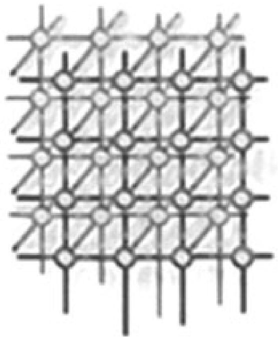
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2009; **21**:301–319
Published online 1 September 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1359



Topology agnostic hot-spot avoidance with InfiniBand

Abhinav Vishnu[1,*,†], Matthew Koop[1], Adam Moody[2], Amith Mamidala[1], Sundeep Narravula[1] and Dhabaleswar K. Panda[1]

[1]*Department of Computer Science and Engineering, 395 Dreese Labs, 2015 Neil Avenue, Columbus, OH 43210, U.S.A.*
[2]*Lawrence Livermore National Lab, 7000 East Avenue, Livermore, CA 94550, U.S.A.*

SUMMARY

InfiniBand has become a very popular interconnect due to its advanced features and open standard. Large-scale InfiniBand clusters are becoming very popular, as reflected by the TOP 500 supercomputer rankings. However, even with popular topologies such as constant bi-section bandwidth Fat Tree, hot-spots may occur with InfiniBand due to inappropriate configuration of network paths, presence of other jobs in the network and un-availability of adaptive routing. In this paper, we present a hot-spot avoidance layer (HSAL) for InfiniBand, which provides hot-spot avoidance using path bandwidth estimation and multi-pathing using LMC mechanism, without taking the network topology into account. We propose an adaptive striping policy with batch-based striping and sorting approach, for efficient utilization of disjoint network paths. Integration of HSAL with MPI, the *de facto* programming model of clusters, shows promising results with collective communication primitives and MPI applications. Copyright © 2008 John Wiley & Sons, Ltd.

Received 27 November 2007; Revised 30 March 2008; Accepted 7 May 2008

KEY WORDS: InfiniBand; MPI; hot-spot; Fat Tree; LMC; collective communication

1. INTRODUCTION

In the last decade or so, *cluster computing* has become popular by providing the users with an excellent price to performance ratio [1]. High-speed commodity interconnects have become popular,

*Correspondence to: Abhinav Vishnu, Department of Computer Science and Engineering, 395 Dreese Labs, 2015 Neil Avenue, Columbus, OH 43210, U.S.A.
†E-mail: vishnu@cse.ohio-state.edu

Contract/grant sponsor: Publishing Arts Research Council; contract/grant number: 98-1846389

InfiniBand [2] in particular, due to its advanced features and open standard. Parallel applications executing on these clusters primarily use MPI [3,4] as the *de facto* programming model. Fat Tree [5] has become a very popular interconnection topology for these clusters, primarily due to its multi-pathing capability. However, even with constant bi-section bandwidth (CBB) Fat Tree, hot-spot(s) may occur in the network depending upon the route configuration(s) between end nodes and communication pattern(s) in an application. Other factors including the presence of other jobs in the network and topology un-aware scheduling of tasks by program launchers may significantly impact the performance of applications. To make the matters worse, the deterministic routing nature of InfiniBand limits the application from an effective use of multiple paths transparently to avoid the hot-spot(s) in the network. InfiniBand specification [2] provides a congestion control protocol, which leverages an early congestion notification mechanism between switches and adapters. However, this approach enforces a reduced data transfer on the existing path, while other paths in the network may be left under-utilized.

A popular mechanism for providing hot-spot avoidance is to leverage multi-pathing. In our previous studies, we provided a framework for supporting multi-pathing at the end nodes [6], popularly referred to as multi-rail networks. We provided an abstraction for supporting multiple ports and multiple adapters and studied various scheduling policies. The evaluation at the MPI layer provided promising results with collective communication and applications. We expanded the basic ideas proposed in this study to provide hot-spot avoidance using the LMC mechanism [7]. Using the adaptive striping policy and the LMC mechanism, we studied the performance with collective communication primitives and MPI applications. We observed that HSAM (hot-spot avoidance with MVAPICH [8]) is an efficient mechanism for providing hot-spot avoidance.

In this study, we thoroughly review our previous proposals and alleviate the deficiencies of HSAM. The inherent limitations of the HSAM design prohibit the utilization of all the physically disjoint paths at run time. As a result, better paths may never be even explored. In this paper, we present a hot-spot avoidance layer (HSAL), which performs batch-based striping and sorting (BSS) during the application execution to adaptively eliminate the path(s) with low bandwidth. The design challenges for integration of MPI with HSAL are also discussed in detail. We also compare the HSAM [7] scheme with MPI integrated with HSAL to compare the performance of different BSS configurations and the original case (no multi-pathing at all). Using MPI_Alltoall, we can achieve an improvement of 27 and 32% in latency with different BSS configurations compared with the best configuration of the HSAM scheme on 32 and 64 processes, respectively. A default mapping of tasks in the cluster shows similar benefits. Using the Fourier transform benchmark from NAS Parallel Benchmarks [9] with different problem sizes, the execution time can be improved by 5–7% with different BSS configurations compared with the best HSAM configuration and 11–13% from the original implementation. Other NAS Parallel Benchmarks [9] do not incur any performance degradation.

The remainder of this paper is organized as follows: In Section 2, we present the background of our study. In Section 3, we present the motivation of our study. Section 4 discusses the HSAL, associated challenges and integration at the MPI layer. In Section 5, we present the performance evaluation of the BSS approach using an InfiniBand cluster. In Section 6, we present the related study. In Section 7, we conclude and present our future directions.

2. BACKGROUND

This section presents the background information of our study. To begin with, an introduction to InfiniBand is provided. This is followed by an introduction to MPI [3,4], including two-sided point-to-point communication and collective communication primitives.

2.1. Overview of InfiniBand

The InfiniBand architecture [2] defines a switched network fabric for interconnecting processing nodes and I/O nodes. An InfiniBand network may consist of switches, adapters (called Host Channel Adapters (HCAs)) and links for communication. InfiniBand supports different classes of transport services: (reliable connection (RC), unreliable connection, reliable datagram and unreliable datagram). RC transport mode supports remote direct memory access (RDMA), which makes it an attractive choice for designing communication protocols. We use this transport for this study. Under this model, each process pair creates a unique entity for communication, called *queue pair* (*QP*). Each QP consists of two queues: *send queue* and *receive queue*. The requests to send the data to the peer are placed on the send queue, by using a mechanism called *descriptor*. A descriptor describes the information necessary for a particular operation. For RDMA operation, it specifies the local buffer, address of the peer buffer and access rights for manipulation of the remote buffer. InfiniBand also provides a mechanism, where different queue pairs can share their receive queues, called *shared receive queue* mechanism. The completions of descriptors are posted on a queue called *completion queue*. This mechanism allows a sender to know the status of the data transfer operation. Different mechanisms for notification are also supported (polling and asynchronous).

