Using Connection-Oriented vs. Connection-Less Transport for Performance and Scalability of Collective and One-sided operations: Trade-offs and Impact

AMITH R MAMIDALA, SUNDEEP NARRAVULA, ABHINAV VISHNU, GOPAL SANTHANARAMAN, DHABALESWAR K PANDA

Technical Report
OSU-CISRC-11/06-TR77

Using Connection-Oriented vs. Connection-Less Transport for Performance and Scalability of Collective and One-sided operations: Trade-offs and Impact *

Amith R. Mamidala Sundeep Narravula Abhinav Vishnu Gopal Santhanaraman
Dhabaleswar K. Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{mamidala, narravul, vishnu, santhana, panda}@cse.ohio-state.edu

Abstract

Communication subsystem plays a pivotal role in achieving scalable performance in clusters. The communication semantics employed are dictated by the programming model used by the application such as MPI, UPC, etc. Out of the gamut of communication primitives, collective and one-sided operations are especially significant and have to be designed harnessing the capabilities and features exposed by the underlying networks. In some cases, there is a direct match between the semantics of the operations and the underlying network primitives. InfiniBand provides two transport modes: (i) Connection-oriented Reliable connection (RC) *supporting Memory and Channel semantics and (ii) Connection-less* Unreliable Datagram (UD) *supporting Channel semantics. Achieving good performance and scalability needs careful analysis and design of communication primitives based on these options.*

In this paper, we evaluate the scalability and performance trade-offs between RC and UD transport modes. We study the semantic advantages of mapping collective and one-sided operations on to memory and channel semantics of IBA. We take AlltoAll as a case study to demonstrate the benefits of RDMA over Send/Recv and to show the performance/memory trade-offs over IB transports. Our experimental results show that UD-based AlltoAll performs 38% better than Bruck's for short messages and upto two times better than the direct AlltoAll over RC. Since IBA does not provide RDMA over UD in hardware, we emulate the same in our study. Our results show a performance dip of up to a factor of three for emulated RDMA Read latency as compared to RC, highlighting the need for hardware based RDMA operations over UD.

*This research is supported in part by DOE grant #DE-FC02-01ER25506, NSF Grants #CNS-0403342 and #CNS-0509452; grants from Intel, Sun MicroSystems, Cisco Systems, and Linux Networks; and equipment donations from Intel, AMD, Apple, IBM, Microway, PathScale, SilverStorm and Sun MircroSystems.

1 Introduction

Clusters built from commodity PCs are being predominantly used as main stream tools for high-end computing owing to their cost-effectiveness and easy availability. In fact, the top 500 list of supercomputers feature large scale clusters delivering TFlops of computational power. One of the primary challenges faced by such massive cluster installations is that of scalability, encompassing both the performance and resource requirements of the system architecture. Among the different components of the cluster architecture, the communication subsystem is particularly important as it plays a pivotal role in achieving overall scalability of the system and enabling higher parallel speed-ups of large scale scientific and engineering applications.

On the other hand, InfiniBand Architecture (IBA) [4] is a new industry proposed standard and is making headway in the high performance networking domain. In addition to delivering low latencies and high bandwidth, its rich set of network primitives can be leveraged by the upper layer communication primitives for achieving high performance and scalability. InfiniBand allows for four conduits of message transport, Reliable Connection (RC), Unreliable Connection, Reliable Datagram (RD) and Unreliable Datagram (UD). Further, IBA specifies memory semantics for communication through Remote Direct Memory Operations (RDMA) and channel semantics through Send/Recv. In current IBA specification [4], RC and RD support both RDMA and Send/Recv while UD supports only Send/Recv.

While raw performance is provided by IBA, leveraging the same and scaling the performance for large clusters requires critical analysis of available options and issues involved. In particular, programming models need

the support of highly efficient collective and one-sided operations [7, 5]. We use *AlltoAll* as a case study for collective operations to evaluate the impact of choosing different communication semantics and trade-offs in their implementations over the IBA transport models (UD and RC) described above. Additionally, we study these trade-offs in the context of one-sided operations. In our evaluation, we clearly observe the advantages of utilizing memory (RDMA) semantics and UD based transport protocols. Since, currently IBA does not support RDMA over UD, we emulate RDMA over UD transport in our study to understand its benefits. Please note that though IBA provides RDMA over RD, it is not implemented by any of the known IBA vendors because of its well known performance limitation of supporting only one message in flight between two end points.

By our study, we underscore the importance of using both the memory semantics and connection-less transports of IBA to achieve good scalability in terms of both performance and resource usage for collective and one-sided communication primitives. Specifically, we aim to achieve the following objectives:

- Analyze the impact of using memory semantics vis-a-vis channel semantics for collectives like AlltoAll in modern high bandwidth networks.
- Demonstrate why currently InfiniBand with the lack of UD based RDMA, does not scale well wrt memory for one-sided operations used by different programming layers.
- Analyse the effectiveness or lack thereof of emulated RDMA over UD in scaling the perfromance the communication primitives.
- Show the trade-offs between the performance and memory requirements in the context of collective operations, in particular for AlltoAll operations.
- Show that increased number of connections for the connection-oriented transports can lead to cache and resource constraints on the NIC leading to decreased performance.

We have implemented our designs and integrated them into MVAPICH [6] which is a popular MPI implementation used by more than 400 organizations world-wide. Our experimental results show that the UD-based AlltoAll performs 38% better than bruck's for short messages and upto two times better than the direct AlltoAll over RC. Further, our results show performance degradation up to a factor of three for emulated RDMA Read latency of one MTU compared to RC. This demonstrates the need to have hardware based RDMA operation over UD for good performance and scalability in next generation clusters.

