Contention-Aware Communication Schedule For High-Speed Communication

Anthony T.C. Tam and Cho-Li Wang

Department of Computer Science and Information Systems

University of Hong Kong

{atctam, clwang} @csis.hku.hk

Abstract

Much of the effort has been devoted to address the software overhead problem in the past decade, which is known as the major hindrance on high-speed communication. However, this paper shows that having a low-latency communication system does not guarantee to achieve high performance, as there are other communication issues that have not been fully-addressed by the use of low-latency communication, such as contention, communication patterns and scheduling of communication events. In this paper, we use the complete exchange operation as a case study to show that with careful design of communication schedules, we can achieve efficient communication as well as prevent congestion on a high-performance commodity network. We have developed a complete exchange algorithm, the Synchronous Shuffle Exchange, which is an optimal algorithm on the non-blocking network. To avoid congestion loss caused by the non-deterministic delays in communication events, a global congestion control scheme is introduced. This scheme uses a global windowing concept to coordinate all participating nodes to monitor and regulate the traffic load, which effectively avoids congestion loss and maintains sufficient throughput to maximize the performance. To improve the effectiveness of the congestion control scheme when working on the hierarchical network, we incorporate information on the network topology to devise a contention-aware permutation. This permutation scheme generates a communication schedule, which is both node and switch contention-free as well as distributing the network loads more evenly across the hierarchy. This relieves the congestion build-up at the uplink ports and improves the synchronism of the traffic information exchange between cluster nodes. Performance results of our implementation on a 32-node cluster with various network configurations are examined and reported in this paper.

1 Introduction

The performance problem related to the communication software has been an active research issue for the past decade. Currently, lightweight messaging systems [7, 18, 19, 25, 32, 35, 36] offer the best communication performance, as they create a fast communication path that bypasses the traditional in-kernel messaging protocol stack (e.g. TCP/IP), which is a serious obstacle in exploiting the high performance of modern network [1, 15, 16]. Although most of these lightweight messaging systems are successful in delivering the raw network performance to higher-level applications, there remain a number of issues that are not well addressed by these lightweight messaging systems, such as:

- Only focus on point-to-point performance - they lack of supports on guiding the development of efficient high-level communication primitives atop of the lightweight messaging layer.
- System behavior under heavy load - with the availability of low-latency communication mechanisms, applications can now generate higher load to the network, that could result in congestion build up in some part of the network, and of the worst scenario, this would induce congestion loss problem. As the TCP/IP protocol suit is not included in the lightweight messaging systems, the system programmers have to consider about supporting the congestion control mechanism on top of these communication systems.
- Contention issue - the degree of contention has a direct implication on the sustainable performance for a particular architecture-application pair. Different combination of hardware and software, together with different communication patterns and schedules may stimulate different congestion behavior. These make modeling of congestion behavior on a global communication event a challenging task.

In this paper, we make use of the most demanding communication pattern on all message-passing machines, the Complete Exchange operation, as an example to illustrate how we could address the communication issues on a high-speed commodity network, which operates under a lightweight messaging system. This paper introduces three techniques to exploit the advantages of low-latency communication, which optimize the communication efficiency as well as guard against the congestion loss problem.

The first technique optimizes the pipelining efficiency by careful scheduling of all message exchanges according to a node contention-free schedule at the packet level. This effectively balances the usage of available network resources, and completely eliminates unnecessary startup overhead and synchronization overhead; and therefore, achieves good communication performance. We make use of a realistic communication model to show that this communication schedule is a bandwidth-optimal algorithm on any non-blocking network.

Even with a node contention-free communication schedule, other factors could introduce non-deterministic delays to well-scheduled communication events, such as variations in process scheduling and competition with

high-priority system activities. Under such circumstances, the buffers of the switches play a crucial role. This is because, as the network links are fully utilizing to achieve high-performance, any deviation from the normal communication schedules will break the logical harmony and induce contention, and therefore, is handled by the switch's buffers. However, network buffers are scarce resources.

The second technique avoids performance loss caused by the overrun of network buffers by introducing a congestion control scheme to prevent oversubscription to the network. The essential feature of this control scheme is the proactive approach in handling congestion, which monitors and regulates the traffic loads and ensures a fair sharing of network resources that avoids buffer overflow. We make use of available resource information, such as the network buffer capacities of the switched network, the communication pattern and communication volume, to derive this resource-aware congestion control scheme.

