Adaptive Connection Management for Scalable MPI over InfiniBand[*]

Weikuan Yu[†] Qi Gao[†] Dhabaleswar K. Panda[†]

Network-Based Computing Lab[†]
Dept. of Computer Sci. & Engineering
The Ohio State University
{*yuw,gaoq,panda*}@*cse.ohio-state.edu*

Abstract

Supporting scalable and efficient parallel programs is a major challenge in parallel computing with the widespread adoption of large-scale computer clusters and supercomputers. One of the pronounced scalability challenges is the management of connections between parallel processes, especially over connection-oriented interconnects such as VIA and InfiniBand.

In this paper, we take on the challenge of designing efficient connection management for parallel programs over InfiniBand clusters. We propose adaptive connection management (ACM) to dynamically control the establishment of InfiniBand reliable connections (RC) based on the communication frequency between MPI processes. We have investigated two different ACM algorithms: an on-demand algorithm that starts with no InfiniBand RC connections; and a partial static algorithm with only $2 * logN$ *number of InfiniBand RC connections initially. We have designed and implemented both ACM algorithms in MVAPICH to study their benefits. Two mechanisms have been exploited for the establishment of new RC connections: one using InfiniBand unreliable datagram and the other using InfiniBand connection management. For both mechanisms, MPI communication issues, such as progress rules, reliability and race conditions are handled to ensure efficient and lightweight connection management. Our experimental results indicate that ACM algorithms can benefit parallel programs in terms of the process initiation time, the number of active connections, and the resource usage. For parallel programs on a 16-node cluster, they can reduce the process initiation time by 15% and the initial memory usage by 18%.*

1. Introduction

Ultra-scale computing environments such as clusters with multi-thousand processors [1] lead to many new challenges in parallel programming, especially in their requirements of supporting parallel programs over an unprecedent number of processes. The MPI (Message Passing Interface) standard [11] has evolved as a *de facto* parallel programming model for these systems. Traditional research over MPI has been largely focused on high performance communication between processes. However, for parallel programs with thousands of processes, another challenging issue is how to establish and maintain the communication channels among thousands of processes. Under such execution environment, even how to get these programs launched gracefully can be a major concern [4, 2, 7].

InfiniBand Architecture (IBA) [8] has been introduced as an open standard in industry to design next generation high-end clusters for both data-center and high performance computing. More and more large cluster systems with InfiniBand are being deployed, such as the 5th, 20th, and 51th most powerful supercomputers as listed in the November 2005 Top 500 list [1]. InfiniBand provides four types of transport services: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC), and Unreliable Datagram (UD). Among the four, the most commonly used service is RC due to its high performance and RDMA capability.

MPI, on the other hand, does not specify any connection model, but assumes that all processes are logically

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

connected and leaves the connection management specific issues to the lower device layer. For connection-less interconnects, such as Quadrics [16] and Myrinet [12], an MPI process can start MPI communication without extra mechanisms to manage peer-to-peer connections. On top of InfiniBand, however, for any pair of processes to communicate over RC for its high performance and RDMA capability, a pair of RC queue pairs (QPs) must be created on each node with a connection established between them. To enable high performance RDMA fast path for small messages, additional RDMA receive buffers also need to be provided for each connection.

Several of the most commonly used MPI implementations over InfiniBand, such as MVAPICH [13], set up RC connections between every process pairs *a priori*. Because of its connection-oriented nature, every process needs to allocate a dedicated QP for each peer process. This leads to quadratic increasing number of RC connections, e.g., 1024*1023 connections for a 1024-process MPI program. These number of connections in turn lead to prolonged startup time for the need of creating queue pairs, exchanging connection information and establishing connections. This also leads to heavy resource usage, taking into account of the memory needed for all the QPs and their associated send and receive WQEs, as well as RDMA send and receive buffers.

Table 1. Average number of communicating peers per process in several large-scale applications (Courtesy of J. Vetter, et. al [17])

Application	Number of Processes	Average Number of Distinct Destinations
sPPM	64	5.5
	1024	< 6
SMG2000	64	41.88
	1024	< 1023
Sphot	64	0.98
	1024	1
Sweep3D	64	3.5
	1024	< 4
Samrai 4	64	4.94
	1024	< 10
CG	64	6.36
	1024	< 11

