Dynamic Time-Variant Connection Management for PGAS Models on InfiniBand

Abhinav Vishnu,* Manoj Krishnan,* and Pavan Balaji‡

* Pacific Northwest National Laboratory, Richland, WA 99352
Email: {abhinav.vishnu, manoj}@pnl.gov

‡ Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL 60439
Email: {balaji}@mcs.anl.gov

***Abstract*—InfiniBand (IB) has established itself as a promising network infrastructure for high-end cluster computing systems as evidenced by its usage in the Top500 supercomputers today. While the IB standard describes multiple communication models (including reliable-connection (RC), and unreliable datagram (UD)), most of its promising features such as remote direct memory access (RDMA), hardware atomics and network fault tolerance are only available for the RC model which requires connections between communicating process pairs. In the past, several researchers have proposed on-demand connection management techniques that establish connections when there is a need to communicate with another process. While such techniques work well for algorithms and applications that only communicate with a small set of processes in their life-time, there exists a broad set of applications that do not follow this trend. For example, applications that perform dynamic load balancing and adaptive work stealing have a small set of communicating neighbors at any given time, but over time the total number of neighbors can be very high; in some cases, equal to the entire system size.**

In this paper, we present a dynamic time-variant connection management approach that establishes connections on-demand like previous approaches, but further intelligently tears down some of the unused connections as well. While connection tear-down itself is relevant for any programming model, different models have different complexities. In this paper, we study the Global Arrays (GA) PGAS model for two reasons: (1) the simple one-sided communication primitives provided by GA and other PGAS models ensure that connection requests are always initiated by the origin process without explicit synchronization with the target process—this makes connection tear-down simpler to handle; and (2) GA supports applications in several domains such as computational chemistry (NWChem) and computational biology (ScalaBLAST) that demonstrate this behavior making it an obvious first target for the proposed enhancements. We evaluate our proposed approach using NWChem computational chemistry application using up to 6144 processes, and show that our approach can significantly reduce the memory requirements of the communication library while maintaining its performance.

I. Introduction

Amongst many network interconnects that integrate into cluster computing systems today, InfiniBand (IB) [1] has established itself as a promising interconnection technology. As reflected in the TOP500 [2] rankings, IB has been observing wide acceptance due to its high performance and open standard, with 28% of the systems using IB as their interconnect. Even though the IB standard describes multiple communication models (including reliable-connection (RC), and unreliable datagram (UD)), most of its promising features (RDMA, hardware atomics, network fault tolerance) are only available for the RC model which requires connections between communicating process pairs. This is a scalability concern as each connection can consume up to 44KBytes [3] of memory, resulting in hundreds of megabytes of connection memory footprint on large scale systems.

To address this problem, on-demand connection management mechanisms have been proposed with the message passing interface (MPI) model [4], [3], [5], [6], [7] as well as partitioned global address space (PGAS) models such as Global Arrays (GA) [8], [9]. The proposed techniques improve the memory footprint of a communication library by establishing connections only when there is a need to communicate, and not before. While such techniques work well for algorithms and applications that only communicate with a small set of processes in their life-time, there exists a broad set of applications that do not follow this trend. For example, applications that perform dynamic load balancing and adaptive work stealing [10] have a small set of communicating neighbors at any given time, but over time the total number of neighbors can be very high; in some cases, equal to the entire system size.

In this paper, we present a dynamic time-variant connection management approach that establishes connections on-demand like previous approaches, but further intelligently tears down some of the unused connections as well. While connection tear-down itself is relevant for any programming model, different models have different complexities. For example, programming models such as MPI provide a number of communication features including both two-sided and one-sided communication. In such models, since either of the two end processes in the connection can initiate communication, a process cannot tear-down a connection without coordinating with the remotely connected process; this can be cumbersome and expensive. On the other hand, PGAS models such as Global Arrays (GA) [8] provide a simpler communication model where a single process initiates communication in a one-sided manner (either a get/read or a put/write). Each node uses a separate thread known as the data server that contributes a part of the memory on the node to the "global address space". Processes can only get data from or put data into this global address space exposed by the data server (either over the network or through shared memory); direct process-to-process communication is not provided by this model. This restricted model ensures that connection requests are always initiated by the origin process without explicit synchronization with the target process—this makes connection tear-down simpler to handle.