InfiniBand defines an entity called *subnet manager*, which is responsible for discovery, configuration and maintenance of a network. Each InfiniBand port in a network is identified by one or more local identifiers (LIDs), which are assigned by the subnet manager. As InfiniBand supports only destination-based routing for data transfer, each switch in the network has a routing table corresponding to the LID(s) of the destination. However, deterministic routing nature of InfiniBand limits the intermediate switches to route the messages adaptively. To overcome this limitation, InfiniBand provides a mechanism, LID mask count (LMC), which can be used for specifying multiple paths between every pair of nodes in the network. The subnet manager may be specified with different values of LMC mechanism (0–7), creating a maximum of 128 paths. Leveraging the LMC mechanism to avoid the hot-spot(s) in the network is the focus of this paper.

2.2. Overview of MPI protocols

Message passing interface (MPI) [3,4] defines multiple communication semantics. The two-sided communication semantics [3] has been widely used in the last decade or hence for writing a majority of parallel applications. Two-sided communication semantics are broadly designed using the following protocols:

- *Eager protocol*: In the eager protocol, the sender process eagerly sends the entire message to the receiver. In order to achieve this, the receiver needs to provide sufficient buffers to handle

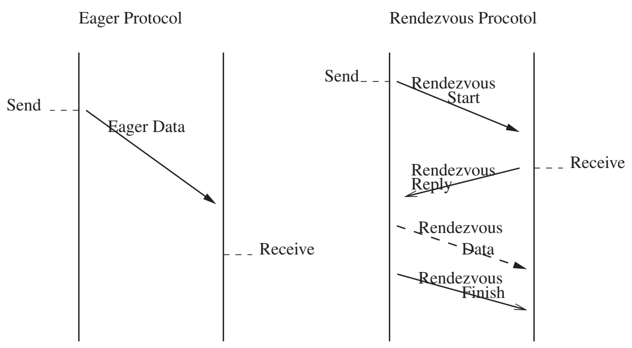


Figure 1. MPI communication protocols.

incoming messages. This protocol has minimal startup overhead and is used to implement low latency message passing for smaller messages.

- *Rendezvous protocol*: The rendezvous protocol negotiates the buffer availability at the receiver side before the message is actually transferred. This protocol is used for transferring large messages when the sender is not sure whether the receiver actually has sufficient buffer space to hold the entire message.

Figure 1 illustrates these protocols. RDMA is used for the data transfer with the rendezvous protocol. The application buffer(s) need to be registered so that the operating system does not swap them during communication.

2.3. Collective communication

In this section, we describe the collective communication primitives in brief. We specifically focus on the primitives that are provided by the MPI specification. We assume that the MPI job has *proc* number of processes.

MPI_Alltoall is used for all-to-all personalized exchange between MPI processes. Each process has a separate message for every other process. The popular algorithms include pair-wise exchange and direct communication. As each process communicates with every other process, it takes $proc-1$ steps for each of these algorithms. MPI_Allgather is used for all-to-all broadcast. Each process has a message for every other process; however, the message is not different for every other process, unlike MPI_Alltoall described above. The popular algorithms are ring algorithm and pair-wise exchange algorithm. In the ring algorithm, each process sends a message to its neighbor and receives a message from the other neighbor in the ring. As a result, this algorithm takes $proc - 1$ steps. A detailed description of various collective communication primitives has been discussed by Kumar *et al*. [10].

3. MOTIVATION

Many researchers have proposed mechanisms for alleviating network hot-spots with Fat Tree networks [5,11]. The efficacy of this topology for different communication patterns has been

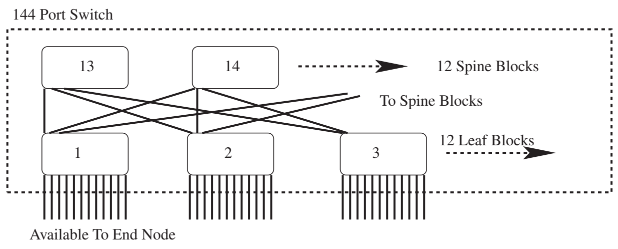


Figure 2. 144-port InfiniBand switch block diagram.

shown. However, the primary assumption with the studies is availability of partially adaptive routing mechanisms provided by the network or knowledge of a pre-defined communication pattern of application among processes. Petrini *et al*. have shown that *a priori* communication may lead to the occurrence of hot-spots in the network [11]. InfiniBand routing mechanism is deterministic. Mechanisms to overcome this limitation has been proposed by Duato *et al.* using splitting of switch buffers, utilization of InfiniBand VLs and BFS-based discovery of InfiniBand subnet [12,13]. However, the current generation InfiniBand hardware does not support these mechanisms.

To present the current state of the problem, we use a CBB switch, as shown in Figure 2. Each switch block consists of 24 ports. The leaf switches (referred to as leaf blocks from here onwards) have 12 ports available to be used by the end nodes, and the other 12 ports are connected to the spine switches (referred to as spine blocks from here onwards). In the figure, blocks 1–12 are leaf blocks and blocks 13–24 are spine blocks. The complete switch has 144 ports available for end nodes. Each block is a crossbar in itself.

To demonstrate the contention, we take a simple MPI program, which performs ring communication with neighbor rank increasing at every step. The communication pattern is further illustrated in the Figure 3 (only $step_1$ and $step_2$ are shown for clarity). Executing the program with n processes takes $n - 1$ steps. Let $rank_i$ denote the rank of the ith process in the program, and $step_j$ denote the jth step during execution. At $step_j$, an MPI process with $rank_i$ sends data to $rank_{i+j}$ and receives data from $rank_{i-j}$. This communication pattern is referred to as *displaced ring communication (DRC)* for the rest of the paper.