The third technique improves the effectiveness of the congestion control scheme when operating on the Hierarchical network. The hierarchical network makes use of higher speed technology as the backbone network to support full-connectivity between many smaller subnetworks, while these subnetworks can be composed of a single switched network or another hierarchical network. We incorporate information on the network topology to devise a contention-aware permutation, which generates a communication schedule that is both node and switch contention-free as well as distributing the network loads more evenly across the hierarchy. This relieves the congestion build-up at the uplink ports and improves the synchronism of the traffics information exchange between cluster nodes.

The effects of these contention-aware techniques are evaluated on a cluster with 32 nodes, which are tested by interconnecting these nodes with different network configurations. The experimental results show that our complete exchange algorithm can utilize more than 90% of the underlying network bandwidth for most of the testing configurations.

The rest of this paper is organized as follows. Section 2 provides an overview of the network technology and the communication pattern that we are focusing on. We also describe a system model of the interconnection network, in which our analyses are based on. Section 3 presents the three contention-aware techniques, which combine to provide an efficiency complete exchange algorithm. In this section, we also explain the difficulties in achieving high-performance communication under the real environment. The experimental evaluation of this contention-aware complete exchange schedule on various network configurations are presented in Section 4. Section 5 discuss others important related work, and finally, concluding remarks are made in Section 6.

2 Background

2.1 Complete Exchange Operation

Complete exchange, also known as all-to-all personalized communication, is a collective operation takes place with a set of processes, and each process has a distinct set of data to transmit to every other process in the set. It is known to be the most stringent communication requirement imposed on the interconnection network. Such a communication pattern occurs in numerous numerical and scientific applications.

Due to its importance, complete exchange operation has been extensively studied in the past. Most of the studies are focused on designing communication schedules to avoid contention delay induced by the topological constraints of the underlying networks, such as hypercubes [2], meshes [29], tori [33], fat-trees [24], multistage interconnection networks [37]and multi-dimension networks [10]. These algorithms exploit the full performance of the underlying networks by carefully scheduling communications to avoid both node and link contention. Thus, these communication schedules are almost shaped to the target network constraints.

General speaking, algorithms for complete exchange can be classified into two categories, the direct or indirect approaches. For the direct algorithm, each process directly sends those data blocks to each of the destination processes using separate communication steps. A clear advantage is that the messages are delivered right to the destinations without going through any intermediate nodes; hence, each message appears only once in the network. This favors networks with higher connectivity such as multistage interconnection networks and the crossbar networks, since the major issue is to schedule the transmissions such that no link contention takes place. For the indirect algorithm, data blocks for a set of destination processes are combined to a larger block and are sent to a representative process, this is then forward to the correct destination processes. The indirect approach reduces the number of communication steps to reduce the startup cost; however, it introduces more traffic in the network and extra software overheads in performing data permutation. Thus, indirect approach favors small size messages exchange, while direct approach favors long messages exchange [3, 4, 6].

To avoid both link and node contention, some complete exchange algorithms split the communication schedule into multiple phases. Each phase corresponds to a contention-free routing of messages between nodes. This approach restrains the parallelism between different phases, as a process would not enter next phase unless it has finished the message exchange of the current phase. Besides, not all processes are active in each phase [17, 33]; therefore, inactive processes have to be kept idle for the whole phase. Furthermore, to achieve this contention-free synchronism, the schedule would induce substantial synchronization overhead. First, processes ahead of schedule cannot continue, this means some of the links carry no data. Second, it is hard to enforce this synchronism on a distributed system. In particular, synchronization achieved by software solutions (e.g., barrier) could contend with normal data transfer and this may become a waste of bandwidth. Bokhari et al. [5] have pointed out that by using

a relaxed synchronization scheme that possibly increases the network contention, the overall performance of the complete exchange communication could be improved.

2.2 Ethernet-Based Cluster Interconnect

Ethernet-based network is the most widely used local area networking technology. Although standard Ethernet has limited bandwidth in supporting cluster computing, its enhanced versions, such as Fast Ethernet (FE), Gigabit Ethernet (GE) and 10 Gigabit Ethernet provide sufficient bandwidth with a steady upgrade path in building large-scale commodity clusters. Therefore, many self-made large-scale clusters are using Fast Ethernet and/or its successor as the base of their interconnections.