Therefore, in this paper, we study the Global Arrays (GA) PGAS model for two reasons: (1) the simplicity of the model makes dynamic connection tear-down more natural to achieve; and (2) GA supports several applications that demonstrate this behavior making it an obvious first target for the proposed enhancements. We evaluate our proposed approach with NWChem [11] computational chemistry application using up to 6144 processes, and show that our approach can significantly reduce the memory requirements of the communication library while maintaining its performance. We refer to our approach as Advanced Connection Tear-down Schemes (ACTS) on InfiniBand.

The rest of the paper is organized as follows. In section II, we present the related work. In section III, we present the background of our work. In section IV, we present the design of ACTS, discussing the challenges presented by the PGAS models. In section V, we present the performance evaluation of ACTS using NWChem [11], comparing it to the state of the art implementation, ARMCI-ODCM [9]. We conclude and present our future directions in section VI. We begin with the description of the background work.

II. Related Work

Multiple approaches have been proposed for on-demand connection management with InfiniBand for MPI [12], [4], [3], [5], [6], [7]. Recently, mechanisms for on-demand connection management have been presented with PGAS models using Global Arrays [9].

Koop et al., proposed designs using multiple transport semantics with InfiniBand [1]. Using unreliable datagram connection-less transport semantics with InfiniBand, Koop et al., has proposed designs for copy based approaches [3]. Copy based approaches are applicable for MPI, since it has implicit synchronization. Using zero-copy based approaches with InfiniBand, Koop et al. also proposed using zero copy based mechanism provided by InfiniBand, since the unreliable datagram transport semantics do not support RDMA [5]. Koop et al. have also presented multi-transport InfiniBand semantics using unreliable datagram and reliable connection transport semantics [6]. Recently, work related to extended reliable connection semantics has also been proposed. Using this transport, multiple processes on the same node can share the data transfer queue, allowing memory requirements to increase corresponding to the number of nodes, rather than the number of processes [7]. Recently, we have also proposed methods for on-demand connection for Global Arrays [9]. However, none of the previously proposed approaches handle adaptive tear-down of connections, which we propose and evaluate in this paper using ACTS.

III. Background

A. InfiniBand Transport Semantics

In this section, we present a brief introduction to the InfiniBand transport semantics. We specifically focus on the reliable connection transport semantics. Reliable connection is the most popular transport semantics for designing runtime communication system over InfiniBand [3]. A variety of features including RDMA, Automatic Path Migration [13] and hardware atomics are available with this semantics, which are not available with Unreliable Datagram(UD) - the other popular transport semantics provided by InfiniBand. RC provides exactly one, in-order data delivery to the destination and exact-once notification of the data delivery to the initiator, making it an attractive choice to design communication runtime systems for MPI and PGAS models. Figure 1 shows the communication

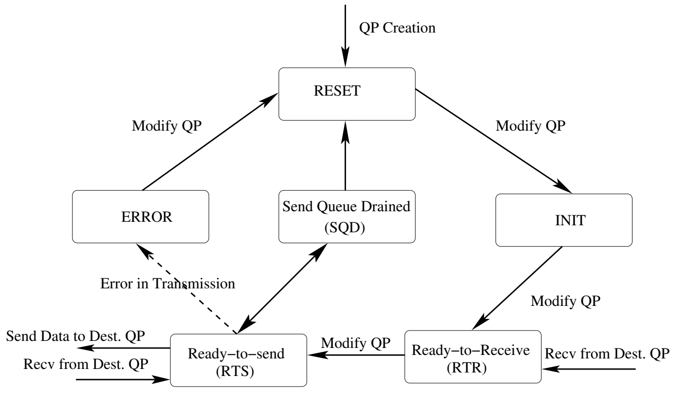

Fig. 1. QP Communication State Diagram

states and their transitions of an InfiniBand queue pair (QP) - communication channel of RC transport. A QP is created in a RESET state and assigned a unique number called qp_{num}. There are multiple state transitions which allow the QP to receive data (Ready-to-Receive (RTR)) and send data (Ready-to-Send (RTS)) from destination QP. Other state transitions (not used in ACTS design) are not explained due to lack of space.