We take an instance of this program with 24 processes and schedule MPI processes with $rank_0$–$rank_{11}$ on nodes connected to block 1 and $rank_{12}$–$rank_{23}$ to block 2. We use MVAPICH [8], a popular MPI over InfiniBand, as our MPI implementation for the evaluation of DRC. As each block is a crossbar in itself, no contention is observed for intra-block communication. However, as the step value increases, the inter-block communication increases and a significant link contention is observed. The link contention observed during $step_{12}$ (each process doing inter-block communication) is shown in Figure 4, with thicker solid lines representing more contentions. The quantitative evaluation is shown in Figure 5.

From Figure 4, we can see that some links are over-used to a degree from four to zero. As the degree of link usage increases, the bandwidth is split among the communication instances using the link(s), making them *hot-spots*. In our example, paths using block 13 split bandwidth for four different communication instances making the set of links using this block hot-spots. In Figure 5,

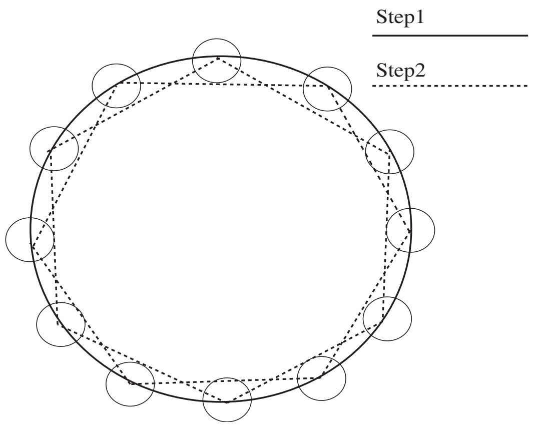


Figure 3. Communication steps in DRC.

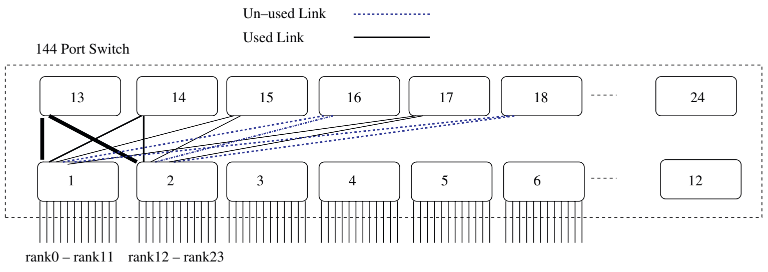


Figure 4. Link usage with DRC.

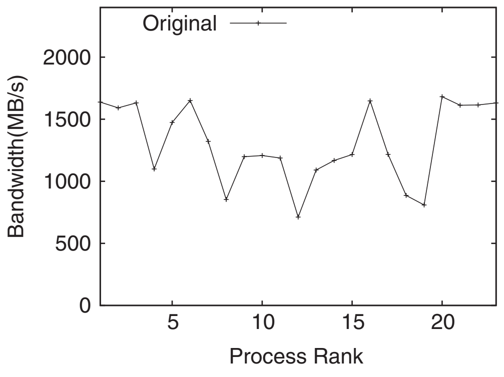


Figure 5. DRC, 24 processes.

we show the results of our evaluation. The bandwidth observed by $rank_0$ to different processes is reported. We observe that for $rank_{12}$ communication, the bandwidth observed is the least by the process. At this point, all communication instances are inter-block.

An important observation is that the presence of a CBB does not necessarily provide contention-free routes for DRC. Lack of fully/partial adaptive routing support in the network leads to inefficient usage of the links. Other topologies with over-subscription of links are likely to face similar or worse performance degradation. Such topologies include the 3-D tori and oversubscribed Fat Trees for larger-scale clusters, which would be the norm for upcoming InfiniBand clusters.

In the following section, we present the design of HSAL of InfiniBand, which alleviates this problem. It leverages InfiniBand multi-pathing mechanism and support from the Subnet Manager. Reliable connection transport semantics are used for better estimation of path bandwidth and discussion on scheduling policies is presented.

4. BASIC ARCHITECTURE

In this section, we present the basic architecture of HSAL for providing network topology agnostic hot-spot avoidance with InfiniBand clusters. Figure 6 shows it further. The architecture comprises three main components: *communication scheduler*, *scheduling policies* and *completion filter*.

The communication scheduler is the central part of our design. It interfaces with the upper-level applications for scheduling the data on multiple paths available in the network. The upper-level application may include programming models (MPI [3,4], distributed shared memory, global arrays and other PGAS languages). The policy used for data transfer to the peer is based on the scheduling policy specified by the user. Scheduling policies may themselves be static or dynamic [6]. We discuss the adaptive striping policy in detail, which uses a feedback mechanism from the network for path bandwidth estimation. The architecture also consists of a completion filter, which is responsible for processing of data transfer completions. As discussed in Section 2, the data transfer with InfiniBand provides a completion on the sender side. In this paper, we leverage this mechanism for path bandwidth estimation to provide hot-spot avoidance.

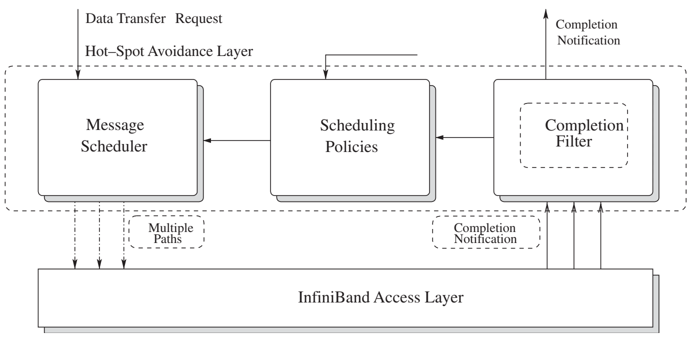


Figure 6. Basic Architecture of HSAL.

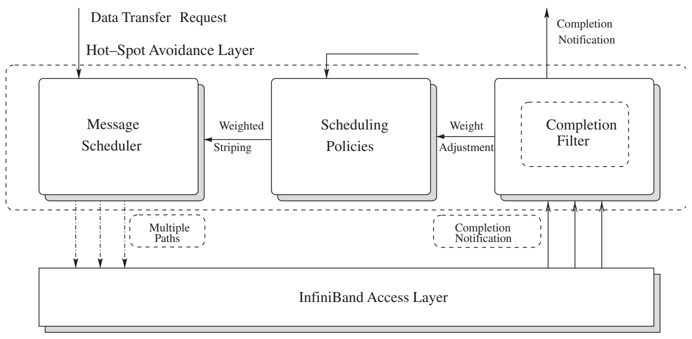


Figure 7. Feedback loop in adaptive striping.