Currently, there are two approaches in building a large-scale cluster using the Ethernet-based interconnections. First, using a single high-performance, high port density chassis switch to connect all machines. Second, using a hierarchical network, in which cluster nodes are connected to Fast Ethernet switches and using the Gigabit Ethernet as a backbone network to interconnect all Fast Ethernet switches. Given that the backbone capacity of the interconnection network is greater than the bandwidth demands of the whole cluster, both approaches could support a fully connected network with similar performance. In reality, both approaches could suffer on architectural constraints that limit their actual performance.

Non-Blocking Switch The use of switches in LANs becomes an effective way to increase the network bandwidth. Besides of the improvement in network performance, these switches provide greater flexibility and interconnect scalability in network design. For example, some commercial network products even support non-blocking switching capability up to hundreds or even thousand ports [21]. A switch is said to be non-blocking if the switching fabric is capable of handling the theoretical total of all ports, such that any routing request to any free output port can be established successfully without interfering other traffics.

Hierarchical Network The hierarchical approach makes use of compatible network technologies, which is a cost-effective method in incremental scaling of the cluster. With Ethernet-based devices, connections between different technologies are commonly bridged by one or more uplink ports, which are add-ons to those "lower" end devices. Due to the requirement of having higher channel bandwidth for the uplinks, this limits the switching mechanism adopted on this type of interconnection. For example, cut-through switching is not suitable for the uplink connections, and makes the store-and-forward switching be the only feasible solution. However, the change of channel bandwidth between two technologies may induce hot spot since store-and-forward switching causes cumulating of upstream and downstream packets over those uplink ports.

Performance limitations Theoretically, connecting all cluster nodes via a single non-blocking switch provides the best performance. However, there are other internal factors that hinder the switch performance. In particular, the buffering mechanism used within the switches is one of the crucial factors. There are many variations of switch's buffering architecture, most commodity switches fall into one or a combination of these three basic types: input-buffered, output-buffered and shared-buffered.

For the input-buffered architecture, incoming packets are queued in buffers, one per input port. This is the simplest design as the internal speed of the buffers only operates at the same speed as the input/output links. However, it is known to have the Head-Of-Line (HOL) blocking problem. Packets block at the head of the queue also block the packets behind them, even if some of these packets are destined for idle output ports. By using queuing analysis, HOL blocking is shown to reduce throughput to 58% even under uniform traffic. While for the other two architectures, output-buffered and shared-buffered, their buffering mechanisms avoid the HOL problem, and thus have a higher congestion tolerance and provide better performance than input-buffered switches. However, due to technological constraints, the performance of the buffers must be fast enough to sustain simultaneous access, and this requires more complex and stringent design.

Apparently, even under a node contention-free schedule, sharing of uplinks is needed on a hierarchical network. For instance, all cross-switch traffics are going via the uplinks to the Gigabit Ethernet switch, packets have to contend for the shared links although they may be from distinct sources and to distinct destinations. In theory, under a node contention-free schedule, any transient congestion over the uplinks could be handled by the buffers in the switches as well as the higher throughput of the uplink connections. However, the loosely-coupled nature of the clusters does not guarantee to adhere to a tightly synchronized schedule. Any random delay on scheduling communication events of the complete exchange operation may result in considerably contention. As congestion is handled by the buffers in the switches, therefore, we see that the buffering mechanism plays a critical role in the overall performance of the hierarchical network.

2.3 System Model

We analyze the performance of the complete exchange algorithms based on a communication model discussed in [30]. This communication model involves several parameters: send overhead (O_s), network latency (L), network gaps (g_s, g_r), receive overheads (O_r, U_r) and network buffer capacity (B_L). The parameter O_s stands for the software overhead associated with the send process for sending a *b*-byte data packet. The overall cost reflects the performance of the host node, e.g. CPU and system bus speeds, and the involved communication protocol. The parameter L is the hardware latency of moving a *b*-byte packet from the physical memory of the source node to the physical memory of the destination node. It encapsulates network-dependent features, such as network topology, network speed, and diameter between communicating entities. The value of L is subjected to the traffic load in

a real network. With the hierarchical network, we have two different latency values for communication between cluster nodes within a switch and across switches.