B. Aggregate Remote Memory Copy Interface (ARMCI)

ARMCI is a communication runtime system which provides a general-purpose, efficient, and widely portable remote memory access (RMA) operations (one-sided communication) optimized for contiguous and non-contiguous (strided, scatter/gather, I/O vector) data transfers. ARMCI also provides a set of atomic and mutual exclusion operations. ARMCI leverages the low level network primitives provided by modern networks and multi-core systems. It is supported on clusters (InfiniBand, Ethernet) and high-end systems (IBM BGs, Cray XTs, Cray XE6).

C. Connection Structure with ARMCI over InfiniBand

In ARMCI, a process with lowest rank in a node is called *master* and other processes on the node are called *clients*. The master process creates a thread - *data server*, which performs one-sided operations on behalf of clients on other nodes such as (put, get, lock, accumulate), which may not be efficiently implemented using memory semantics like RDMA. Master process on a node is treated as a client by data server(s) of another node(s).

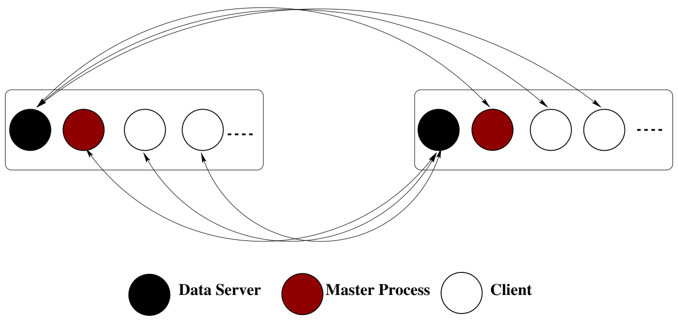

Fig. 2. Connection Establishment Pattern in ARMCI

A client establishes connection(s) to data server(s) on another node; there are no client-client connections. Each chunk of a Global Array is allocated by the master process, created as a shared memory segment, and registered with the network (if necessary). This allows each client on a node to read and write to the chunk directly. It also allows clients on other nodes to read/write directly from/to the chunk. Hence, it is necessary only to establish connections with the data server on a node, which has access to the chunk, since it is a thread of the master process. Figure 2 shows a typical connection management scenario with ARMCI.

IV. Overall Design

In this section, we present the overall design of ACTS. We begin with a description of the existing connection management approach with ARMCI over IB. Without loss of generality, we have used terms QP and connection interchangeably for the rest of the section.

A. Existing Connection Management with ARMCI over IB

The ARMCI-ODCM [9] uses the following connection management protocol, with an assumption that connection to the corresponding data server does not exist:

- Initiate a *Connection req* message to the data server, *Create connection* and transition the QP to *INIT* state
- Send the *Connection info* to the data server and wait for data server's Connection info
- Transition the connection from *INIT-RTR* and *RTR-RTS* states. Continue with the *Data transfer*.

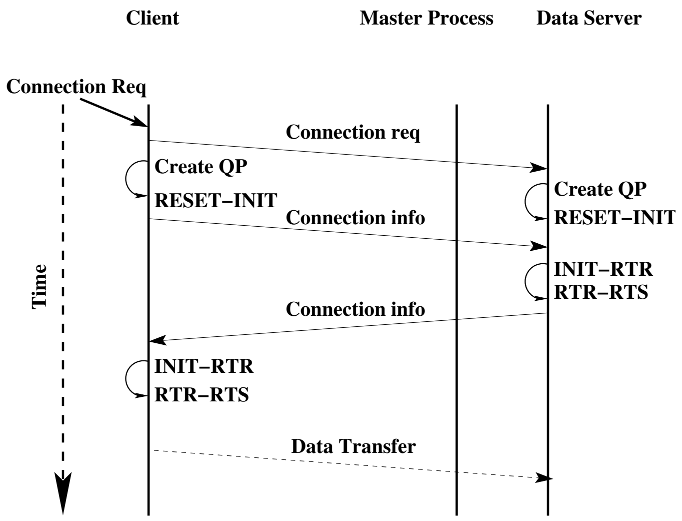

Fig. 3. ARMCI-ODCM:Connection Establishment Protocol [9]