4.1. Adaptive striping

In this section, we present the adaptive striping policy. In our previous studies [6,14], we provided initial discussion on scheduling policies for small and large messages. Adaptive striping policy primarily focuses for large messages, as the bandwidth estimation using small messages is not very accurate.

In adaptive striping scheme, we assume that the latencies of the paths are similar and focus on their bandwidth. The primary goal is to stripe the data in such a manner that all the stripes reach the destination at the same time. As multiple paths may have different bandwidths, the stripe sizes may be different. A static approach for assigning weights to different paths fails to address the changes in the state of contention and traffic in the network. To address this issue, we leverage the completion notification mechanism of InfiniBand, which is generated after each message delivery and an acknowledgment is received on the sender side. With the help of completion filter, the progress engine of our HSAL uses polling to check any new completion notifications and mark the timings of their arrival. Hence, the adaptive striping policy uses a feedback mechanism to estimate different path bandwidths. This feedback-based control mechanism is illustrated in Figure 7. In the following section, we discuss the limitations of HSAM [7] and present the BSS scheduling policy, which is a refined form of adaptive striping policy. Under this policy, a subset of paths is used at every step, providing a better estimation of path bandwidth and potentially better path utilization.

4.2. The HSAM design and limitations

To begin with, in this section we discuss the HSAM approach in brief [7]. Under this approach, we use the LMC mechanism for creation of multiple physical paths and use the adaptive striping policy described above for data transfer. However, the inherent design of HSAM imposes some practical constraints:

- Sending a message stripe through each path requires posting a corresponding descriptor. Hence, this may lead to significant startup overhead with increasing number of paths.
- For each message stripe, a completion is generated on the sender side. With increasing number of paths, more completions need to be handled, which can potentially delay the progress of the application.

- The accuracy of path bandwidth is significantly dependent upon the discovery of the completions, as mentioned in the scheduling policies sections. With increasing number of paths, the accuracy may vary significantly.

4.3. BSS scheduling policy

To overcome these problems, we propose the BSS scheduling policy. Let n_{paths} represent the total number of paths between every pair of processes and n_{batch} represent the batch of paths used during a communication instance for a message. Each BSS configuration is represented as a 2-tuple: (n_{paths}, n_{batch}), where $n_{batch} \leq n_{paths}$. These values can be specified by the user. However, as discussed above, increasing n_{paths} beyond the number of physically disjoint paths may not be beneficial. Let W_{total} represent the aggregated weight of n_{paths} and w_i represent the weight of the ith path between a pair of processes. To begin with, we initialize all the paths with equal weights:

$$w_i = \frac{W_{total}}{n_{paths}} \tag{1}$$

As HSAL is topology and communication pattern agnostic, this weight assignment reflects the generic nature of HSAL. This by no means is a limitation of the framework and other initial weight assignments may be plugged in as well. For every communication instance, BSS selects the first n_{batch} of paths from a non-decreasing weight sorted array. The data are striped on these paths proportional to their weights. Let W_{batch} denote the total weight of the paths in the batch and M be the size of the data to be transferred. The ith path sends $w_i/W_{batch} \cdot M$ amount of data. Let t_i denote the time taken by the stripe on the ith path. As mentioned earlier, t_i is calculated using the data delivery notification mechanism of RC transport model. The updated weight w_i' of the path is represented by the following equation:

$$w_i' = W_{batch} \cdot \frac{\frac{w_i}{t_i}}{\sum_{k \in batch} \frac{w_k}{t_k}} \tag{2}$$

The variance in bandwidth estimation is alleviated using a linear model for updating paths:

$$w_i' = (1 - \alpha) \cdot w_i + \alpha \cdot W_{batch} \cdot \frac{\frac{w_i}{t_i}}{\sum_{k \in batch} \frac{w_k}{t_k}} \tag{3}$$

To enable faster convergence on the paths to use, we use a higher value of α. In performance evaluation, we use 0.95 as the value of α. Once the paths are updated, the array of path weights is sorted again. We also note that each communication instance impacts only the weights of the paths that have been used during the communication. However, it does affect the ordering of the paths to be used for next iteration. In this regard, our weight updation policy is rather a heuristic, where the optimal algorithm would update the local weights keeping global weights into account. However, the latter approach is not scalable, as it requires addition global exchange of weight arrays. Hence, we implement only the heuristic for our evaluation.

4.3.1. Detailed design issues

In this section, we present the detailed design issues with the BSS algorithm.

- *Selecting the number of paths*: In Section 3, we presented the block diagram of the switch used in our performance evaluation. A maximum of 12 physically independent paths are available between every pair of nodes. Using the *ibtracert* mechanism provided by the InfiniBand access layer, we have concluded that the subnet manager is able to configure these paths for usage. Hence, we use 12 as the maximum number of paths with BSS. In addition, we also evaluate other BSS configurations (4, 2), (8, 2) and (12, 3).
- *Scalability issues*: The scalability issues with RC transport model have been a focus of research for many researchers [8,15]. Most of the researchers have focused on scalability issues with 'no-multi-pathing' case. Multi-pathing requires creation of multiple QPs, which aggravates this issue significantly. Although it is beyond the scope of this paper, we hereby briefly mention possible solutions for alleviating these problems:
    - On-demand connection management with unreliable datagram-based approach for small messages and RC for large messages.
    - Maintaining an upper bound on the number of RC queue pairs. This would provide an upper bound on the connection memory taken by the MPI library [8,15].
    - ConnectX [16], the next generation InfiniBand, promises to provide QP sharing benefits for processes on the same node. This would be beneficial for clusters based on many-core architectures and reduce the connection memory consumption significantly.

4.3.2. Integration with MPI

In this section, we present the integration of HSAL with MPI, the *de facto* programming model used in the current clusters. Much of the HSAL design is programming model independent. However, some of the design parameters need modification at the MPI layer for efficient usage of HSAL. In the upcoming sections, we discuss the detailed design issues associated with this integration. This is further reflected in Figure 8.

- *Multiple RDMA completion notifications*: In our design, bulk data transfer in the rendezvous protocol is striped into multiple smaller messages. Hence, multiple completion notifications may be generated for a single message at the sender side. The completion filter component in our design notifies the MPI protocol layer only after it has collected all the notifications.