The parameter B_L corresponds to the available buffers in a switch, which is a measure of the network tolerance of the switch in handling contention. Parameters g_s and g_r encapsulate the minimum time between consecutive injection or reception of *b*-byte packets to or from the network by the communication hardware. It models the data transfer capabilities of the host machine and the network interface controller, such as DMA transfer and the network technology in use. For a homogeneous cluster, we generally assume that $g_s \approx g_r$ and simplify the expression by $g = \max(g_s, g_r)$. Lastly, parameters O_r and U_r stand for the software overheads induced by the asynchronous reception of a *b*-byte packet. With O_r captures the costs of all kernel events including interrupt overhead and memory copy, and U_r captures the cost of user-space events such as data processing and high-level protocol handling.

With current CPU performance and the adoption of low-latency communication support, software overheads induced in communication have been significantly reduced. To take advantage of the full-duplex communication, we assume that the cluster communication system satisfies this condition, $(O_s + O_r + U_r) < g < L$. As a result, under no conflict, the one-way point-to-point communication cost (T_{p2p}) in transferring an M-byte long message between two processes at any two machines of the cluster is modeled as :

$$T_{p2p}(M) = O_s + (k-1)g + L + O_r + U_r \tag{1}$$

where $k = \frac{M}{b}$, which corresponds to the fragmentation of an M-byte message to k data packets of size *b* bytes. For optimal performance, *b* usually stands for the maximum transfer unit (*mtu*) of the underlying communication scheme.

To abstractly describe a two-level switch hierarchy, we use a tree topology as depicted in Figure 1. All cluster nodes are the leaf nodes, and are grouped into disjoint sets with d_1 members in each set. Members of the same set are connected to a parent which is a switch node located at Level 1, and all communications generated by the set - both within set and across set, have to go through this switch node. Communications between sets are established through the root switch node, which fully connects all Level 1 switch nodes. To support high performance communication, we assume that the switch-to-switch link bandwidth c_2 and the node-to-switch link bandwidth c_1 satisfy this constraint, $d_1 c_1 \leq c_2$, which ensures that the throughput capacity of the uplink is able to handle all upstream/downstream traffics generated by the whole set at any particular instant. We also assume that the backbone bandwidth of those Level 1 switches are greater than or equal to $d_1 c_1 + c_2$, and the backbone bandwidth of the root switch is greater than or equal to $d_2 c_2$. With these assumptions, the aggregated bandwidth available to a cluster with *p* nodes (where $p = d_1 d_2$) is bounded by $d_1 d_2 c_1$.

3 Contention-Aware Complete Exchange Algorithm

In this section, we show how to apply the three contention-aware techniques

- Efficient pipelining - to generate a bandwidth-optimal complete exchange algorithm, the Synchronous Shuffle Exchange, which operates efficiently on any non-blocking network.
- Global windowing - a proactive congestion control scheme that avoids buffer overflow problem.
- Contention-aware permutation - which helps alleviating the congestion loss problem on the hierarchical network.

The combined effects of these techniques is to optimize the communication efficiency and guard against the congestion loss problem.

3.1 Synchronous Shuffle Exchange Algorithm

The spirit of this algorithm is the node contention-free schedule operated at the packet level without explicit synchronization operation. By effectively utilizing the send and receive channels, this scheme multiplexes all the messages seamlessly to a single pipeline flow by scheduling consecutive packets to different destination nodes according to a node contention-free permutation (φ). Such that at a particular instant i^j, each process is sending the j^{th} packet to process $\varphi(myid, i)$ directly. There are three numerical functions that can be used online to compute the node contention-free permutation. They are the shift pattern, bitwise XOR pattern and the edgecolor pattern [31]. As each process can uniquely match to different process at each packet transmission step, it guarantees no two packets are directed to the same destination at the same instant, thus achieving no node contention. Algorithm 1 shows the corresponding communication schedule used by this algorithm, and Figure 2 presents an example packet transmission sequence that generated by using the edgecolor permutation on a 6-node cluster.