Figure 3 shows the connection establishment protocol. The multiple phases of connection establishment are overlapped, by incurring an addition cost of *Connection req* message. As presented in our previous work, the overlap protocol reduces the overall connection establishment time significantly [9]. The overlap protocol has potential for high impact for ACTS, since it reduces the overall time for connection establishment significantly.

B. ACTS Design

In this section, we present multiple aspects of ACTS - approaches to select a victim connection and tear-down protocols for one-sided communication. We also discuss cross-cutting issues not addressed by ACTS.

1) Selecting Victim Connection: The micro-architecture literature has proposed a wide variety of approaches for selecting a victim cache line/page [14]. The primary objective is to provide best temporal and spatial locality for cache lines, while keeping the cost of searching for victim cache line minimal.

Considering the list of active connections as a cache, the objective for ACTS is to provide best temporal locality for a connection, while minimizing the cost of searching for a victim connection. We leverage the Least Recently Used (LRU) [14] connection tear-down method for selecting the victim connection. LRU tear-down method provides equal priority to the connections ordered in terms of access patterns - useful for the class of applications exhibiting quite irregular structure. Applications using dynamic load balancing and work stealing [11], and design of application kernels using task pools [10] exhibit this pattern.

However, these applications and benchmarks may exhibit a regularity in communicating to a small subset of processes The usage of collective communication primitives (scatter/broadcast/reduce for data partitioning/result collection), requesting an information typically held by a single/small collection of processes (like counters/locks), work stealing algorithm(s) which use topology information to prefer topologically closer partners, adds to this regularity. To handle this, we use a multi-queue LRU method, where queues are ordered by number of times a connection has been teared down. A victim connection is chosen from the tail of the queue with the least number of teared down connections. This approach is referred to as LRU-M for rest of the paper.

2) Low Overhead Tear-down Protocols with Overlap: PGAS models such as GA [8] provide a simple one-sided communication model, where a single process/thread initiates data transfer - making decision for connection creation/tear-down completely one-sided. Efficient tear-down protocol for two-sided message passing with MPI would require implicit synchronization. The one-sided tear-down protocol is explained here:

- If the connection exists, proceed with the data transfer
- If the connection does not exist and the overall number of connections are less than connection threshold *max conn*, proceed with ARMCI-ODCM [9], else
- proceed with the following protocol:
  - select a victim connection (using LRU or LRU-M)
  - ensure all the pending data transfers have been initiated
  - Initiate a *flush* message, to ensure the completion of outstanding data transfer operations
  - Initiate a *tear-down* message, wait for the acknowledgment and proceed with ARMCI-ODCM [9]