  At the receiver, the MPI protocol layer also needs to know when the data message has been transferred into the destination buffer. In our original design, this is achieved by using a *rendezvous finish* control message. This message will be received only after the rendezvous data messages, as ordering is guaranteed for a single InfiniBand connection. However, this scheme is not enough for multiple paths. In this case, we have to use multiple rendezvous finish messages—one per each path where rendezvous data are sent. The receiver notifies the MPI protocol layer only after it has received all the rendezvous finish messages. It should be noted that these rendezvous finish messages are sent in parallel and their transfer times are overlapped. In general, they incur negligible overhead.

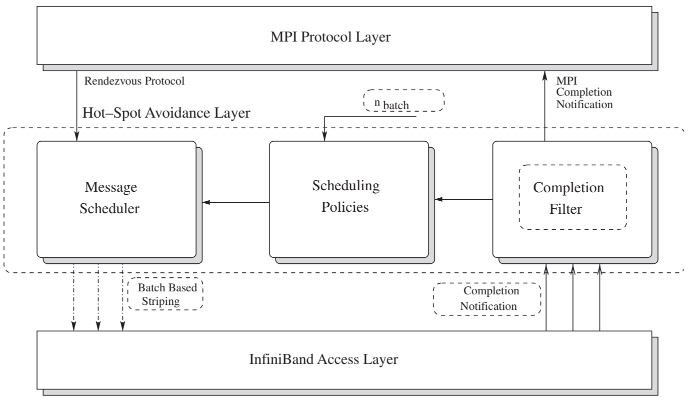


Figure 8. BSS-based MPI over HSAL.

- *Out-of-order messages*: Owing to the requirement of MPI-1 two-sided communication, it is desirable to process messages in order in our multi-rail implementation. As we use RC service provided by InfiniBand for each path, messages for a single path are delivered in order. However, there is no ordering guarantee in the presence of multiple paths. To address this problem, we introduce a *packet sequence number* (PSN) variable shared by multiple paths used between a pair of processes. For convenience, we refer to this set of multiple paths as a connection. Every message sent through this connection carries current PSN and also increments it. Each receiver maintains an *expected sequence number* (ESN) for every virtual channel. When an out-of-order message is received, it is enqueued into an *out-of-order queue* associated with this connection and its processing is deferred. This queue is checked at proper times when a message in the queue may be the next expected packet.

  The basic operations on the out-of-order queue are *enqueue*, *dequeue* and *search*. To improve the performance, it is desirable to optimize these operations. In practice, we have found that out-of-order messages are rare and incur negligible overhead.
- Our design focuses on data transfer involving large messages. For small messages, software-based approaches for hot-spot measurement are not feasible. In the future, we plan to work on the optimizing collective communication primitives for small messages. Each rendezvous data transfer involves a send-handle and receive-handle data structure on the sender and receiver sides, respectively. These data structures are used for notification upon the completion of data transfer on the sender and receiver sides. Compared with the HSAM [7] scheme, each data transfer in the BSS scheduling policy involves different W_{batch} values. To address this issue, we piggyback W_{batch} value with the rendezvous start message. This is further illustrated in Figure 9. The receive-handle stores this value at the receipt of the rendezvous start message. As RDMA is used for actual data transfer, the receiver is unaware of the individual data size written to its memory by different stripes. Hence, we piggyback the $w_i/W_{batch} \cdot M$ information with the ith finish message. Once the aggregated value of weight received from different finish messages is equal to the W_{batch}, the MPI protocol layer is notified of the completion of data transfer.
- *Efficacy of adaptive striping*: In our implementation, the start times of all stripes are almost the same and can be accurately measured. However, completion notifications are generated by

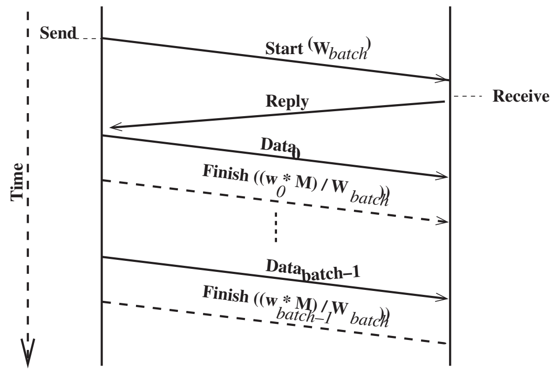

Send
Start (W_{batch})
Receive
Reply
$Data_0$
Finish $((w_0 * M) / W_{batch})$
$Data_{batch-1}$
Finish $((w_{batch-1} * M) / W_{batch})$
Time

Figure 9. HSAL over rendezvous protocol.

the InfiniBand layer asynchronously and we record only the finish time of a stripe as we have found its completion notification. As the progress engine processing can be delayed due to application computation, we can obtain only an upper bound of the actual finish time and the resulting delivering t_i is also an upper bound. Therefore, one question is how accurately we can estimate the delivering time t_i for each path. To address this question, we consider three cases:

1. Progress engine is not delayed. In this case, accurate delivering time can be obtained.
2. Progress engine is delayed and some of the delivering times are overestimated. In this case, weight redistribution will not be optimal, but it will still improve performance compared with the original weight distribution.
3. Progress engine is delayed for a long time and we find all completion notifications at about the same time. This will essentially result in no change in the weight distribution.

We can see that in no case the redistribution results in worse performance than the original distribution. In practice, case 1 is the most common and accurate estimation is expected. This is typically true for synchronizing collective communication primitives including, MPI_Alltoall, MPI_Allgather, DRC and MPI_Allreduce.

5. PERFORMANCE BENEFITS WITH MPI OVER HSAL

In this section, we evaluate the performance of the BSS policy using a 64-node InfiniBand cluster. Different configurations of the BSS policy ((4, 2), (8, 2) and (12, 3)) are evaluated and compared with the HSAM [7] design. Although more combinations are possible, we show the results for best value of n_{batch}, keeping n_{paths} constant. In Section 4, we mentioned that HSAM is a special case of the BSS scheme, when n_{paths} and n_{batch} are the same. For clarity, we refer to this configuration as the HSAM [7] scheme. In our previous study, we concluded that the HSAM scheme with 4 paths is the best configuration [7]. We also compare the performance of the BSS scheme with no multi-pathing case, the scenario commonly used in most MPI implementations over InfiniBand. This is referred as 'Original' in the performance graphs, unless otherwise mentioned. Our evaluation consists of three parts. In the first part, we use Intel MPI Benchmark [17] for collective communication. In the second part, we focus on NAS Parallel Benchmarks [9], particularly, Fourier transform, which primarily uses MPI_Alltoall. We begin with a brief description of our experimental testbed.