In fact, research on communication characteristics of parallel programs [17] indicates that not all pairs of MPI processes communicate among each other with equal frequency. Table 1 shows the average number of communicating peers per process in some scientific applications. The majority of process pairs do not communicate between each other. Thus, maintaining a fully-connected network not only leads to the aforementioned scalability problems, but also negatively affects the performance of the communicating processes. This is because the MPI program has to continuously check for potential messages coming from any process, which drives the CPU away from attending to traffic of the most frequently communicated processes and destroys the memory cache locality that could be achieved thereof. There have been discussions in the IBA community to either provide only UD-based communication, or RC on-demand via an out-of-band asynchronous message channel. However, UD does not provide comparable performance as RC; the processing of out-of-band asynchronous messages will introduce the need of another thread that contends for CPU with the main thread. So these solutions can result in performance degradation. It remains to be systematically investigated what connection management algorithms can be provided for parallel programs over InfiniBand, and what are their performance and scalability implications.

In this paper, we take on the challenge of providing appropriate connection management for parallel programs over InfiniBand clusters. We propose adaptive connection management (ACM) to manage different types of InfiniBand transport services. The time to establish and tear town RC connections is dynamically decided based on communication statistics between the pair of processes. New RC connections are established through either an unreliable datagram-based mechanism or an InfiniBand connection Management-based mechanism. We have also studied strategies to overcome challenging issues, such as race conditions, message ordering and reliability, for the establishment new RC connections. The resulting ACM algorithms have been implemented in MVAPICH [13] to support parallel programs over InfiniBand. Our experimental data with NAS application benchmarks indicate that ACM can significantly reduce the average number of connections per process, and it is also beneficial in improving the process initiation time and reducing memory resource usage. Note, one of the side effects of ACM is that it moves connection establishment from the process initiation stage into the actual critical path of parallel communication. Our evaluation also indicates that ACM has little performance impact to microbenchmarks and NAS scientific applications since only very few connections are established on the basis of frequent communication.

The rest of the paper is presented as follows. Section 2 provides background information on InfiniBand and its connection management interface. Section 3 and 4 describe ACM and the connection establishment mechanisms in detail. Section 5 provides performance results. Section 6 provides a brief review of the related work. Section 7 concludes the paper.

2. Background

In this Section, we provide brief overviews of InfiniBand architecture and its connection management interface.

2.1. Overview of InfiniBand Architecture

The InfiniBand Architecture (IBA) [8] is an open specification designed for interconnecting compute nodes, IO nodes and devices in a system area network. As shown in Fig. 1, it defines a communication architecture from the switch-based network fabric to transport layer communication interface for inter-processor communication. Processing nodes and I/O nodes are connected as end-nodes to the fabric by two kinds of channel adapters: Host Channel Adapters (HCAs) and Target Channel Adapters (TCAs). IBA supports communications in channel semantics with traditional send/receive operations, as well as communications in memory semantics with RDMA operations. RDMA operations allow one side of the communication parties to exchange information directly with the remote memory without the involvement of the remote host processors.

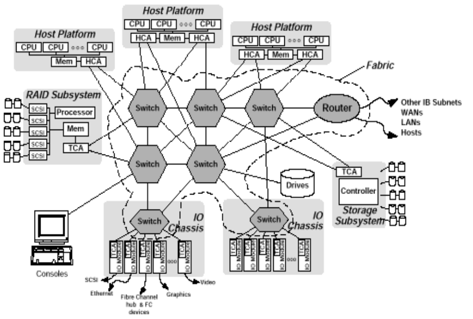

Fig. 1. The Switch Fabric of InfiniBand Architecture (Courtesy InfiniBand Trade Association)

InfiniBand provides four types of transport services: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC), and Unreliable Datagram (UD). The often used service is RC in the current InfiniBand product and software. It is also our focus of this paper. To support RC, a connection must be set up between two QPs before any communication. In the current InfiniBand SDK, each QP has a unique identifier, called *QP-ID*. This is usually an integer. For network identification, each HCA also has a unique a local identifier (*LID*). One way to establish a connection is to exchange the QP IDs and LIDs of a pair of QPs and then explicitly program the queue pair state transitions. Another way is to use InfiniBand connection management interface as described later in Section 2.2. In the IBA community, a new interface called RDMA CMA (Connection Management Agent) has been proposed recently [14]. RDMA CMA over IBA is derived on top of IBCM, but provides an easy-to-use, portable, yet similar approach for connection establishment. It is currently available only in the OpenIB Gen2 stack. We plan to study the benefits of RDMA CMA in our future work.