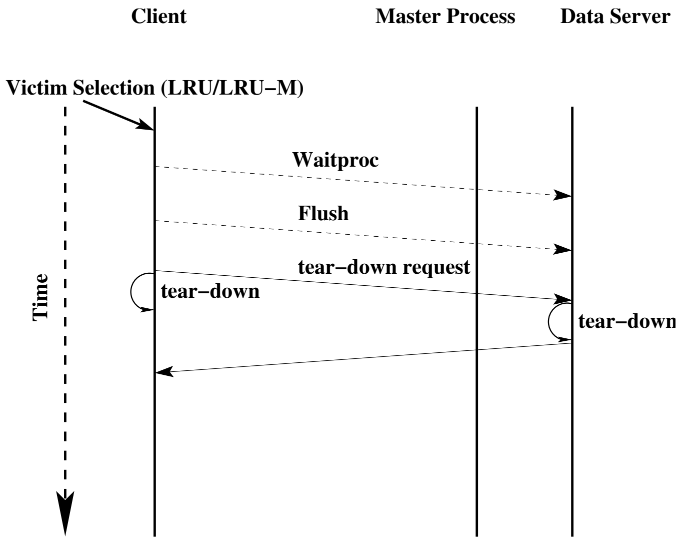

Fig. 4. ACTS Tear-down Protocol

The completion of pending data transfers is initiated by using *Waitproc* primitive provided by ARMCI [15]. It has similar semantics as MPI_Waitall - allowing the reuse of posted buffers after completion. The *flush* message ensures that all the data transfers have been completed in the target memory (through the data server and RDMA). Fence primitive provided by ARMCI is used to achieve this purpose. Figure 4 shows the ACTS tear-down protocol in detail. The line corresponding to Waitproc and flush are dotted, since they may be a no-op, if they have been explicitly executed by application/global arrays layer (using ARMCI_Waitproc and ARMCI_Fence, respectively). The request for tear-down is initiated using the UD based transport designed for ARMCI-ODCM [9]. The reliability is handled in a similar way, as presented in ARMCI-ODCM.

The process initiated the tear-down immediately after sending the tear-down request to the data server. Since tear-down takes in order of hundreds of micro-seconds, while the tear-down request message takes a couple of seconds, significant overlap is achieved in the tear-down protocol.

Other approaches for tear-down protocol include initiating local tear-down by a process, as soon as *max conn* threshold is needed. This approach would require more connections to be teared-down and created again, since following synchronization primitives (such as fence used frequently by applications) would re-create the connections, incurring significant overhead. Hence, we do not consider this approach for implementation.

C. Discussion

Impact of eXtended Reliable Connection (XRC): Koop et al., have presented MPI designs using advanced transport semantics such as XRC over InfiniBand [3], [6]. The proposed approaches allow a process to share connections to target processes co-located on a destination node. The overall connection memory footprint scales with number of nodes, rather than the number of cores.

Unlike MPI, ARMCI connection structure is asymmetric, as presented in the section III. A process creates a connection only with the data server on a target node. The processes may achieve any benefit by using XRC. However, the data server may use XRC for the processes on a remote node. The current ACTS design does not handle XRC based transport semantics,; it may be extended using the following approach. In the new protocol, when a data server receives the *tear-down* request, it checks the number of established connections with co-located processes on the remote node. The data server decrements the reference count, and destroys the connection, when the reference count is zero. We propose to implement this design as a part of our future work.

Limitation: Initiation of Connection Tear-down: The current ACTS design allows processes to request connection tear-down. A data server may not initiate a request, since a process may send a message to the data server, while the connection request has been initiated. We propose to handle this as a part of our future work.

V. Performance Evaluation of ACTS

In this section, we present a performance evaluation of NWChem [11] using ACTS, comparing its performance to ARMCI-ODCM [9] - the default implementation of ARMCI [15]. We also define *max conn* - a threshold to define the maximum number of connections a process is allowed to create. The evaluation is performed with multiple victim selection methods (LRU and LRU-M). We begin with a description of Experimental Testbed.

A. Experimental Testbed

Chinook [16] is a 160 TFlops system that consists of 2310 HP DL185 nodes with dual socket, 64-bit, Quad-core AMD 2.2 GHz Barcelona Processors. Each node has 32 Gbytes of memory and 365 Gbytes of local disk space. Communication between the nodes is performed using InfiniBand with Voltaire [17] Switches and Mellanox [18] Adapters.

B. Performance Evaluation with NWChem

NWChem is a high performance computational chemistry software package [11], designed and developed at Environmental and Molecular Science Laboratory at Pacific Northwest National Lab. The package implements a variety of lower to higher order methods for computational chemistry from density functional theory to computationally expensive methods like coupled clustered computation. It has been scaled on high-end systems achieving 1.3 Pflops on 220K cores on Cray Jaguar at Oak Ridge National Lab [19].

In previous work, it has been established that ARMCI-ODCM does not exhibit any overhead in comparison to the completely connected model [9]. Hence, ARMCI-ODCM is used as the reference point for performance comparison with ACTS. The horizontal axis on each of the performance graphs shows the connection management method. For ACTS, each of the bar is a 2-tuple: (*max conn*, (LRU/LRU-M)). The max connections are varied from 4 - 128 and number of processes are varied from 2048 - 6144. Due to large memory requirements of NWChem [11], four processes per node are used.