We now derived the lower bound cost for the synchronous shuffle exchange algorithm on our system model - the complete-connected cluster. Assuming that each node is capable to send and receive a message in one time unit, such that $(O_s + O_r + U_r) < g < L$. With this capability, a process can actively send and receive at the same time, thus can fully utilize the bidirectional channels. If we assume that each process sends and receives *k* data packets to and from *p-1* nodes, then the minimum amount of packets being sent and received in the complete exchange operation by each process is $2k(p-1)$ packets or $2kb(p-1)$ bytes if each data packet is of size *b* bytes.

As the minimal time in sending or receiving a packet of size *b* bytes is bounded by the send gap (g_s) and receive gap (g_r), and each machine can inject or receive no more than one packet within this gap, we deduce that the minimal time required for the synchronous shuffle exchange algorithm under such a cluster communication abstraction is

$$
\begin{aligned}
T_{sync} &= O_s + max((k(p-1)-1)g_s, (k(p-1)-1)g_r) + L + O_r + U_r \\
&= (k(p-1)-1)\, max(g_s,\, g_r) + O_s + L + O_r + U_r \\
&= k(p-1)g + T_w \qquad (2) \\
&\qquad \text{where } T_w = O_s + L - g + O_r + U_r
\end{aligned}
$$

From this formulae, we see that the necessary conditions to achieve the above time bound on the *k*-item complete exchange are:

1. Each data packet must be sent directly to the target node without detour.
2. Each cluster node is actively sending and receiving the data packets without network stalling during the whole course of operation.

Condition 1 ensures that all messages appear once in the network, and therefore, minimizes the total transmission delay and messaging overhead. While with Condition 2, we ensure that no bubble exists in the network pipelines. As our performance goal is to minimize the communication time, existence of bubbles in the network pipeline means that we have to take longer time to complete the transmission. Thus, any schedule that ensures no bubble appears in the network pipelines achieves better performance.

3.2 Global Window Congestion Control

If every operation is executed on schedule and the network resources are scalable, then the synchronous shuffle exchange could finish with minimal time. However, in reality, logical synchrony is not enforced due to the distributed nature of the cluster system. Random delays between communication events, such as scheduling delays, could break this harmony and result in “transient hot-spot” in the switch. Observed that the more packets are targeting to the same output link, which are arriving from different sources at different time period, the higher chance of having conflicts even under a regular and uniform pattern. When two or more packets contend for the same output link, buffering of conflicting packets would result in routing delay. As the buffering technique within the switch has a significant impact on the network performance, we reckon that the effectiveness of the synchronous shuffle exchange algorithm would be affected by the head-of-line blocking problem on the input-buffered switch.

Generally, having logical synchrony on all cluster nodes is an idealistic assumption for the case of commodity clusters, which have no hardware synchronization support. To impose this synchrony, explicit synchronization operations can be used, e.g. Gang Scheduling is one of the approaches to solve this problem. However, this

brings on extra synchronization overhead to the total communication time, and also stalls the communication pipelines as no data communications are taking place during synchronization. Since performance loss is caused by oversubscription to the network, which induces packet loss at the bottleneck region, the best solution to avoid congestion loss is to prevent oversubscription to the network. That can be done by applying some form of traffic control on each node to minimize and manage the contention problem.

The conjecture behind the contention problem induced by the synchronous shuffle exchange algorithm on a non-blocking network is the non-deterministic delays on communication events. With the hierarchical network, two more sources of delay could contribute to this non-determinism.

- Queuing delay at the uplinks: with the hierarchical network, inter-switch traffics have to go through shared uplinks and conflicting packets are buffered; therefore, increases the network delay.
- The difference of network latencies between nodes: even with the use of faster technology for upper level interconnections, additional transmission delays on delivering the data across the hierarchy would induce the contention problem.

To achieve optimal performance on the hierarchical network, sharing of links is necessary. Thus, having link contention is a fact that we must confront with. Although mild contention increases network delay, it does not severely degrade the performance, unless the congestion persists for a long period of time, which results in buffer overflow. Therefore, it would be useful to have a congestion control scheme to prevent oversubscribing the network.