5.1. Experimental testbed

Our testbed cluster consists of 64 nodes: 32 nodes with Intel EM64T architecture and 32 nodes with AMD Opteron architecture. Each node with Intel EM64T architecture is a dual socket, single core with 3.6 GHz, 2 MB L2 cache and 2 GB DDR2 533 MHz main memory. Each node with AMD Opteron architecture is a dual-socket, single core with 2.8 GHz, 1 MB L2 cache and 4 GB DDR2 533 MHz main memory. On each of these systems, the I/O bus is $\times 8$ PCI-Express with Mellanox [16] MT25208 dual-port DDR Mellanox HCAs attached to 144-port DDR Flextronics switch. The firmware version is 5.1.400. We have used open fabrics enterprise distribution version 1.1 for evaluation on each of the nodes and OpenSM as the subnet manager, distributed with this version.

5.2. Performance evaluation with collective communication

In this section, we present the evaluation of BSS, HSAM and Original for various collective communication patterns. We use MPI_Alltoall, MPI_Allgather and MPI_Allreduce for collective communication. We also evaluate different mappings of process ranks to nodes in the network:

- *Sequential mapping*: The processes are mapped to the nodes in a sequential manner. As an example, let the ith output switch port be represented by $port_i$, as discussed in Section 3. A process with MPI rank i is scheduled on $port_i$. This is referred as SM in the rest of the section.
- *Default mapping*: The processes are assigned randomly to the nodes. This is also the default behavior of various program launchers (multi-purpose daemon is an example used by MVAPICH2). However, for a consistent comparison between different configurations of BSS, HSAM and Original implementations, the same mapping is used. Default mapping also represents the nodes assigned to a job, due to fragmentation in the cluster aggravated by the completion of previous jobs. This is referred as DM in rest of the section.

Figure 10 shows the results for MPI_Alltoall with 48 processes. Pair-wise exchange algorithm is used for MPI_Alltoall [10]. However, the exchange partner for the non-power-of-2 case (48 processes) is different from the power-of-2 case (64 processes). We see that BSS (12, 3) performs the best, reducing the latency to half in comparison with the original case. Compared with the HSAM, latency decreases by 27%. Compared with the BSS (4, 2) case, the improvement in performance is 15%. The improvement in performance is due to the adaptation of path weights at the run time by the BSS scheme. The HSAM scheme is also able to adapt the path weights, but it does not use all the paths and ends up with sub-optimal paths for usage. Figure 11 shows the results for MPI_Alltoall with a default mapping of processes. We see that the benefits are significant compared with the Original case as well as HSAM. Hence, BSS provides benefits with a different scheduling of processes.

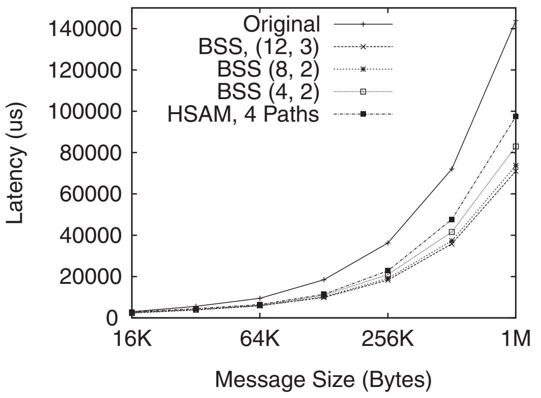


Figure 10. MPI_AlltoAll (48 × 1), SM.

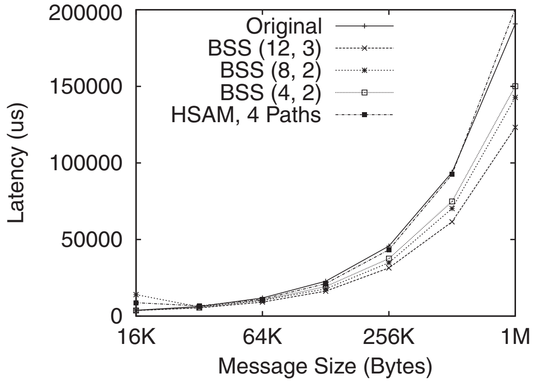


Figure 11. MPI_AlltoAll (64 × 1), DM.

Figure 12 shows the results for MPI_Allgather with 48 processes. For the message sizes presented in these figures, MPI_Allgather uses the ring algorithm [10]. Looking from the topology perspective, each switch block has exactly one out-bound and in-bound communications. Rest of the communication is within a leaf block. As a result, insignificant contention is observed, and all the cases perform similarly. However, under random allocation of nodes to a job, the number of in-bound and out-bound communication instances increases and the contention increases significantly. Figure 13 shows the results for such a scenario. For 64 processes, default distribution of processes leads to significant contention. As a result, the improvement compared with the Original and the HSAM scheme is 43 and 32%, respectively.

Figures 14 and 15 show the results for DRC with 24 and 64 processes, respectively. We see that significant improvement is observed in all cases compared with the HSAM case. Although not shown here, the Original implementation performs much worse than HSAM.

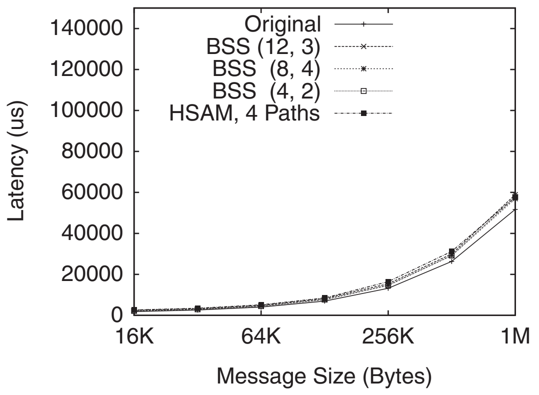


Figure 12. MPI_Allgather (48 × 1), SM.