Figure 5 shows the performance of ARMCI-ODCM and ACTS with multiple parameters using NWChem and pentane input deck on 2048 processes. The ARMCI-ODCM implementation takes 131s. The *max conn* parameter is varied from 128-4. The overall execution time varies from 128s-142s with the variation. Using 128 as *max conn*, the overall tear-down time is .11 seconds for LRU, while it is .09 seconds for LRU-M. Similarly for *max conn* value as 32, the overall tear-down overhead is .704 s and .621 seconds for LRU and LRU-M, respectively. The reduction in *max conn* increases the overall overhead of tear-down significantly, due to increase in the number of connections being teared-down. However, the overall degradation varies from 2%-7% (calculated by subtracting the time of ACTS implementation from ARMCI-ODCM). ACTS provides significant reduction in connection memory utilization. In our experimental evaluation, we have observed that all connections are created during the execution of this input deck (results not shown due to lack of space). Hence, with *max conn* as 128, a 4-fold reduction in connection memory is achieved with 2% degradation in performance. Similarly for *max conn* as 32 and 4, a 16-fold and 128-fold improvement in connection memory is observed with a degradation of 5% and 7% in performance. While the irregular nature of execution makes it difficult to model the communication pattern, these results provide guidelines for selecting a reasonable value of *max conn* without incurring significant loss in performance.

Figure 6 shows the performance evaluation for 4096 processes. The execution time for ARMCI-ODCM is 83s. The overall execution time for various ACTS configuration varies from 82.4 to 88 seconds. The maximum number of created connections vary from 93 to 121. Hence, for 128 as *max conn*, we do not observe any connection tear-downs. For 32, the difference between LRU and LRU-M is .55 s and .49s respectively. For 4, the difference between LRU and LRU-M is 1.61s and 1.44s, respectively. As observed for 2048 processes, LRU-M outperforms LRU. The overall connection memory benefits achieved vary from 3- 4 fold, with a performance degradation of up to 7%. Hence, significant connection memory benefits are still achievable with marginal performance degradation.

Figure 7 shows the performance evaluation with 6144 processes. ARMCI-ODCM presents the overall execution time to 78s. Varying the *max conn* parameter from 128-4, the overall execution time varies from 79-84s. The observed performance degradation is up to 7%. The maximum number of created connections vary from 91-117. Hence, 128 *max conn* does not observe any tear-down of connections. The overall connection memory benefits achieved are similar to 4096 processes, as observed in the previous chart. The break time is .55s and .49s for LRU, LRU-M with 32 as *max conn*, respectively. With 4 as *max conn*, these values are 1.61 and 1.44s, respectively.

Figure 8 shows the performance evaluation the siosi7 input deck for 4096 processes. The execution time for ARMCI-ODCM is 240s. For ACTS based approaches, the overall execution time varies from 243-246s. While no connection tear-down are observed for 128, the overall connection tear-down overhead is negligible in comparison to the overall execution time.

From the performance evaluation presented above, there are multiple important conclusions:

- ACTS is able to reduce the overall memory consumption significantly, with a negligible/insignificant performance penalty. For extreme values of *max conn*, the overall overhead is up to 7%. The results presented in this section may be used by an application scientist as a guideline for the overall connection memory given to ARMCI.
- LRU-M victim selection method results in lesser or equivalent overhead in comparison to the legacy LRU method. Providing higher priority to processes, which have been teared down more often than the others, the LRU-M method is able to capture the partial regularity, which exists in the application.