The rest of the paper is organized in the following way. In Section 2, we provide an overview of the InfiniBand Architecture. In Section 3, we explain the motivation for our work. In Section 4, we discuss detailed design issues. We evaluate our designs in Section 5 and talk about the related work in Section 6. Conclusions and Future work are presented in Section 7.

2 Background

In this section, we present a brief overview on the InfiniBand Architecture and collective communication.

InfiniBand Architecture Overview: InfiniBand Architecture (IBA) [4] is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. A typical IBA cluster consists of switched serial links for interconnecting both the processing nodes and the I/O nodes. The IBA specification defines a communication and management infrastructure for inter-processor communication. InfiniBand supports different classes of transport services. In current products, Reliable Connection (RC) service Unreliable Datagram (UD) service and Unreliable Connection (UC) are supported.

Memory Semantics "vs" Channel Semantics: IBA supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and Memory Semantics (RDMA communication model).

In channel semantics, each send request has a corresponding receive request at the remote end. Thus there is a one-to-one correspondence between every send and receive operation. Receive operations require buffers posted on each of the communicating QP, which amount to a large number. In order to allow sharing of communication buffers, IBA allows the use of Shared Receive Queues (SRQ). In which multiple QPs have a common Receive Queue.

In memory semantics, Remote Direct Memory Access (RDMA) operations are used. These operations do not require a receive descriptor at the remote end and are transparent to it. For RDMA, the send request itself contains the virtual addresses for both the local transmit buffer and the receive buffer on the remote end. The RDMA operations are available with the RC Transport. IBA also defines an additional operation: RDMA write with immediate data. In this operation, a sender can send a limited amount of immediate data alone with a regular RDMA operation.

Figure 1 shows the basic working of both the RDMA and the Send/Recv models. The main steps involved are labeled with sequence numbers. The main difference between the two is the requirement of posting a receive descriptor for the send/recv model.

Connection-Oriented Reliable Connection (RC): IBA specifies the RC transport layer as a reliable communication layer. In this transport layer, both channel and memory semantics are supported. The communication packets are completely managed by the RC transport layer. Further, the RC transport layer guarantees reliability and in-

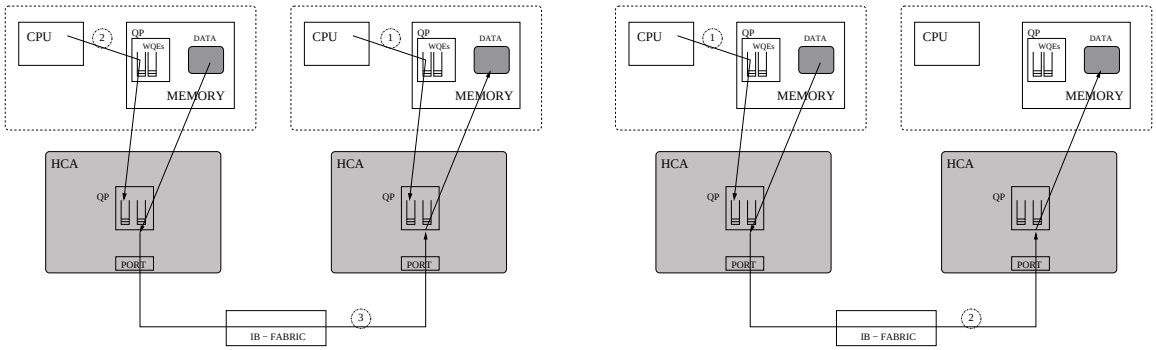

Figure 1. InfiniBand Transport Models (a) Send/Recv Model, (b) RDMA Model

order delivery of data.

Connection-Less Unreliable Datagram (UD): The UD Transport specified by IBA supports only the channel-based communication semantics. The basic communication is achieved by the UD layer by exchanging network MTU sized datagrams. These datagrams can be sent to a UD QP by any other UD QP in the network. An explicit connection between the two QP's in not needed. The reliability and order of delivery of these datagrams is not guaranteed by IBA and needs to be managed by the application.

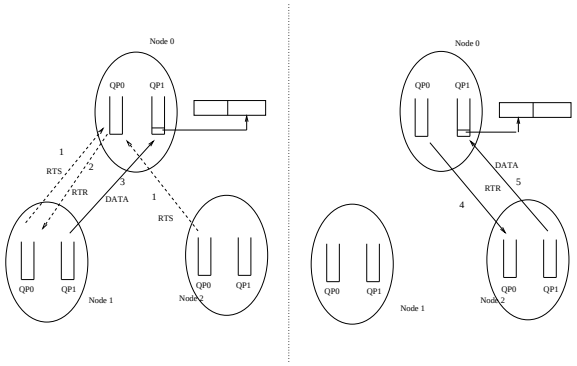

Figure 2. Zero Copy over Channel Semantics

Zero-Copy over Channel Semantics: A zero-copy protocol using channel semantics is outlined in Figure 2. The basic idea is to use a handshake to make sure that the remote side has preposted the descriptor pointing to destination buffer. This mechanism applies well when a single pair of process is communicating. However, if more than one process is sending messages to the same destination node, serialization of the messages occurs if a single receive queue is used for the incoming messages. This is the case with the Shared Receive Queue of RC. This is indicated in the Figure 2. Though the RTS messages have been concurrently issued by both the processes in the first step, the RTR message arrives only in the fourth step after the data delivery of the previous message is completed.

Caching connection information at NIC: For the connection-oriented transport, the NIC maintains a separate cache called as ICM to store the queue pair context and completion queue information. It also stores the latest address translation information of the buffers used. In the case of connection-less transport, the demand on the cache resources is much less as one queue pair is enough to communicate with any process in the network unlike the connection-oriented case which needs a QP for each pair of nodes.

