since the rank can accept 28 DC connection requests at the same time by preparing 28 DC-QPs in the destination rank. AQPS can alleviate this issue in a similar way by using 4 DC-QPs and 26 RC-QPs. In addition, AQPS can reduce the DC connection overhead by preparing only 4 DC-QPs in the destination rank and letting other messages handled by 26 RC-QPs. It is assumed that AQPS achieves shorter latency than DC-RDMA-OD because of this additional effect.

AQPS achieves the latency reduction of 13% with 64 processes when compared to RC-RDMA-OD. RC-RDMA-OD suffers from the performance degradation issue because there are more than 64 active QPs with 64 processes. On the other hand, AQPS limits the number of RC-QPs to 26 and DC-QPs to 8 to avoid the issue. In this way, AQPS achieves shorter latency than RC-RDMA-OD.

Fig. 12 shows the result of *repeat*. The Y-axis shows the execution time normalized to that of DC-RDMA-OD. AQPS achieves the execution time reduction of 21% when compared to DC-RDMA-OD and of 24% when compared to RC-RDMA-OD. This is achieved because AQPS periodically checks the changes in communication pattern to adapt if there is a change.

Overall, we can see that the overhead of evaluating diffrent mappings and gathering communication statistics is outweighed by the latency benefit.

5. RELATED WORK

Hari Subramoni et al. proposed a variant of DC-RDMA [5]. A pool of DCs are prepared and they are assigned to pairs of source peer and destination peer in a round-robin fashion so that the consecutive sends to different destination peers will not use the same DC to alleviate the performance degradation issue which is described in Section 3.1. On the other hand, AQPS finds a good mapping of RC-QPs and DC-QPs to pairs of source peer and destination peer using statistics. It can offload some of the destination peers to RC-QPs to deal with the issue when facing that kind of situation. In addition, it tries to make the most of RC-QPs which are faster than DC-QPs.

Matthew J Koop et al. proposed a method to select IB transport service and its send methods dynamically [5]. RC-RDMA, a channel using RC and *sendrecv* commands (call it RC-sendrecv), a channel using UD and *sendrecv* (call it UD-sendrecv) are selected according to the message size. They allocate RC-RDMA or RC-sendrecv when the number of messages received for a pair of source peer and destination peer exceeds the threshold value. Weikuna Yu et al. proposed a method to connect and disconnect RCs [12]. Their method adds RC-RDMA channels on demand. Jiuxing Liu et al. proposed a method to allocate RC-RDMA in a first-come first-served manner [6]. First N connections are given RC-RDMA channels (N is a parameter). Abhinav Vishnu et al. proposed a method to allocate RC-RDMA using Least-Recently-Used (LRU) algorithm [11]. Every time a connection is requested and there are N RC-RDMA channels the LRU connection is torn down and a new connection is established (N is a parameter). These four methods cannot handle the issue of DC and RC. On the other hand, AQPS finds a good mapping of RCs and DCs to pairs of source peer and destination peer using statistics and it can handle the issues.

6. CONCLUSIONS

An MPI library using InfiniBand network can choose RC-QP pair or DC-QP pair to use for each source-destination pair. The amount of latency improvement by using RC-QP for DC-QP is usually positive, but it fluctuates and even becomes negative in some communication patterns. In addition, it is not allowed to use RC-QP for all source-destination pairs because RC-QPs consume prohibitive amount of memory with millions of MPI processes. Therefore, the MPI

library need to select mapping of RC-QPs and DC-QPs according to the current communication pattern. We proposed a method to find a proper mapping by comparing potential latency benefits for different mappings and different historical time-windows. Communication statistics for the time-windows are gathered at run-time and used for the comparison. Our method achieves 13%-19% latency reduction in microbenchmarks when compared to the method using only DC-QPs.

Future work includes adding software message channels using UD and XRC to the proposed method and evaluating it to find whether or not these channels provide additional benefit.