In this study, we adopt a proactive approach in the congestion control. This congestion control scheme is different from traditional approaches. Conventional mechanisms for controlling congestion are based on end-to-end windowing schemes [28]; however, they are not suitable for collective operations in high-speed networks. This is because they are usually *reactive* schemes. They probe for congestion signals, such as packet loss and timeout signals, and respond by recovering the loss and regulating the traffic load to avoid further loss. However, we have already lost some packets, and this has affected the performance. Besides, the feedback information from the network is usually outdated due to the propagation and transmission delays. Hence, any reactive action taken may be too late to avoid further loss. Furthermore, end-to-end windowing only provides traffic information on individual connection. It lacks in a global picture of the network, such as the number of traffic sources and the communication pattern in used.

With cluster computing, the traffic pattern is predictable in the case of collective operations on an enclosed network. Therefore, we can utilize those available information, such as the network buffer capacities (B_L) of the switched network, the communication pattern and communication volume, to derive some resource-aware congestion control scheme. With our global congestion control scheme, each source is assigned with a predefined resource limit, and our scheme forces them to regulate their traffic loads below this limit. By having a fair share of

resources, this ensures that no source will exceed its allowed traffic capacity and avoids congestion loss.

We have observed that during the execution flow, at i^{th} communication step, a process is sending a data packet to another process according to the node contention-free permutation scheme φ. If every operation is on schedule, the number of outstanding data packets (η) in transit from a process to other process is bounded by $\left\lceil \frac{L}{g_s} \right\rceil$. Under mild congestion, the process experiences slight increase of η. If congestion persists, this eventually induces packet loss, and η will increase considerably. The above observation implies that to avoid overflowing the network buffers, we need to regulate the number of outstanding packets (η).

The principle behind our congestion control scheme is quite simple. When applying this scheme on our complete exchange algorithm, all senders are assigned with a global window (W_g) at the beginning of the communication event. This W_g factor acts as a controller to limit the amount of traffic that a particular sender can inject into the network. If a sender finds that sending out a data packet may overload the network, when $\eta = W_g$, it just halts current transmission and waits until it is safe to transmit, i.e. $\eta < W_g$. By picking the correct value for W_g, this scheme guarantees that during any interval, the total number of packets entering the network does not exceed the sum of a pre-specified limit, which is the network buffer capacity at the bottleneck region. To compute W_g, we need to identify the bottleneck region and measure the buffer capacity (B_L) associated to the bottleneck, then we derive W_g from B_L on the principle of fair sharing.

Takes the hierarchical network as an example. Based on the communication pattern and schedule, we estimate the average number of packets (ν) generates at each communication step which are forwarded to the bottleneck region. Without lost of generality, let's take the FE/GE hierarchical network as an example. Assume that the uplink ports are the critical bottlenecks and they are of input-buffered architecture. Under the synchronous shuffle schedule, in *p*-1 communication steps, a process generates *p*-1 data packets which are destined to *p*-1 distinct nodes. However, only $d_1 - 1$ packets are switched locally, and the rest, $p - d_1$ packets, are forwarded by the Fast Ethernet switch to its uplink port. Therefore, there are $(p - d_1)d_1$ packets being forwarded upstream by each FE switch in *p*-1 communication steps. Based on the node contention-free permutation, the same amount of data packets are switched from the Gigabit backbone back to each FE switch. Thus, the average number of data packets directed to each uplink port per communication step is

$$\nu = \frac{(p - d_1)d_1}{p - 1} \tag{3}$$

From this, we derive the value of W_g, which is

$$W_g = \left\lfloor \frac{B_L}{\nu} \right\rfloor = \left\lfloor \frac{B_L(p-1)}{(p - d_1)d_1} \right\rfloor \tag{4}$$

3.3 Contention-Aware Permutation

However, knowing the value of W_g is a necessary but not sufficient condition to avoid congestion loss on the hierarchical network. This is because W_g is derived from taking the average traffic load, and unlike traditional end-to-end scheme, global windowing needs to monitor and regulate all traffic flows of a process, not just one connection. If the traffic distribution is not uniformly spread across the network, the global windowing scheme could not fulfill its function correctly. This is being shown in Figure 3. In this example, we assume that the bottleneck region of the 4X4 two-level hierarchical network is at the uplink ports with $B_L = 30$. However, under the XOR permutation scheme, we could experience the contention loss problem even though global windowing is adopted.