Collective Algorithms: In this section we present the different algorithms used for MPI collectives. For very small messages, MPI_Alltoall uses the Bruck's Algorithm [2] which is a store and forward algorithm taking $log(n)$ steps. For small to medium messages, *Alltoall* uses a non-blocking algorithm [7] where each node sends messages to all other nodes. Messages are scattered so that not all messages are directed towards a node to avoid congestion and hot-spot effects. A simple rule could be a process sends the ith message to process whose rank is (my_rank+ i)$\%$ n. These are currently used in MPICH [3]. This algorithm works well for medium messages where the cost of store and forward is absent. For large messages, a pair-wise exchange algorithm is used which consists of $(n-1)$ steps. In step i, the process with rank r sends and receives data from process with rank r$\oplus$i [7]. *Barrier* is implemented using a dissemination based algorithm which consists of $log(n)$ steps [8]. For *Allgather* small messages, a recursive doubling algorithm again consisting of $log(n)$ steps is employed.

3 Motivation

As clusters increase in size, the performance and scalability of the communication subsystem becomes the key requirement for achieving overall scalability of the system. In this context, the efficiency of collective and one-sided operations is especially important as they are the widely used communication operations in different programming models like UPC, MPI-2, etc. The underlying cluster interconnect plays a crucial role in determining the performance achievable for a given communication operation. InfiniBand, which is a leading high performance

interconnect offers several features and capabilities. Out of these the prominent ones are the support for memory semantics in the form of RDMA operations together with the standard Send/Recv channel semantics. Also, it allows for two communication modes of transport, Reliable Connection (RC) which is connection-oriented and Unreliable Datagram (UD) which is connection-less. RDMA operations are supported over RC transport but lack in UD. Thus, one way to support RDMA over UD is to emulate them over Send/Recv as illustrated in Figure 3.

As shown in Figure 3 there exist different design choices for implementing collective and one-sided operations. Mainly, these can be classified into choosing RDMA vs Send/Recv. The primary motivation of choosing one or the other is dependent on to what extent these semantics match with the upper application layer communication requirements. In addition, with each of the semantics, once again there are two options of choosing the underlying transport implementing the semantics. Thus, it becomes necessary to evaluate the trade-offs in choosing either of the two transports. However, it is to be noted that currently IBA does not support memory semantics over UD. We consider the above mentioned issues one after another in the subsequent sections.

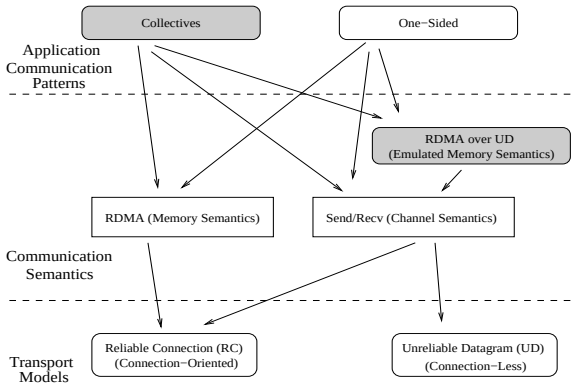

Figure 3. Broad Overview

3.1 RDMA "vs" Send/Recv

In this section we explain why RDMA is better than Send/Recv for certain collectives and one-sided operations.

High-Concurrency collective operations: For maximizing overall concurrency in modern high bandwidth networks, collective operations like *AlltoAll* use the direct algorithm for short and medium sized messages. As explained in Section 2, in this algorithm, each process issues non-blocking sends to all other processes in the group. The current practice of implementing this algorithm is using copy-based approaches.

Though this algorithm is used for small to medium messages, copy costs become dominant as the system size increases. This is because in AlltoAll, the total size of the messages copied increases linearly with the number of processes involved. Using RDMA presents a potential design alternative for implementing High-Concurrency *AlltoAll* with zero-copy. Also, the semantics of the direct algorithm match well with that of RDMA with each process writing data into other processes' buffer. It is to be emphasized that zero-copy with concurrency is possible with RDMA as the ordering and placement of messages is automatically taken care of by the underlying layer. Achieving zero-copy for concurrent network operations over Send/Recv can be inefficient and needs to be investigated.

One-sided Operations: One-sided operations benefit directly from the native RDMA support of the underlying network fabric. The main benefits of offloading RDMA to the network processor are zero-copy capabilities on pinning based networks and no cache pollution. The latter is a significant factor to be considered for true overlap of computation and communication. RDMA also is a good fit for the languages like UPC, etc. which are known to be suited to unstructured computations and irregular communication patterns [1].

3.2 Comparison of IBA Transports, RC and UD

Performance "vs" Resource Scalability: Connection-oriented protocols offer better latency and bandwidth capabilities as the fragmentation and assembly of messages is taken care of by hardware. Connection-less protocols on the other hand do not have these advantages as they transfer one MTU of data at a time. Due to lack of network level packet ordering in connection-less protocols, they have the drawback of requiring packet buffering at the receiver to handle out-of-order packets and hence restrict zero-copy data movement over UD.

RC also poses its own suite of problems. The number of connections needed for RC transport grows quadratically with the number of communicating processes limiting the scalability as the resource usage grows linearly with increasing number of point-to-point channels. This problem can be alleviated to certain extent with the use of a single reception point for the incoming messages (SRQ).

In light of the above mentioned discussions, it is necessary to understand the implications of these in detail both in terms of performance and resource usage.