2.2. InfiniBand Connection Management

InfiniBand Communication Management (IBCM) encompasses the protocols and mechanisms used to establish, maintain, and release different InfiniBand transport services, such as RC, UC, and RD. Communication Managers (CMs) inside IBCM set up QPs (or end-to-end context for RD) upon calls to the IBCM interface. CMs communicate with each other and resolve the path to remote QPs through an address resolution protocol. There are two models to establish a connection: one is Active/Passive (also referred as client/server) model, the other Active/Active (or peer-to-peer) model. In the client/server model, the server side listens for connection requests with a service id; the client side initiates a connection request with a matching service ID. In the peer-to-peer model, both sides actively send connection requests to each other, and a connection is established if both requests contain matching service IDs. Compared to the peer-to-peer model, the client/server model is more mature in the current InfiniBand implementations, and is what we have studied in this paper.

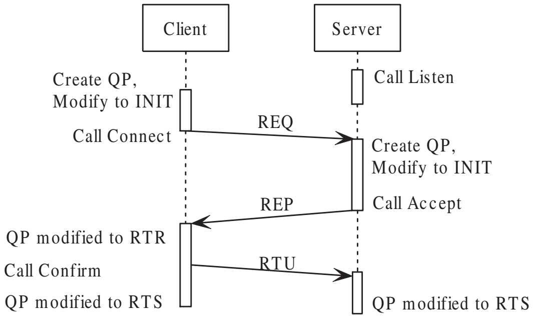

Fig. 2. The Client/Server Model of IBCM

Fig 2 shows the diagram of the client/server model of IBCM. The server begins to listen on a service ID. A client then creates a QP and initiates a request (REQ) to the server with a matching service ID. If the server can match the service ID, a callback function is called to create a QP, accept the client's request, and confirm the request with a reply (REP). When the client receives the server's reply indicating that its request is accepted, a client-side callback handler is called, within which the client confirms the establishment of a new connection back to the server via a RTU (ready-to-use) message. When the server receives RTU, the connection is then ready for communication. In the client/server model, QPs are created in an INIT state and progressed through RTR (Ready-to-Receive) to RTS (Ready-to-Send) by CMs as shown in the figure.

3. Adaptive Connection Management

Since different interconnects may have different software and hardware capabilities and exhibit different communication characteristics, the design of MPI [11] in general leaves interconnect specific issues to ADI (abstract device interface) implementations. For example, MPI does not specify any connection model, but assumes that all processes are logically connected. This does not lead to complications for connection-less interconnects, such as Quadrics [16] and Myrinet [12], on which MPI process can start MPI communication without extra care for managing peer-to-peer connections. InfiniBand [8], however, comes with a plethora of transport services, from the typical connection-less Unreliable Datagram (UD) to the high-performance connection-oriented Reliable Connection (RC). Different types of transport services come with different performance qualities and different resource requirements. Within a parallel application spanning thousands of parallel processes, a process potentially needs to handle such resource requirements for a large number of connections depending on the number of peers it is communicating with. On top of this, a scalable MPI implementation also needs to satisfy the memory requirements from parallel applications. These complexities all need to be handled within rigid resource constraints. Therefore, when and how to enable what types of connections is a very important design issue for scalable and high performance MPI implementations.

To this purpose, we propose Adaptive Connection Management (ACM) to handle these complexities. There are two two different ACM algorithms: on-demand and partially static. In the on-demand algorithm, every process is launched without any RC connections; in the partially static algorithm, each process is initially launched with at most $2 * logN$ of RC connections to communicate with peers that have a rank distance of 2^N from it. These initial $2 * logN$ RC connections are meant to capture the frequent communication patterns based on the common binary tree algorithms used in many MPI collective operations. The main design objective of ACM is to manage InfiniBand transport services in an adaptive manner according to the communication frequency and resource constraints of communicating processes. To this purpose, new RC connections are established only when a pair of processes have exhibited a frequent communication pattern. In this paper, this is decided when two processes have communicated more than 16 (an adjustable threshold) messages. Compared to the commonly used static connection management, ACM algorithms are designed to allow the best adaptivity, while the partially static algorithm also allows applications to pre-configure common communicating processes with RC connections at the startup time. The rest of the section provides the design details of ACM.

3.1. Designing ACM in MPI over InfiniBand

Fig. 3 shows a diagram about the intended ACM functionalities in a typical MPI software stack. As shown in the figure, ACM works in parallel with the ADI's channel interface (CH2), which is in charge of the actual message communication functionalities. While the channel interface mainly provides appropriate channels to transport messages based on their sizes and destinations, ACM controls when to activate the transport services of different performance capabilities and maintains the communication statistics of these channels. Specifically, ACM manages the following information about transport services over InfiniBand.