In this example, the size of W_g is $\lfloor \frac{30}{3.2} \rfloor = 9$. Assume that at communication step *i*, four packets originate from switch 2 and head for switch 3 are blocked by some cause, e.g. HOL, so as those packets that follow in step *i*+1, *i*+2, and *i*+3 from the same switch. However, no process is aware of the congestion problem unless their global windows become saturated. This may only be happened until step *i*+8 when processes in switch 3 detect the congestion problem. By that time, processes in switch 1 have already sent out all their packets to processes in switch 3, which further increases the queue length at switch 3. Moreover, processes in switch 0 are not aware of the problem. This is because global windowing collects traffic information on the base of past events, but none of these past events could indicate the congestion problem in switch 3. As a result, processes in switch 0 continue to send all their packets to processes in switch 3, which finally overflow the buffer.

An obvious reason for this failure is that the feedback information on traffic condition is not regularly gathered from all part of the network. Thus, information on part of the network is outdated. Although the overflow situation could be detected and resolved by both global windowing and individual end-to-end flow control scheme, performance has been suffered as packets are lost inevitably. If we can arrange all communication events in a way that each process is communicating with different processes reside in a node linked to different switches at each communication step. Then, the traffic loads would become more evenly distributed as well as having more regular information feedback between different processes in different part of the network.

An approach in generating this kind of dispersive pattern is by adopting a contention-aware permutation, which includes knowledge on the network constraints to generate the communication schedule. Observed that the original permutation is obtained by some simple functions (φ) which operate on inputs such as current communication step and node id. A simple method to incorporate the network structure into the original permutation is by redefining a mapping between the logical node id to its physical id. One example of such permutation scheme (ϕ) can be as follows:

$$\text{logical id} = \left\lfloor \frac{\text{physical id}}{d_1} \right\rfloor + (\text{physical id} \,\%\, d_1) * d_2 \tag{5}$$

Carry on with the previous example. If we apply the XOR permutation on the re-mapped logical id, we get the communication schedule as shown in Figure 4, which is a more evenly distributed pattern with respect to both switches and cluster nodes. We observe that with this new communication pattern, a process is communicating with different processes located in different part of the network in consecutive communication steps. This greatly relieves the contention at the uplink ports and improves the effectiveness of our congestion control scheme.

Based on the global windowing and the contention-aware permutation scheme, we have transformed the synchronous shuffle exchange algorithm (Algorithm 1) to work efficiently on the two-level hierarchical network, and the modified algorithm is given in Algorithm 2.

3.4 Related Algorithm - Pairwise Exchange

As mentioned in Section 2.1, quite a bit of work has been done in implementing complete exchange operation on specific network topologies. With the use of switched network that supports full connectivity, the internal communication flows within the network become less critical, but avoiding node contention is crucial to the performance. A commonly used scheme in this type of network is to schedule the sequence of communication events so that *p-1* rounds of disjoint pairwise exchanges are performed [9]. The standard approach is using the XOR bitwise operation to pair up processors in each round. However, the major drawback of the XOR bitwise operation is the requirement of $p = 2^x$ in order to symmetrically pairing up all the nodes. For the case with $p \neq 2^X$, the number of rounds becomes $2^{\lceil log_2 p \rceil} - 1$, and during each round, not all the nodes find a matching partner. A general solution to the disjoint pairing problem is by using the edgecolor pairing of the complete graph [31], which does not have the power of two constraint. The resulting algorithm becomes the Generalized Pairwise Exchange algorithm (Algorithm 3). Under this mapping scheme, the performance is only slightly deteriorated with *p* communication rounds for all odd cases, instead of having *p-1* communication rounds for all even cases.

However, as depicted in Algorithm 3, the non-stalling condition (Condition 2) is not enforced under this scheme. Although there is no explicit synchronization appeared between consecutive rounds and both send and receive operations are of non-blocking semantics, the *p*-1 rounds have an implicit synchronization cost that introduces bubbles to the network pipelines. The predicted communication cost for this complete exchange operation is

$$\begin{aligned} T_{gpw} &= (p-1)(O_s + kg + L - g + O_r + U_r) \\ &= kg(p-1) + (p-1)T_w \end{aligned} \tag{6}$$

When comparing the two cost formulae, Eq. 2 & 6, we notice that the generalized pairwise exchange algorithm

has a higher messaging overhead, which is proportional to the number of cluster nodes as denoted by $(p-1)T_w$.