NIC level cache and resource management: As mentioned in the earlier section, each QP has certain resource requirement on the IB HCA. Based on its own constraints, the HCA caches the information related to a certain number of QP's in its cache for faster processing. This method of operation of an IB HCA uniquely impacts the performance of a connection-oriented design, leading to a basic performance trade-off.

While the connection-less designs need very few QPs (typically 1 QP) for its communication needs, the connection-oriented designs need a large number of these

(*order (n)*, where n is the system size). Though the individual RC QPs can provide better performance than UD based communications, having more than a few RC QPs can lead to resource constraints at the IB HCA and the performance achieved by the RC QPs' can significantly drop. This trade-off imposes a significant limitation on the system size that a RC QP based design can efficiently connect. This trade-off need to be evaluated and analyzed quantitatively.

3.3 Emulating RDMA over UD

As seen from the earlier section 3.1one-sided operations benefit greatly utilizing the RDMA semantics exposed by the underlying layer. Also, as mentioned in section 3.2 the scalability and performance of these RDMA operations is directly dependent on the underlying transport mode used. RC provides memory semantics in the form of RDMA but as shown in section 3.2 it has several scalability limitations. UD on the other hand is scalable but it lacks memory semantics. RDMA over UD is desired in order to support scalable one-sided operations.

Efficient support for scalable one-sided operations is required in many scenarios. Studies from Yelick et al have demonstrated the scaling of UPC to a large number of nodes [1]. In one of their papers, they "break-down" the exchange phase of the AlltoAll to achieve overlap of computation and communication where possible. In their approach, a given node communicates directly with every node in the system using one-sided operations. Since current connection-less UD transport does not provide RDMA operations, in light of the above mentioned performance benefits of RDMA and scalability benefits of UD based transports, the possibility of emulating RDMA like operations over UD presents an important scenario. The performance implications of the emulation need to be carefully studied to understand the utility of RDMA-like operations over UD.

As can be seen there exist multiple design decisions which need careful evaluation for in-depth understanding of the trade-offs. In this paper, we aim to accomplish the following:

(i) Understand the benefits of RDMA vs Send/Recv for High Concurrency AlltoAll operation (ii) Understand the Performance and Resource Scalability trade-offs for RC vs UD both at Micro-Benchmark and Benchmark Level (iii) Explore RDMA over UD emulation and understand its implications.

4 Design and Implementation

In this section, we present the design choices and our designs of RDMA *AlltoAll*, RDMA over UD and UD-based *AlltoAll*.

4.1 High Concurrency RDMA-based *AlltoAll*

AlltoAll, as described in Section 2 uses a direct algorithm where each process issues simultaneous non-blocking send operations to all the other processes. The current implementations use copy based approaches over Send/Recv for implementing these. These copy costs become significant as the system size increases. Thus, we need zero-copy approaches to remove this overhead. There are two alternatives of achieving zero-copy: (i) using Send/Recv and (ii) using RDMA.

Using Send/Recv semantics for high-concurrency zero-copy operations has a fundamental drawback. As illustrated in Section 2, achieving zero-copy over Send/Recv leads to serialization of network transactions and our results indicate that copy-based approaches perform better. Please note that we are assuming that a SRQ is used for incoming messages.

RDMA on the other hand provides benefits of zero-copy. Utilizing RDMA allows the direct transfer of data from the source buffer to the destination buffer with the ordering taken care of automatically by the network. Most importantly, this also holds true irrespective of the global order in which the messages are injected into the network. We now present important issues for doing direct *AlltoAll* over RDMA.

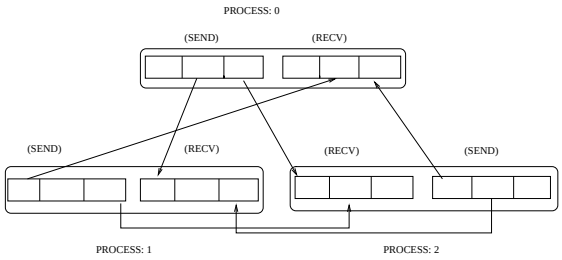

Figure 4. High-concurrency AlltoAll

Zero Copy: The main idea in our approach is to expose the registered receiver buffer to all the processes participating in the collective operation. Doing so enables direct transfer of data from a given process send buffer to the target buffer as shown in Figure 4. This would also require pinning of the send buffers of the *AlltoAll*. Once the framework for zero-copy is ready, the processes issue non-blocking RDMA write operations according to the direct algorithm as discussed in Section 2. All these operations place data directly into the receive buffers thus ensuring zero-copy. The unpinning of the memory buffers can be done in a lazy manner as done in MVAPICH. This cuts down the overhead of pinning and un-pinning when the application re-uses the same buffers.

However, one salient observation to be made in this approach is the need for explicit synchronization before the *AlltoAll* begins. This is because a given process can access a remote process's receive buffer only after the remote process called the *AlltoAll*. Issuing a write operation before

that causes incorrect behavior due to two reasons. Firstly, the framework for zero-copy might not be ready for the write to succeed. Secondly, even if the framework is ready the application might still be using the receive buffer. In this case, writing into this buffer is clearly not admissible.

Address Exchange and Completion Semantics: Address exchange is a primary requirement for zero-copy RDMA-based protocol. In our approach, we exchange addresses along with the synchronization phase. This would be similar to the allgather operation of MPI. However, if the application uses the same receive buffer, then this address can be cached so that the address exchange can be eliminated. The decision to use the cached copy or not can be taken during the synchronization phase. If the process needs to use a different buffer, it explicitly notifies this to the other processes during synchronization. We have used a log(n) algorithm similar to the one used in Barrier and Allgather, mentioned in Section 2.