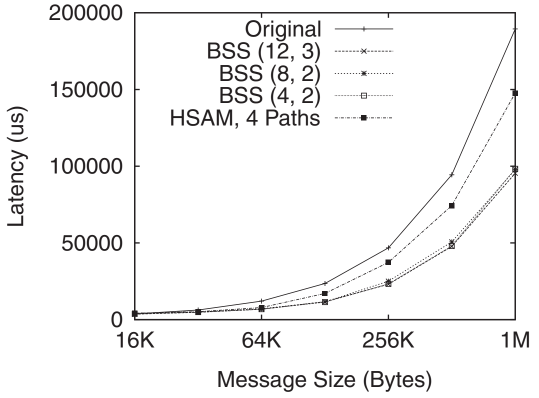


Figure 13. MPI_Allgather (64 × 1), DM.

5.3. Performance evaluation with MPI applications

Figures 16 and 17 represent the results for NAS Parallel Benchmarks [9] with Fourier transform with Class B and Class C problem size, respectively. In all experiments, we use only the SM. As seen in the previous section, the SM of processes produces the least performance improvement. We expect that the benefits shown with the SM of processes are the least a job will achieve for any mapping of processes.

Fourier transform benchmark uses collective communications primitives such as MPI_Alltoall, MPI_Reduce and MPI_Bcast. For MPI_Alltoall, we saw significant performance benefits in the previous section. We note that the benefits are transferred to the Fourier transform benchmark.

For Class B, compared with the Original case, different versions of the BSS scheme show performance benefits ranging from 10 to 11% in the execution time. Different configurations of BSS further improve the execution time by 6–7%. Similar improvements are seen for both 32 and

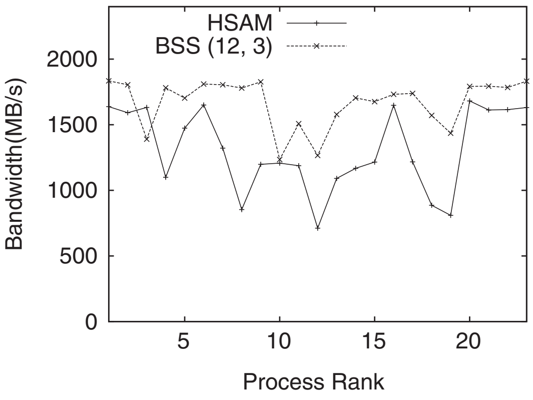


Figure 14. DRC (24 × 1), SM.

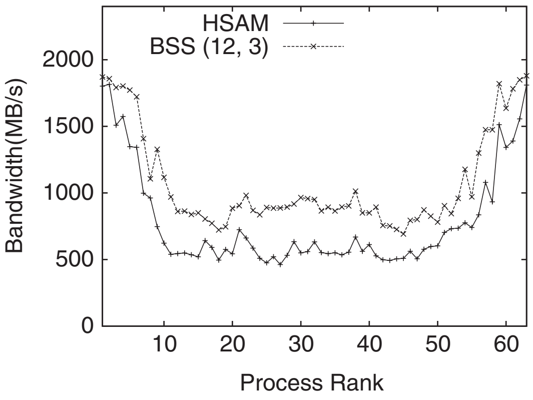


Figure 15. DRC (64 × 1), SM.

64 process cases. For Class C problem size, BSS configurations perform similarly. Compared with the original case, BSS configurations improve the execution time by 13% for 32 process case and 6.5% compared to the HSAM scheme. Similar performance improvement is observed with the 64 process case.

6. RELATED STUDY

A popular mechanism to alleviate hot-spots in the network is to leverage multi-rail clusters [6,14,18,19]. Coll *et al.* have proposed the general algorithms for utilization of multi-rail clusters and studied their benefits with point-to-point and collective communication primitives [18]. Liu *et al*. have studied the designs for providing support for multi-rail InfiniBand clusters with two-sided communication [6]. To support MPI-2 one-sided communication, enhanced designs have been

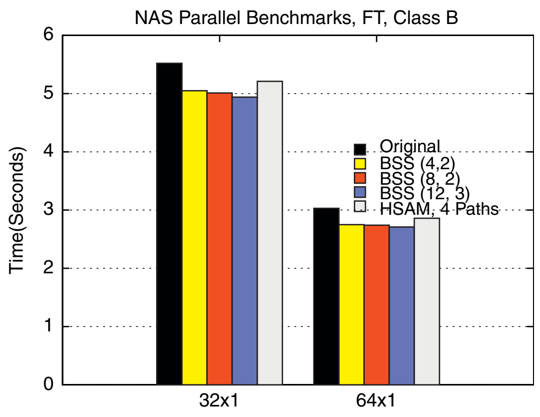


Figure 16. NAS Benchmarks, FT, Class B.

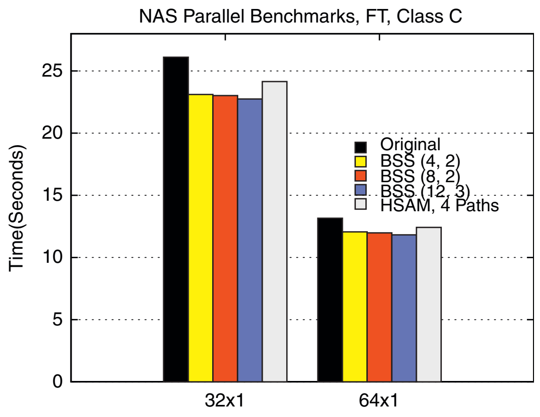


Figure 17. NAS Benchmarks, FT, Class C.

proposed by Vishnu *et al.* [14,20]. IBM HPS architecture supports multiple adapters with connectivity to different switches. An initial study for avoiding congestion, and fail-over has been studied by Sivaram *et al.* [19]. Petrini *et al*. have also studied the hardware features with Quadrics [21] Interconnection network for providing congestion control [11,22]. Similar studies have been performed with Myrinet [23] for providing congestion control [24]. The current generation of Myrinet clusters leverages *dispersive routing* to avoid hot-spots. Under this mechanism, a random selection of path is done during data transfer.

Recently, many researchers have focused on providing congestion control with InfiniBand networks [25–27]. The study proposed by Santos *et al*. has also been accepted with InfiniBand specification [2]. The primary idea of the above studies is to generate an explicit congestion notification mechanism to the destination (early forward congestion notification mechanism). The destination sends a early backward congestion notification to the source. Mechanisms for rate control and early rate increase upon recovery from congestion have also been proposed [25]. However, our approach significantly differs from the above approach by using the multi-pathing mechanism. These mechanisms are likely to be available with the next generation InfiniBand products [16]. The solutions proposed in this study can be easily extended with the congestion control solutions proposed for InfiniBand.