VI. Conclusions and Future Work

In this paper, we have presented a dynamic time-variant connection management approach that establishes connections on-demand like previous approaches, but further intelligently tears down some of the unused connections as well. While connection tear-down itself is relevant for any programming model, different models have different complexities. In this

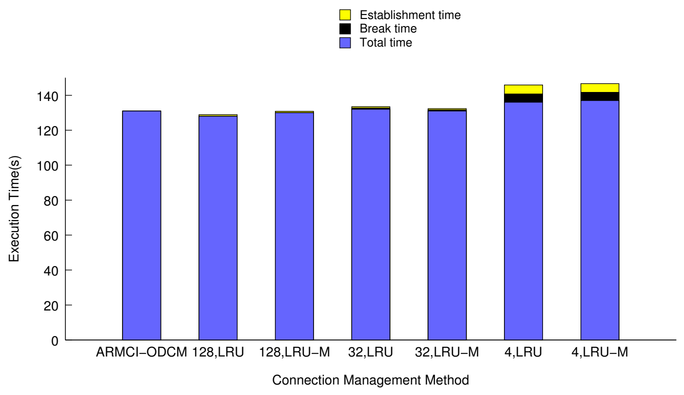

Fig. 5. NWChem-2048 Processes, Pentane Input Deck

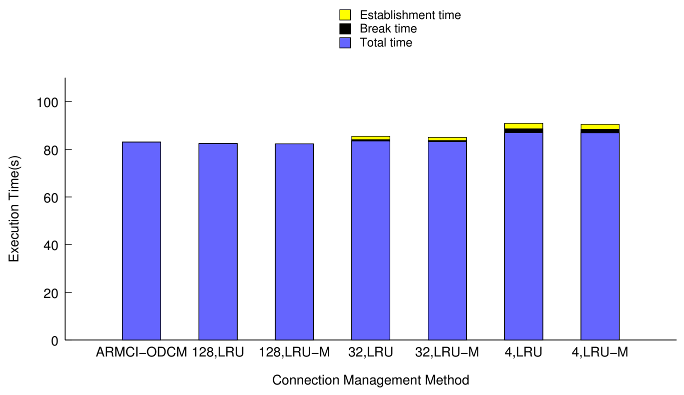

Fig. 6. NWChem-4096 Processes, Pentane Input Deck

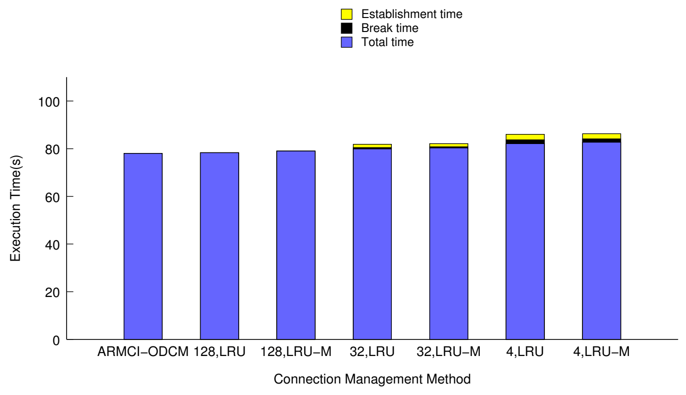

Fig. 7. NWChem-6144 Processes, Pentane Input Deck

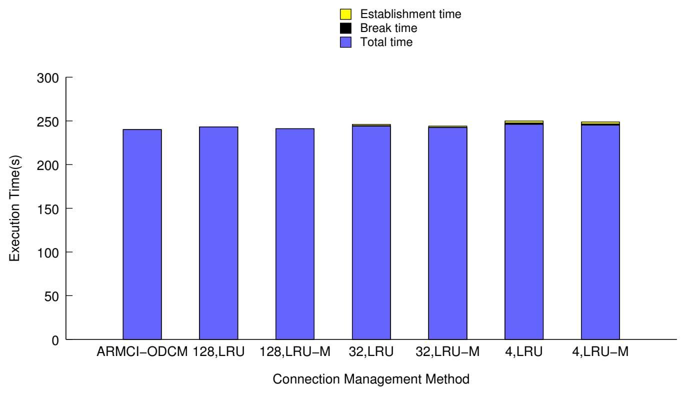

Fig. 8. NWChem-4096 Processes, siosi7 Input Deck

paper, we have studied the Global Arrays (GA) PGAS model for two reasons: (1) the simple one-sided communication primitives provided by GA and other PGAS models ensure that connection requests are always initiated by the origin process without explicit synchronization with the target process—this makes connection tear-down simpler to handle; and (2) GA supports applications in several domains such as computational chemistry (NWChem) and computational biology (ScalaBLAST) that demonstrate this behavior making it an obvious first target for the proposed enhancements. We have evaluated our proposed approach using NWChem computational chemistry application using up to 6144 processes, and concluded that our approach can significantly reduce the memory requirements of the communication library while maintaining its performance.