For tracking completions, we have used RDMA with Immediate of IBA so that a completion entry is generated whenever a RDMA write finishes. All the processes poll for (n-1) completions where n is total number of participating processes. The work requests for using the Immediate mode of RDMA can be preposted on the SRQ.

4.2 Emulating RDMA over UD:

As discussed in Section 3, RDMA over UD emulation is required to support scalable one-sided operations. In this section we explore the possibility of emulating RDMA over UD. We outline the basic issues that need to be resolved for doing the emulation.

There are two broad design choices of emulating RDMA over UD, using zero-copy or copy-based. However, using zero-copy over UD becomes difficult to accomplish as UD does not guarantee in-order delivery of packets. Also, as shown in the earlier section, zero-copy over channel semantics serializes network transactions. Due to these two reasons, we follow a zero-copy approach for our emulation. However, a major drawback of this approach is cache pollution which is absent in zero-copy protocols.

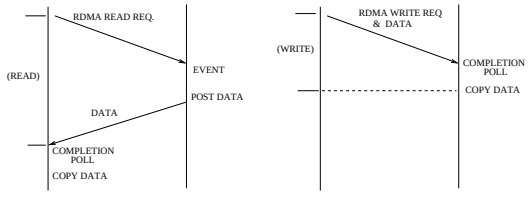

Figure 5. Emulating RDMA over UD

We now describe the protocols used for copy-based RDMA over UD emulation.

RDMA Read emulation over UD: As Figure 5 indicates, designing RDMA Read requires three steps. In the first step, the issuer sends the RDMA Read Request together with the address of the remote buffer. Once the remote side receives the request, it posts the data corresponding to the address back to the process issuing the request. The issuing process receives the data and copies into the correct destination buffer.

There are two important issues with this basic protocol. The first one pertains to the asynchronous progress of the operation. A RDMA Read request arriving at the remote end necessitates the early service of this request to achieve lowest latency possible. In the case of the native RDMA support over RC, this is taken care of by a dedicated processor on the NIC. To achieve this over UD, we need a separate host-level service thread which manages these tasks. The service thread can either poll for the incoming requests or block until a request arrives. The polling mode is not suitable because the service thread would contend with the application threads for cpu. To make the communication progress least obtrusive to the application's computation, we need to use the blocking mode.

Another issue is that the service thread can execute in the user address space or as a part of the kernel in the form of kernel-level thread. However, in either cases, blocking mode needs to be used to avoid contention of CPU resources. Our implementation is based on the user-level implementation. The application is responsible for spawning its own service thread.

IBA allows for the blocking mode or the event notification mechanism. The RDMA request message has a special bit set. This bit called as the solicitation flag triggers a completion notification event once the message arrives at the other end. After the notification is generated, the service thread is woken which follows-up on the remaining procedure. In our case, this would be posting the data corresponding to the address specified in the request packet. If the address region is not-pinned, a registration operation is required on the appropriate buffers before the descriptor is posted. Also, note that a UD operation supports transfer of only one MTU of data per descriptor. Thus, a chain of descriptors is required in case the data exceeds one MTU size. As explained in the next section, the event notification overhead can be avoided for RDMA Write operations.

RDMA Write emulation over UD: We now consider the operation of a Write over UD. As shown in the Figure, in the first step the issuing process posts the RDMA write request to the remote process. The request in this case carries both the remote address and also the data payload. Once the request reaches the remote end, the remote process copies the data into the address specified by the issuer.

Please note that another important issue which needs to be addressed in detail is that of security. The integrity of the process memory has to be carefully safeguarded so that only authorized processes can access the memory window. Since our focus is on the performance aspects of the emulation, we assume security as not an issue in our study. As opposed to a Read operation, Write operation can be optimized not to have an event generated. This is because the host process

can do a lazy copy of the data into its memory window whenever it checks for the completion of the operation. In our case, an operation is complete if the associated IBA completion entries have been successfully polled from the completion queue.

4.3 UD-based AlltoAll

As explained in Section 2, RC and UD transports exhibit different performance and scalability trends. To completely evaluate the trade-offs between these two modes of transport, we choose *AlltoAll* as the benchmark to reflect the benefits of choosing RC vs UD. *AlltoAll* over RC is already integrated into MVAPICH [6]. We now present two new designs of *AlltoAll* over UD. The first one is the direct algorithm as explained in Section 2. This algorithm exclusively uses UD for communication. The second one explained below uses both forms of transport RC and UD for data transfer.

Hybrid algorithm: This algorithm is the variant of the pair-wise exchange algorithm which is used for large messages. As discussed in Section 2, this algorithm consists of (n-1) steps where in each step a given process exchanges messages with a different peer process. This algorithm can be modified such that from a given number of steps, k steps can use RC transport for messaging and the remaining (n-1-k) steps use UD. In our implementation, the parameter k is the same across all the processes to ensure homogeneity.

We now outline the implementation issues common to both the algorithms.

Communication over UD: As explained in earlier sections, UD allows for MTU-size packet transfer between any two processes. All the information the processes need to know are the QP specific information such as QP numbers, Qkeys and Address-vectors for routing. The first step required in the communication of a message is the fragmentation into MTU-sized chunks. These chunks are then posted to the UD-QP after filling in the descriptors with the address information of the destination process. To receive the message the reverse process is required, assembling of the chunks of the data into the receive buffer. Also, note that since UD does not guarantee order of delivery of the data, each packet is marked with a sequence number so that the data is placed in the correct manner.

Reliability mechanism: Since UD is unreliable, we need host-level reliability to ensure retransmission if any packet loss is encountered. We use a Nack based approach to tackle the issue. A Nack based approach is used because the probability of packet drop in a cluster is relatively very low. Infact, we observed no packet drop in our experiments. However, we need to buffer the packets in case the packet loss is experienced. Owing to the property of *AlltoAll* that at any given time only two operations can be outstanding, reliability protocol is simplified. Thus, at any point a maximum of two messages are buffered.