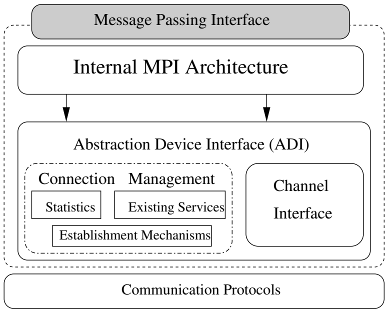

Fig. 3. Adaptive Connection Management in MPI Software Stack

- **Existing transport services** – In ACM, each process maintains information about which transport services are available to reach peer processes. The most commonly used InfiniBand transport services are UD and RC. All processes start with a UD queue pair, through which other processes can request the establishment of RC connections.

- **Resource allocation and communication statistics** – Messages can be transmitted over the existing transport services. The number of messages and the total message size are recorded as communication statistics, which determine when to set up new RC connections. Future work can also introduce mechanisms to dismantle RC connections when some processes are either quiescent or relatively less active for a certain amount of time.

- **Establishment mechanisms for new transport services** – Currently, we have exploited two different mechanisms for establishing new RC connections over InfiniBand: (1) Connection establishment via UD-based QP-ID exchange; and (2) IBCM-based RC connection establishment. In future, we plan to study the benefits of RDMA CMA for MPI connection management over the InfiniBand Gen2 stack [14].

On-demand and partially static ACM algorithms can use either connection establishment mechanisms to set up new RC connections. In this work, we intend to study combinations of ACM algorithms and connection establishment mechanisms to gain insights into the following questions:

1. How much can the adaptive connection management help on reducing the number of the connections for scientific applications?

2. What performance impact will the adaptive connection management have?

3. How much can the adaptive connection management help on reducing the process initiation time?

4. What benefits will the adaptive connection management have on the memory resource usage?

4. Connection Establishment

In this section, we describe the design of two mechanisms for connection establishment: UD-based and IBCM-based mechanisms. Though these two mechanisms have some similarities in their design issues, such as progress rules and duplication avoidance, the strategies to overcome them are different. The rest of the section describes these issues separately in detail.

4.1. UD-Based Connection Establishment

Fig. 4 shows the diagram of UD-based connection establishment. Upon frequent communication between Proc A and Proc B, Proc A sends a request for new connection to Proc B. Proc B responds with a reply to acknowledge the request. A new connection is established at the end of a three-way exchange of *request*, *reply* and *confirm* messages. The actual scenario is more complicated than what is shown in the diagram. We describe the detailed design issues as follows:

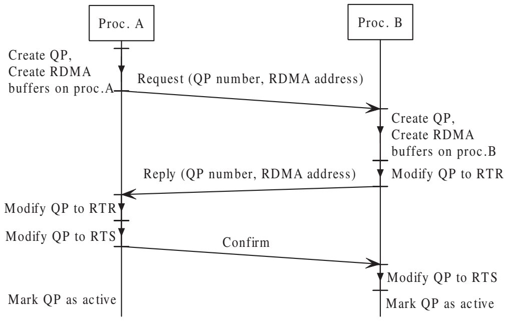

Fig. 4. UD-Based Connection Establishment

- **Progress rules** – When using UD-based connection management, an MPI process has to handle extra UD messages for connection setup purposes. To provide a clean and light-weight solution, we provide a dedicated completion queue for UD messages. However, this completion queue is being polled at a much reduced frequency compared to the completion queue for regular data messages. This is intended to reduce the extra burden on the main MPI process for attending to extra traffic.

- **Ordered reliable message delivery** – Any of three messages may get lost since they are sent over UD. We introduce timeout-based retransmission to handle such situations. However, this may also lead to duplicated requests. For this problem, a sequence number is introduced along with UD messages to avoid redundant requests.

- **Race conditions** – Race conditions between two requests can occur for the establishment of the same RC connection since Proc B may have sent out a connection request to Proc A at the same. Both Proc A and B are trying to set up a RC connection between them. To guard against race conditions, a process responds to a request with a positive acknowledgment only when it has not initiated a request or its rank is higher than the source rank contained in the request. Status flags are introduced to reflect the progression of connection state during the connection establishment.

4.2. IBCM-Based Connection Establishment

In the IBCM-based mechanism each process starts a new listening thread with a unique service ID, which is set to its rank plus a constant so that every process knows the service IDs of all other processes. When a process wants to establish a connection, it sends a request to the corresponding target. The procedure in general follows what we have described in section 2.2. In particular, we describe the following design issues as follows.