We continue to address the limitations of ACTS - handling multi-sided initiation tear-down requests and using eXtended Reliable Connection. We plan to address these limitations in future work and evaluate the extended approaches with NWChem and upcoming applications such as STOMP and others.

References

[1] InfiniBand Trade Association, “InfiniBand Architecture Specification, Release 1.2,” October 2004.

[2] “TOP 500 Supercomputer Sites,” http://www.top500.org.

[3] M. J. Koop, S. Sur, Q. Gao, and D. K. Panda, “High Performance MPI Design Using Unreliable Datagram for Ultra-Scale InfiniBand Clusters,” in *International Conference on Supercomputing*, 2007, pp. 180–189.

[4] W. Yu, Q. Gao, and D. K. Panda, “Adaptive Connection Management for Scalable MPI over InfiniBand,” in *International Parallel and Distributed Processing Symposium*, 2006.

[5] M. J. Koop, S. Sur, and D. K. Panda, “Zero-copy Protocol for MPI Using Infiniband Unreliable Datagram,” in *International Conference on Cluster Computing*, 2007, pp. 179–186.

[6] M. J. Koop, T. Jones, and D. K. Panda, “MVAPICH-Aptus: Scalable High-performance Multi-Transport MPI over InfiniBand,” in *International Parallel and Distributed Processing Symposium*, 2008, pp. 1–12.

[7] M. J. Koop, J. K. Sridhar, and D. K. Panda, “Scalable MPI design over InfiniBand using eXtended Reliable Connection,” in *International Conference on Cluster Computing*, 2008, pp. 203–212.

[8] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, “Global Arrays: A Portable “Shared-Memory” Programming Model for Distributed Memory Computers,” in *SuperComputing*, 1994, pp. 340–349.

[9] A. Vishnu and M. Krishnan, “Efficient On-demand Connection Management Protocols with PGAS Models over InfiniBand,” in *International Conference on Cluster, Cloud and Grid Computing*, 2010.

[10] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha, “Scalable Work Stealing,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1–11.

[11] R. A. Kendall, E. Aprà, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. Ju, J. A. Nichols, J. Nieplocha, T. P. Straatsma, T. L. Windus, and A. T. Wong, “High Performance Computational Chemistry: An Overview of NWChem, A Distributed Parallel Application,” *Computer Physics Communications*, vol. 128, no. 1-2, pp. 260–283, June 2000.

[12] J. Wu, J. Liu, P. Wyckoff, and D. K. Panda, “Impact of On-Demand Connection Management in MPI over VIA,” in *International Conference on Cluster Computing*, 2002, pp. 152–159.

[13] A. Vishnu, A. Mamidala, S. Narravula, and D. K. Panda, "Automatic Path Migration over InfiniBand: Early Experiences," in *Proceedings of Third International Workshop on System Management Techniques, Processes, and Services, held in conjunction with IPDPS'07*, March 2007.

[14] A. Aho, P. Denning, and J. Ullman, "Principles of Optimal Page Replacement," *Journal of the ACM*, vol. 1, pp. 80–93, 1971.

[15] J. Nieplocha and B. Carpenter, "ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-Time Systems," in *Lecture Notes in Computer Science*. Springer-Verlag, 1999, pp. 533–546.

[16] "Chinook SuperComputer, Environmental Molecular Science Lab, PNNL," http://emsl.pnl.gov.

[17] "Voltaire Technologies," http://www.voltaire.com/.

[18] "Mellanox Technologies," http://www.mellanox.com/.

[19] E. Aprà, A. P. Rendell, R. J. Harrison, V. Tipparaju, W. A. deJong, and S. S. Xantheas, "Liquid Water: Obtaining The Right Answer For The Right Reasons," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–7.