5 Performance Evaluation

In this section, we explain the tests conducted and the analysis of the results. We first briefly describe our experimental testbed.

Each node of our testbed has two 3.6 GHz Intel processor and 2 GB main memory. The CPUs support the EM64T technology and run in 64 bit mode. The nodes are equipped with MT25208 HCAs with PCI Express interfaces. A Flextronics 144-port DDR switch is used to connect all the nodes. The operating system used was RedHat Linux AS4.

5.1 Performance Trade-Offs

In this section, we present the benchmark-level evaluation and analysis of the various designs and performance trade-offs studied in this paper.

5.1.1 Impact of RDMA over Send/Recv

In this section, we first demonstrate that zero-copy techniques for high-concurrent network transactions over Send/Recv with SRQ is not a good idea. We then demonstrate the performance benefits of RDMA compared to the copy-based approach.

Concurrent vs Serial: In this test, several nodes send messages to the root node. The benchmark is implemented in two different ways, with and without copy. The zero copy test follows the protocol explained in Section 2 and imposes serialization. With copy, the root copies the messages from pinned buffers to the receive buffer but there is concurrency in network operations. As can be seen from Figure 7, using copy-based or concurrent transactions performs considerably better than zero-copy or “serial” as in the figure.

AlltoAll over RDMA vs Send/Rec: The performance comparison of the zero copy RDMA-based *AlltoAll* vs Copy-based approach is shown in Figure 8. As can be seen from the figure the zero-copy *AlltoAll* performs about 38% better for the 32 nodes. Also, for more number of nodes, the performance gains are over 33% for small to medium messages. This demonstrates the impact of using memory semantics vs channel semantics for doing zero-copy high-concurrency collective operations. The copy-based *AlltoAll* is based on SRQ channel semantics incorporated into MVAPICH.

5.1.2 RC vs. UD

We now demonstrate the trade-offs between using connection-oriented RC vis-a-vis connection-less UD. We first do a a micro-benchmark evaluation and then choose *AlltoAll* as the Benchmark to compare RC and UD.

Micro-Benchmark Level Evaluation:

Basic Evaluation of IBA Transports: Figures 6 (a) and (b) illustrate the performance of RC and UD in latency and bandwidth. As indicated in the figures, RC performs better than UD, both in latency and bandwidth. However,

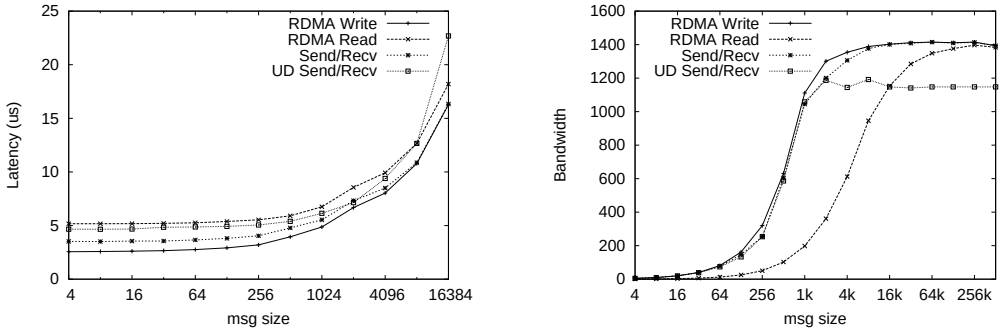

Figure 6. Basic Performance (a) Latency (b) Bandwidth

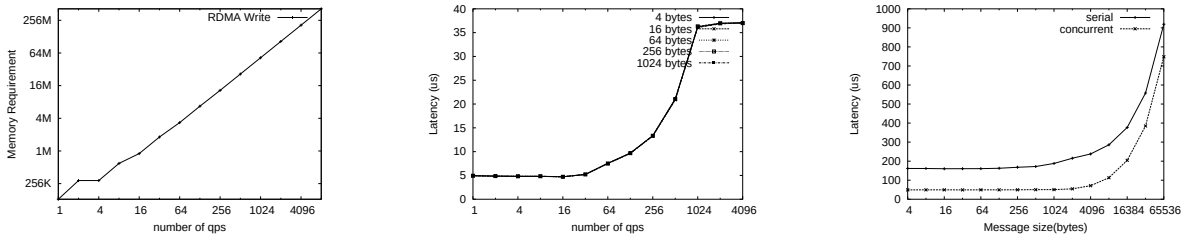

Figure 7. (a) Queue Pair Memory requirements, (b) RC Multiple QPs (c) Concurrent vs Serial

for MTU messages, both perform almost equally well for send/recv case. But, for large messages, UD performance drops compared to RC. This can be seen in the latencies of messages over MTU and in the bandwidth numbers. This can be attributed to the per-MTU overhead and lack of efficient pipelining.

Memory Scaling of RC: Figure 7(b) explains the principal drawback of using RC i.e. growing memory requirements with increase in number of connections (QPs). As can be seen from the figure, there is an upward linear trend of memory usage with the number of connections. We assign one QP per connection.

NIC caching effects: To explain the caching effects, we constructed a simple ping-pong latency benchmark with multiple connections present between the two nodes. Messages are exchanged in a round-robin manner and their latencies measured. As indicated in Figure 7(b), the latency of messages start increasing from number of connections (QPs) equal to 32. This is because the NIC has to DMA the QP context information every time this is flushed from the cache.