Synchronization – To establish a RC connection via IBCM, some information such as source/destination ranks must be exchanged during the connection establishment phase. And we also need to make sure that the receive descriptor be posted before the RC connection progresses to RTR. Thus, it would be more efficient if one can integrate the receive descriptor posting and RDMA receive buffers exchange into the process of IBCM connection establishment. To this purpose, we ensure the server have prepared the receive descriptors and buffers before it replies back to the client. After the client receives the reply, we make sure that the client completes the same preparation before it confirms back to the server with a RTU (ready-to-use) message (Fig. 2). Only until the server receives the expected RTU message and the client get the correct local completion of RTU message, will the connection be marked as active on both sides. Both sides are then correctly synchronized on the state of the new RC connection and the connection is ready to use.

Race conditions – Each process has two possible activities for establishing new connections. One is the main MPI thread that may connect to a target process as a client, the other being the listening thread that functions as a CM server for incoming connection requests. It is critical to ensure that, at any time, one of them is driving the establishment of a new connection. Otherwise, both of them will fail for incorrect sharing of the same queue pair.

We describe our solution with two arbitrary processes, Proc A and B, the rank of A greater than B. When both of them simultaneously send a request to the other process, we let A act as a server to continue and have the CM server of B to reject the request from A. The client request from B continues to be processed by the CM server of A and finish the establishment of a new connection. In addition, to avoid an inconsistent connection state, before A accepts B's request, it also needs to wait until a reject notification from B is received. Structures related to connection progression are critical sections being protected by mutexes and they are used to avoid the race conditions between the main thread and the CM thread.

5. Performance Evaluation

In this section, we describe the performance evaluation of our design. The experiments were conducted on two clusters. One is a cluster of 8-node SuperMicro SUPER P4DL6, each with dual Intel Xeon 2.4GHz processors, 1GB DRAM, PCI-X 133MHz/64-bit bus. The other is a cluster of eight SuperMicro SUPER X5DL8-GG nodes: each with dual Intel Xeon 3.0 GHz processors, 512 KB L2 cache, PCI-X 64-bit 133 MHz bus, 533MHz Front Side Bus (FSB) and a total of 2GB PC2100 DDR-SDRAM physical memory. The nodes are connected using the Mellanox InfiniScale 24 port switch MTS 2400. The original MVAPICH [13] release we used is 0.9.5 with patches up to 118. We evaluated the original static connection management of MVAPICH-0.9.5 (referred to as *Orig*) and the following combinations of ACM algorithms and connection establishment mechanisms (UD/IBCM): partially static ACM with UD (UD-PS), on-demand ACM with UD (UD-OD), and on-demand ACM with IBCM (CM-OD).

5.1. Average Number of Connections

One of the main benefits of adaptive connection management is to increase the scalability of MPI implementations in terms of scalable usage of RC connections over InfiniBand. Table 2 lists the average number of InfiniBand RC connections used in the NAS application benchmarks with different ACM configurations compared to the original static algorithm. With UD-OD, the number of RC connections for NAS benchmarks are in general less than the numbers reported before in [19]. This suggests that UD-OD can indeed eliminate all the connections that are either not communicating or very rarely. For example, the number of connections used in EP is 0, because no connections are established between the processes since the processes only communicate rarely with a few barrier operations for the purpose of synchronization.

5.2. Process Initiation Time

Since adaptive connection management reduces the initial number of InfiniBand RC connections, the time

Table 2. Average Number of Connections in NAS Benchmarks

Algorithm	SP	BT	MG	LU	IS	EP	CG
16 Processes							
Orig	15	15	15	15	15	15	15
UD-OD	6	6	5	3.6	15	0	2.7
UD-PS	9.5	9.5	7	7	15	7	7.8
32 Processes							
Orig	–	–	31	31	31	31	31
UD-OD	–	–	7	4.1	31	0	3.8
UD-PS	–	–	9.5	9	31	9	9.8

needed for the establishment of these connections in the original static algorithm is no longer needed. We have investigated the benefits of our algorithms in terms of process initialization time. We first measured the initialization time using a ssh/rsh-based startup scheme and noticed that the variation in startup time is too high to obtain the portion of reduced initialization time. Instead we incorporated the ACM algorithms into a scalable MPD-based startup scheme and measured the initialization time.