7. ACKNOWLEDGMENTS

This work is partially funded by MEXT's program for the Development and Improvement for the Next Generation Ultra High-Speed Computer System, under its Subsidies for Operating the Specific Advanced Large Research Facilities.

8. REFERENCES

[1] P. N. Brown, R. D. Falgout, and J. E. Jones. Semicoarsening Multigrid on Distributed Eemory Machines. *SIAM Journal on Scientific Computing*, 21(5):1823–1834, 2000.

[2] D. Crupnicoff, M. Kagan, A. Shahar, N. Bloch, and H. Chapman. Dynamically-Connected Transport Service, 7 2012. US Patent 8,213,315.

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

[4] InfiniBand Trade Association Std. InfiniBand™ Architecture Specification, Vol. 1, Rel. 1.2.1, 2007.

[5] M. J. Koop, T. Jones, and D. K. Panda. MVAPICH-Aptus: Scalable high-performance multi-transport MPI over InfiniBand. *In Proc. of IPDPS*, pages 1–12, 4 2008.

[6] J. Liu, J. Wu, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. *In Proc. of ICS'03*, pages 295–304, 2003.

[7] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, 2012. http://www.mpi-forum.org/docs/mpi-3.0/.

[8] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. C. R. D. Skeel, L. Kale, and K. Schulten. Scalable Molecular Dynamics with NAMD. *Journal of computational chemistry*, 26 (16):1781–1802, 2005.

[9] H. Subramoni, K. Hamidouche, A. Venkatesh, S. Chakraborty, , and D. K. Panda. Designing MPI Library with Dynamic Connected Transport (DCT) of InfiniBand: Early Experience. *In Proc. of ISC*, pages 278–295, 6 14.

[10] I. T. Todorov, W. Smith, K. Trachenko, and M. T. Dove. DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism. *Journal of Materials Chemistry*, 16 (20):1911–1918, 2006.

[11] A. Vishnu, M. Krishnan, and P. Balaji. Dynamic Time-Variant Connection Management for PGAS Models on InfiniBand. *In Proc. of CASS'11 (IPDPS Workshops)*, pages 740–746, 2011.

[12] W. Yu, Q. Gao, and D. K. Panda. Adaptive Connection Management for Scalable MPI over InfiniBand. *In Proc. of IPDPS'06*, pages 32–39, 2006.

APPENDIX

A. CALCULATION OF OVERHEAD

The detail of overhead calculation mentioned in 3.3 is explained.

A.1 Overhead of performing polling

A message arrival is detected by polling on the entry of a ring-buffer of a DC-RDMA or RC-RDMA channel. The entry is pointed by its consumer pointer. Therefore, the polling trials are performed on l entries. A message arrival detection for a source rank can be delayed because the message for the rank might arrive while performing the polling trials for the other ring-buffers. Let $i-1$ be i'. $\frac{i'}{l}$ portion of the messages for the i-th ring-buffer arrives before polling on the ring-buffer, which delays the detection by $\frac{i'}{2}p$. $\frac{l-i'}{l}$ portion of the messages arrives after polling on the ring-buffer and is detected in the next round of the polling series, which delays the detection by $(\frac{l-i'}{2}+i')p$. Therefore, the average delay to one ring-buffer for one message arrival is $\frac{l}{2}p$. The delay from the viewpoint of a program is the total delay to the ring-buffer with the most messages and estimated as $(\max_i n_i)\frac{l}{2}p$.

A.2 Overhead of sending credit messages

A credit message should be exchanged at least once every s message receives for DC-RDMA or RC-RDMA channels and therefore the overhead is calculated as $\sum_{i=1}^{l} n_i \frac{c}{s}$.

A.3 Overhead of establishing software message channels

Control message exchanges to establish m RC-RDMA channels costs $m\rho$ and control message exchanges to establish $l-m$ DC-RDMA channels costs $(l-m)\delta$ and thus the overhead is calculated as $m\rho + (l-m)\delta$.