Benchmark Level Evaluation (AlltoAll): In this section, we first evaluate the performance of the *AlltoAll* latency for three different algorithms : *AlltoAll* over UD as discussed in this paper, *AlltoAll* over SRQ in MVAPICH, *AlltoAll* using Bruck's Algorithm in MVAPICH. The legends in the graph being direct-UD, direct-SRQ and Bruck respectively. We then show the trade-offs in performance vs resource utilization using the hybrid algorithm proposed in this paper.

Comparison of RC and UD: As shown in Figure 9 for short messages UD performs better than RC. This is in tune with the micro-benchmark evaluation above. However, as the message size increases, the performance of RC is better than UD. This is because of the better bandwidth capabilities of RC. Also, UD performs better 37% better than Bruck's which is the currently used algorithm for *AlltoAll* short messages.

Also, as seen from the figures the latency of small messages of direct-UD is almost double than that of direct-SRQ for both 32 and 61 processes. This can be mainly attributed to the NIC caching overhead. As Figure 7 indicates, the caching overhead comes into play after 16 connections have been established. Since every message incurs this overhead, the total overhead increases linearly with the size of the process group. Thus, if 1 us overhead is incurred per message, for 61 processes it would be around 60 us. For UD, this overhead is not present because only one QP is used which can be easily cached. Further, our approach performs better by around 36% compared to Bruck.

Hybrid Algorithm - Choosing Performance 'vs' Memory: We now present the results from hybrid algorithm proposed in this paper which allows choosing varying number of connections for the *AlltoAll* operation. The important point to note is that using this algorithm the upper layer can configure the maximum number of connections to use depending on the resource availability on the system. Figure 8 which shows the performance of *AlltoAll* with varying number of RC connections used and UD. The "rc-limit"