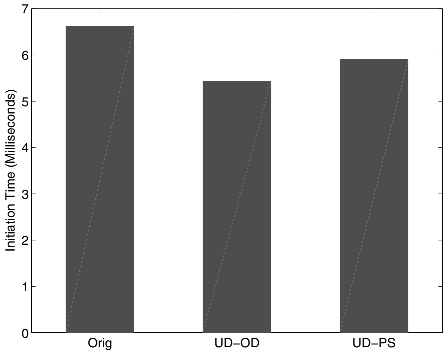

Fig. 5. Initiation Time of Different Connection Management Algorithms

Fig. 5 shows process initialization time for 32-process programs over the 16-node cluster. Compared to the original algorithm, UD-OD can reduce process initialization time by 15-20%, while UD-PS can reduce the time by around 10%. The amount of reduction in initialization time is smaller for UD-PS because around $2 * logN$ connections need to be established at the beginning.

5.3. Reduction in Memory Usage

Another benefit of adaptive connection management is reducing the memory resource usage. Because the number of connections is tightly coupled to the communication behavior of parallel applications, it is not easy to decide an appropriate time to take a snapshot of memory usage. To gain insights into the memory usage, we measured the initial memory usage when the parallel processes first start up, i.e., at the end of `MPI_Init`.

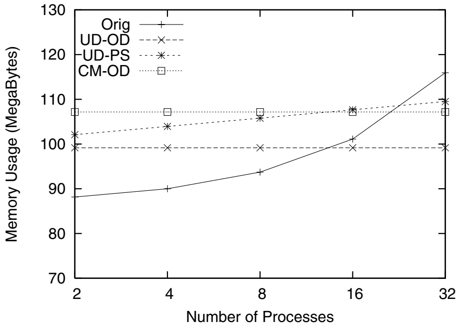

Fig. 6. Memory Usage of Different Connection Management Algorithms

Fig. 6 shows the memory resource usage for a parallel program with varying number of processes over the 16-node cluster. Compared to the original, all ACM algorithms start with slightly higher memory usages, this is because the connection management algorithms introduced additional data structure such as a UD-queue pair and/or a CM server thread, which consumes slightly more memory. However, the original algorithm has a clearly faster increasing trend of memory usage compared to others. For a 32-process application, UD-OD can reduce memory usage by about 17%, while UD-PS can reduce the memory by about 12%. Again, because UD-PS has to set up around $2 * logN$ connections, the amount of memory usage is higher than that of UD-OD. These results suggest ACM algorithms are beneficial in terms of memory resource usage. These benefits are expected to be more significant as system size increases.

5.4. Impact on Latency and Bandwidth

To find out the impact of ACM on the basic latency and bandwidth performance of MVAPICH, we have compared the performance of different algorithms with the original. As shown in Figures 7 and 8, UD-OD and UD-PS have negligible impacts on the latency and bandwidth performance. This suggests that our implementations are indeed light-weight and efficient. However, CM-OD causes degradation on latency and bandwidth. This is expected because the IBCM-based ACM introduces additional threads for managing connection requests and establish new connections.