7. CONCLUSIONS AND FUTURE STUDY

In this paper, we have presented a novel performance-based adaptive approach for hot-spot avoidance, which performs batch-based striping and sorting (BSS) during the application execution to eliminate the path(s) with low bandwidth in the network. We have also thoroughly reviewed our previous proposals for providing hot-spot avoidance, particularly hot-spot avoidance with MVAPICH (HSAM). The inherent limitations of the HSAM design prohibits the utilization of all the physically disjoint paths at runtime. As a result, better paths may never be even explored. We have also implemented the HSAM [7] scheme in our MPI stack to compare the performance of different BSS configurations, the best configuration of the HSAM [7] scheme and the original case (no multi-pathing at all). Using MPI_Alltoall, we have achieved an improvement of 27% with different BSS configurations compared with the best configuration of HSAM on 48 processes. The default mapping of tasks in the cluster shows similar benefits. Using MPI_Allgather and MPI_Allreduce with the default mapping of tasks, an improvement of 32% in latency is observed for 64 processes. Using NAS Parallel Benchmark, Fourier transform, with different problem sizes, the execution time is improved by 5–7% with different BSS configurations compared with the best HSAM configuration.

In the future, we plan to continue the research in this direction. We plan to evaluate candidate petascale applications on larger-scale InfiniBand cluster, especially with multi-core architectures. We plan to design routing engines for the subnet manager, which can optimize at least a subset of communication pattern type for a single scheduling of tasks. This will also help us identify the optimal performance for one instance of execution. ConnectX [16], the next-generation InfiniBand interconnect, is also a promising technology for congestion control. We plan to design hot-spot avoidance schemes based on this interconnect.

REFERENCES

1. TOP 500 Supercomputer Sites. http://www.top500.org [2008].
2. InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2, October 2004.
3. Message Passing Interface Forum. *MPI*: *A Message-Passing Interface Standard*, March 1994.
4. Message Passing Interface Forum. *MPI-2*: *Extensions to the Message-Passing Interface*, July 1997.
5. Leiserson CE. Fat-Trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers* 1985; **34**(10):892–901.
6. Liu J, Vishnu A, Panda DK. Building multirail infiniBand clusters: MPI-level design and performance evaluation. *SC*, Pittsburgh, PA, U.S.A., 2004; 33.
7. Vishnu A, Koop MJ, Moody A, Mamidala AR, Narravula S, Panda DK. Hot-spot avoidance with multi-pathing over InfiniBand: An MPI perspective. *CCGRID*, Rio de Janeiro, Brazil, 2007; 479–486.
8. Network-Based Computing Laboratory. MVAPICH/MVAPICH2: MPI-1/MPI-2 for InfiniBand and iWARP with OpenFabrics. http://mvapich.cse.ohio-state.edu/ [2008].
9. Bailey DH, Barszcz E, Barton JT, Browning DS, Carter RL, Dagum D, Fatoohi RA, Frederickson PO, Lasinski TA, Schreiber RS. The NAS parallel benchmarks. *The International Journal of Supercomputer Applications* 1991; **3**:63–73. citeseer.ist.psu.edu/bailey95nas.html.

10. Kumar V, Grama A, Gupta A, Karypis G. *Introduction to Parallel Computing*: *Design and Analysis of Algorithms*. Benjamin-Cummings Publishing Co., Inc.: Redwood City, CA, U.S.A., 1994.
11. Petrini F, Feng WC, Hoisie A, Coll S, Frachtenberg E. The quadrics network: High-performance clustering technology. *IEEE Micro* 2002; **22**(1):46–57. http://dx.doi.org/10.1109/40.988689.
12. Sancho JC, Robles A, Duato J. A new methodology to computer deadlock-free routing tables for irregular networks. *Communication*, *Architecture*, *and Applications for Network-Based Parallel Computing* 2000; **1797**:45–60.
13. Sancho JC, Robles A, Duato J. Effective strategy to compute forwarding tables for InfiniBand networks. *ICPP'02*: *Proceedings of the 2001 International Conference on Parallel Processing*, Valencia, Spain, 2001; 48–60.
14. Vishnu A, Santhanaraman G, Huang W, Jin HW, Panda DK. Supporting MPI-2 one sided communication on multi-rail InfiniBand clusters: Design challenges and performance benefits. *HiPC*, Goa, India, 2005; 137–147.
15. OpenMPI. Open Source High Performance Computing. http://www.open-mpi.org/ [2008].
16. Mellanox Technologies. http://www.mellanox.com/ [2008].
17. Intel MPI. Benchmark. http://www.intel.com/cd/software/products/asmo-na/eng/219848.htm [2008].
18. Coll S, Frachtenberg E, Petrini F, Hoisie A, Gurvits L. Using multirail networks in high-performance clusters. *IEEE Cluster Conference*, Newport Beach, CA, U.S.A., 2001. citeseer.ist.psu.edu/coll01using.html.
19. Sivaram R, Govindaraju RK, Hochschild PH, Blackmore R, Chaudhary P. Breaking the connection: RDMA deconstructed. *Hot Interconnects* 2005; **12**:36–42.
20. Vishnu A, Benton B, Panda DK. High performance MPI on IBM 12 × InfiniBand architecture. *Proceedings of International Workshop on High-Level Parallel Programming Models and Supportive Environments*, *Held in Conjunction with IPDPS'07*, Long Beach, CA, U.S.A., March 2007.
21. Quadrics Supercomputers World Ltd. http://www.quadrics.com/ [2008].
22. Petrini F, Frachtenberg E, Hoisie A, Coll S. Performance evaluation of the quadrics interconnection network. *Journal of Cluster Computing* 2003; **6**(2):125–142.
23. Myricom Corporation. http://www.myri.com/ [2008].
24. Flich J, Malumbres MP, López P, Duato J. Improving routing performance in myrinet networks. *IPDPS*, Cancun, Mexico, 2000; 27–32.
25. Yan S, Min G, Awan I. An enhanced congestion control mechanism in InfiniBand networks for high performance computing systems. *AINA* 2006; **1**:845–850.
26. Gusat M, Craddock D, Denzel W, Engbersen T, Ni N, Pfister G, Rooney W, Duato J. Congestion control in InfiniBand networks. *Hot Interconnects* 2005; **12**:158–159.
27. Santos JR, Turner Y, Janakiraman GJ. End-to-end congestion control for InfiniBand. *INFOCOM*, San Francisco, CA, U.S.A., 2003.