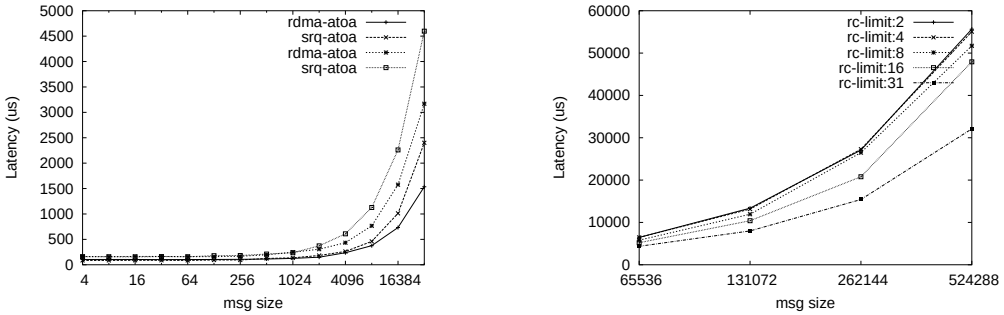

Figure 8. (a)RDMA“vs”Send/Recv (b)Hybrid algorithm

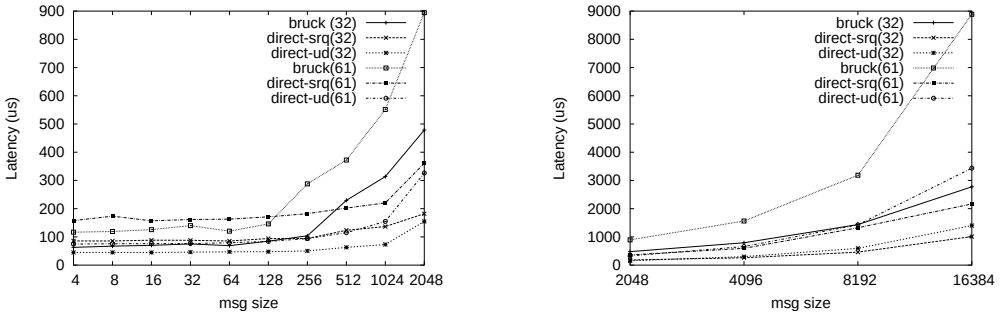

Figure 9. UD Collectives (a) Small messages (b) Medium messages

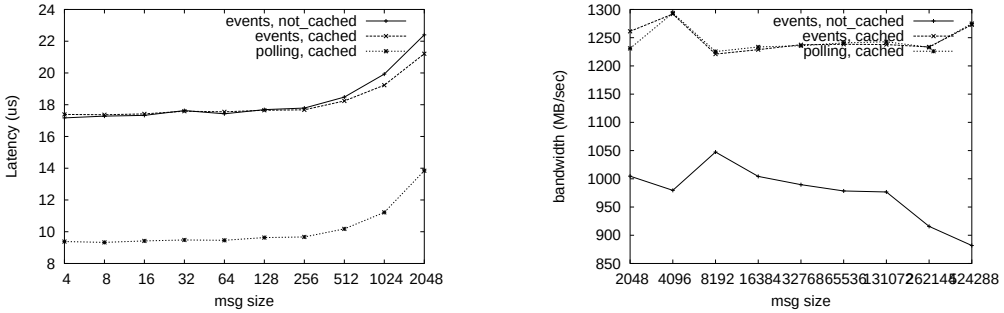

Figure 10. Emulating RDMA Read over UD (a) Latency (b) Bandwidth

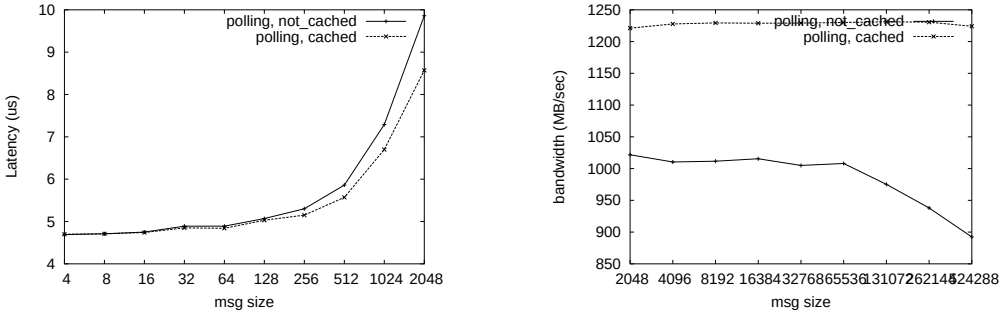

Figure 11. Emulating RDMA Write over UD (a) Latency (b) Bandwidth

corresponds to the number of RC connections used in the algorithm. As shown in the graph, increasing the number of RC connections betters performance but increases resources as shown in Figure 8 (b).

5.1.3 RDMA Emulation over UD:

We now present the results of the emulated Read and Write operations over UD. Figure 10 explains the results of the Read latency tests. We explore three cases, when the destination buffer is in the cache, out of the cache and when polling is used instead of events. Our motivation of taking into caching is because the objective of one-sided operations is to overlap computation and communication. In such scenarios it is very likely that the communication buffer might not be cached while the application is computing on a different chunk of data. As the results indicate caching determines the performance of both the latency and also bandwidth. For applications which have buffers un-cached most of the time, this would lead to performance degradation and most importantly cache pollution. Also, as the figure indicates there is around 8-9 us of event overhead compared to polling. Write operations, as indicated in Figure 11, are relatively less costly as there is no event overhead. But, like Read operations, caching plays a dominant role.

Also compared to RC performance (Figure 6), RDMA emulation performs poorly. This is especially true for one MTU message, RDMA Read which has a factor of more than three performance degradation. In the light of above observations, we conclude that emulating RDMA over UD poses performance limitations and the native support of RDMA over UD is desired.

6 Related Work

Collective Algorithms have been studied well in the past. Thakur et. al. have optimized various collective algorithms over MPICH over Myrinet and IBM SP [8]. RDMA collectives like MPI_Alltoall have also been studied over MVAPICH [7, 5, 6]. Our approach is different from these as we focus on high concurrency collective patterns. One-sided operations have also been studied in depth. Various designs for one-sided operations over InfiniBand have been proposed in [11]. However, these operations have been designed over RC support of IBA. Yelick et al have studied one-sided operations in the context of UPC and Titanium [10]. In [1] the authors propose techniques to achieve overlap of computation and communication using one-sided operations. The benefits of RDMA over UD have been explored in [9]

7 Conclusions and Future Work

As clusters increase in size, the performance and scalability of the communication subsystem becomes the key requirement for achieving overall scalability of the system. In this context, the efficiency of collective and one-sided operations is especially important as they are the widely used communication operations in different programming models like UPC, MPI-2, etc. and have to be designed harnessing the capabilities and features exposed by the underlying networks. In some cases, there is a direct match between the semantics of the operations and the underlying network primitives. InfiniBand provides two transport modes: (i) Connection-oriented *Reliable connection (RC)* supporting Memory and Channel semantics and (ii) Connection-less *Unreliable Datagram (UD)* supporting Channel semantics. Achieving good performance and scalability needs careful analysis and designing of communication operations based on these options.

In this paper, we evaluated the scalability and performance trade-offs between RC and UD transport modes. We investigated the semantic advantages of mapping collective and one-sided operations on to memory and channel semantics of IBA. We have taken AlltoAll as a case study to demonstrate the benefits of RDMA over Send/Recv and shown the performance/memory trade-offs over IB transports. Our experimental results show that the UD-based AlltoAll performs 38% better than Bruck's for short messages and upto two times better than the direct AlltoAll over RC. Since IBA does not provide RDMA over UD in hardware, we emulated the same in our study. Our results show a performance dip of up to a factor of three for emulated RDMA Read latency as compared to RC, highlighting the need for hardware based RDMA operations over UD. We thus emphasize the need for extending the IBA specification to allow for support of RDMA over UD. As future work, we plan to systematically study these trade-offs on large scale cluster and undertake application level evaluation.

References

[1] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap Support. In *IPDPS*, 2006.

[2] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems. *IEEE Transactions in Parallel and Distributed Systems*, 8(11):1143–1156, November 1997.

[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. InfiniBand Architecture Specification, Release 1.1. http://www.infinibandta.org, November 2002.

[5] A. Mamidala, J. Liu, and D. K. Panda. Efficient Barrier and Allreduce on IBA clusters using hardware multicast and adaptive algorithms. In *IEEE Cluster Computing*, 2004.

[6] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html, January 2003.

[7] S. Sur, H.-W. Jin, and D. K. Panda. Efficient and Scalable All-to-All Exchange for InfiniBand-based Clusters. In *International Conference on Parallel Processing (ICPP)*, 2004.

[8] R. Thakur and W. Gropp. Improving the Performance of Collective Operations in MPICH. In *Euro PVM/MPI*, 2003.

[9] B. the connection: RDMA deconstructed. Rajeev Sivaram and Govindaraju, R.K. and Hochschild, P. and Blackmore, R. and Piyush Chaudhary. In *HOTI*, 2005.

[10] UC Berkeley/LBNL. Berkeley upc - unified parallel c. http://upc.lbl.gov/.

[11] A. Vishnu, G. Santhanaraman, W. Huang, H.-W. Jin, and D. K. Panda. Supporting MPI-2 One Sided Communication on Multi-Rail InfiniBand Clusters: Design Challenges and Performance Benefits. In *HiPC*, 2005.