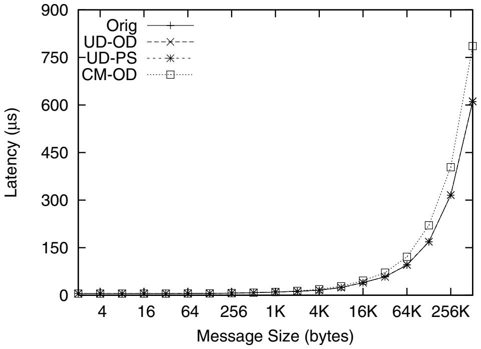

Fig. 7. Latency of Different Connection Management Algorithms

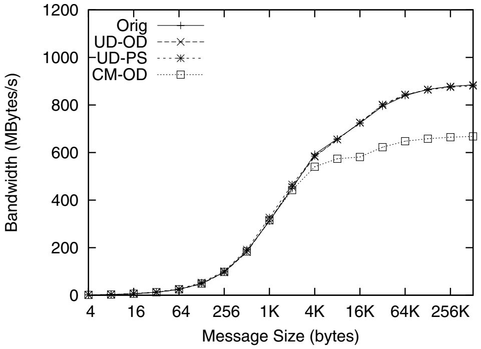

Fig. 8. Bandwidth of Different Connection Management Algorithms

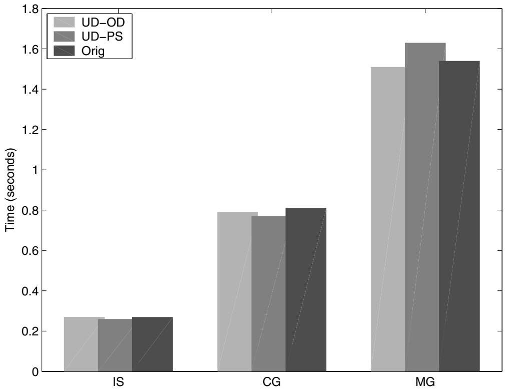

Fig. 9. Performance of IS, CG, MG, Class A

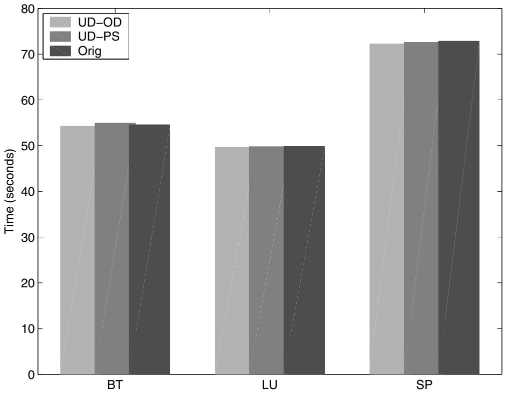

Fig. 10. Performance of BT, LU, SP, Class A

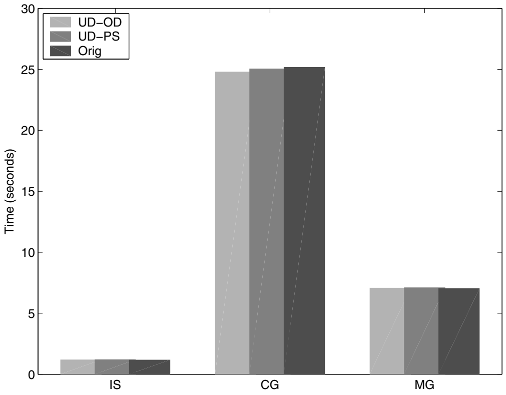

Fig. 11. Performance of IS, CG, MG, Class B

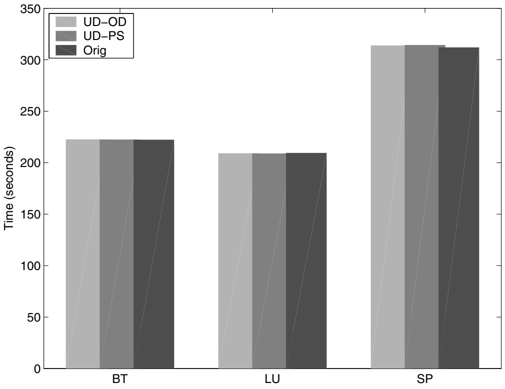

Fig. 12. Performance of BT, LU, SP, Class B

5.5. Performance of NAS Parallel Benchmarks

The NAS suite consists of a set of programs, such as MG, CG, IS, LU, SP and BT. We compared the performance of these NAS programs over UD-PS and UD-OD to the original. Figures 9, 10, 11 and 12 show the performance of NAS programs with different program sizes and numbers of processes. Our testbed has 32 processors, the largest number of processes tested in CG, MG, IS is 32, while only 16 in SP and BT since they require a square number of processes. The performance results of NAS benchmarks indicate that the proposed algorithms have little performance impacts. Thus, the proposed algorithms can provide benefits in terms of scalability and memory resource usage while not having any impact on performance. We believe that the benefits of scalable resource usage would contribute to the performance improvement, which could become noticeable only with larger scale scientific applications. We plan to investigate into this issue further.

6. Related Work

Significant amount of research were done on characterizing, optimizing and improving the performance of different MPI implementations. Wong et. al. [18] studied the scalability of the NAS Parallel benchmarks from the aspect of resident working set and communication performance. Vetter et. al. [17] characterized the communication pattern of large scientific applications in terms of message sizes and most frequently used point-to-point and collective operations. Liu et. al. [10] designed high performance RDMA fast path over InfiniBand. Liu et. al. [9] also studied how to exploit the high bandwidth potential of InfiniBand by enabling multiple RC connections between pairs of MPI processes. Brightwell et. al. [3] investigated the impact of receive queue lengths to the MPI performance.

Lately, the scalability issues of the MPI implementations attracted attentions from more research groups. Yu et. al. [20] studied how to improve the startup time of parallel programs over InfiniBand clusters by optimizing the all-to-all exchange of InfiniBand QP information at the process initiation time. Castin et. al. [5] designed a run-time environment for supporting future generations of peta-scale high performance computing. Petrini et. al. [15] investigated how system noise can affect the performance of parallel programs over on a large-scale system, ASCI Q.

There had been previous research efforts carried out to study the impact of connection management on the performance of parallel applications. Brightwell et. al. [3] analyzed the scalability limitations of VIA in supporting the CPlant runtime system as well as any high performance implementation of MPI. While not taking into account the impacts of the number of connections on the scalable usage of computation and memory resources to different connections, the authors argued that *on-demand connection management* could not be a good approach to increase the scalability of the MPI implementation by qualitative analysis. Wu et. al. [19] demonstrated that on-demand connection management for MPI implementations over VIA [6] could achieve comparable performance as the *static mechanism* with efficient design and implementation. Our work continues the research efforts of on-demand connection management [19] and scalable startup [20] to improve the scalability of MPI implementations over InfiniBand.

7. Conclusions

In this paper, we have explored different connection management algorithms for parallel programs over InfiniBand clusters. We have introduced adaptive connection management to establish and maintain InfiniBand services based on communication frequency between a pair of processes. Two different mechanisms have been designed to establish new connections: an unreliable datagram-based mechanism and an InfiniBand connection management-based mechanism. The resulting adaptive connection management algorithms have been implemented in MVAPICH to support parallel programs over InfiniBand. Our algorithms have been evaluated with respect to their abilities in reducing the process initiation time, the number of active connections, and the communication resource usage. Experimental evaluation with NAS application benchmarks indicates that our connection management algorithms can significantly reduce the average number of connections per process.

In future, we intend to study the benefits of this connection management framework in larger scale InfiniBand clusters. We also plan to apply adaptive connection management algorithms for MPI over the latest OpenIB-Gen2 [14] stack.

Additional Information – Additional information related to this research can be found on the following website: http://nowlab.cse.ohio-state.edu/.

References

[1] TOP 500 Supercomputers. http://www.top500.org/.

[2] R. Brightwell and L. A. Fisk. Scalable parallel application launch on Cplant. In *Proceedings of Supercomputing, 2001*, Denver, Colorado, November 2001.

[3] R. Brightwell and A. Maccabe. Scalability limitations of via-based technologies in supporting mpi. March 2000.

[4] R. Butler, W. Gropp, and E. Lusk. Components and interfaces of a process management system for parallel programs. *Parallel Computing*, 27(11):1417–1429, 2001.

[5] R. H. Castain, T. S. Woodall, D. J. Daniel, J. M. Squyres, B. Barrett, and G. E. Fagg. The open run-time environment (openrte): A transparent multi-cluster environment for high-performance computing. In *Proceedings, 12th*

European PVM/MPI Users' Group Meeting, Sorrento, Italy, September 2005.

[6] Compaq, Intel, and Microsoft. The Virtual Interface Architecture (VIA) Specification. available at http://www.viarch.org.

[7] E. Frachtenberg, F. Petrini, J. Fernandez, S. Pakin, and S. Coll. STORM: Lightning-Fast Resource Management. In *Proceedings of the Supercomputing '02*, Baltimore, MD, November 2002.

[8] Infiniband Trade Association. http://www.infinibandta.org.

[9] J. Liu, A. Vishnu, and D. K. Panda. Building multirail infiniband clusters: Mpi-level design and performance evaluation. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 33, Washington, DC, USA, 2004. IEEE Computer Society.

[10] J. Liu, J. Wu, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. *Int'l Journal of Parallel Programming*, 32(3), June 2004.

[11] Message Passing Interface Forum. MPI: A message-passing interface standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8(3–4), 1994.

[12] Myricom. Myrinet Software and Customer Support. http://www.myri.com/scs/GM/doc/, 2003.

[13] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.

[14] Open Infiniband Alliance. http://www.openib.org.

[15] F. Petrini, D. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *SC '03*, November 2003.

[16] Quadrics Supercomputers World, Ltd. Quadrics Documentation Collection. http://www.quadrics.com/, 2004.

[17] J. S. Vetter and F. Mueller. Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. In *IPDPS*, April 2002.

[18] F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. E. Culler. Architectural Requirements and Scalability of the NAS Parallel Benchmarks. In *Proceedings of Supercomputing*, 1999.

[19] J. Wu, J. Liu, P. Wyckoff, and D. K. Panda. Impact of On-Demand Connection Management in MPI over VIA. In *Proceedings of the IEEE International Conference on Cluster Com puting*, 2002.

[20] W. Yu, J. Wu, and D. K. Panda. Fast and Scalable Startup of MPI Programs In InfiniBand Clusters. In *Proceedings of the International Conference on High Performance Computing '04*, Banglore, Inida, December 2004.