4 Performance Evaluation

Our experimental platform is a cluster consists of 32 standard PCs running Linux 2.2.14. Each cluster node equips with a 733MHz Pentium III processor with 256KB L2 cache, 128MB of main memory, an integrated 3Com 905C FE controller. We use the Directed Point (DP) communication system [19] to drive the network and conduct all our experiments. We have implemented a variant of the Go-Back-N protocol to support limited reliability on DP. In this study, we use four Fast Ethernet switches and one Gigabit Ethernet switch to set up various configurations to evaluate our algorithm.

The GE backbone switch is a chassis switch from Alcatel. It is the model PowerRail 2200 (PR2200) with backplane capacity reaches 22 Gigabit per second (Gbps). This switch is equipped with 8 GE ports on 2 modules, but we only use at most 4 ports in our experiments. Four FE switches are from IBM, which are of the model 8275-326. It is a 24-port input-buffered switch with backplane capacity reaches 5 Gbps. A one-port GE uplink module is installed on each FE switch for connecting to the Gigabit backbone switch. Table 1 summaries all the buffer parameters (B_L) of the above switches, which are used in our algorithm to compute the global windows (W_g) on different network configurations.

To analyze and evaluate the performance of our congestion control mechanism, we have set up five different configurations on this cluster - 16x1, 8x2, 8x3, 6x4 and 8x4, with each configuration corresponds to a different degree of contention on the uplink ports (except configuration 16x1). The configuration *A*x*B* corresponds to connect *A* cluster nodes to each FE switch, and there are total *B* FE switches interconnected by the GE switch. This makes up a cluster size of $A * B$ nodes.

4.1 16-Node Single Switch - 16x1

Our objective of designing efficient communication schemes atop of lightweight messaging system is to support high-level programming model such as MPI. Therefore, it would be informative to have a direct comparison of the DP implementation of the synchronous shuffle exchange algorithm against the MPICH [20] implementation of the complete exchange operations on the same platform. Figure 5 presents the measured results of four complete exchange implementations on a 16-node cluster interconnected by a single input-buffered switch (IBM 8275-326). They are the synchronous shuffle exchange with global windowing (sync+GW), the generalized pairwise exchange (pair), the original MPICH implementation (MPICH) and the generalized pairwise exchange MPI implementation (pair-MPI), which is implemented with the *MPI_sendrecv()* communication primitive. The experiment is conducted with each node sending a long message to every node in the cluster, which ranges from 1 KB to 1200 KB of

data to each node. Both the measured performance and the per-node achieved bandwidth of each implementation are shown in the graphs.

With the global windowing scheme to guide against the congestion loss problem, we show that the synchronous shuffle algorithm can operate efficiently and effectively for the whole message ranges. When compared to the predicted performance (Eq. 2), the contention-aware synchronous shuffle exchange algorithm has its efficiency ranged from 87% to 97% of the theoretical bandwidth. When compared with the generalized pairwise exchange, the results show that the synchronous shuffle algorithm can effectively mask away synchronization overhead and achieves better performance. This shows that the add-on congestion control scheme does not affect the efficiency of the synchronous shuffle exchange algorithm. Indeed, it effectively guards against congestion loss.

When compared with both MPI implementations, we clearly see that the original *MPI_Alltoall()* function performs extremely inefficient. This is because their implementation is based on simple non-blocking *MPI_Isend()* and *MPI_Irecv()* functions, which are issued in an uncoordinated manner. Therefore, it would subject to both node and switch contention as well as the high overhead problem that inherits from the TCP protocol stack. To avoid the node and switch contention, the "pair-MPI" carefully coordinates the communications, and achieves considerable improvement. However, our DP implementation of synchronous shuffle exchange even outperforms the "pair-MPI" implementation significantly. This shows that the traditional protocol stack has severe limitation on achieving high-speed communication. In other words, it is inadvisable to drive the high-performance communication network with the conventional communication protocols.

4.2 16-Node Hierarchical Configuration - 8x2

We start our experiments on the hierarchical network by first using a 16-node configuration. This configuration uses two Fast Ethernet switches with eight nodes connect to each switch, and they are interconnected via the Gigabit Ethernet switch. With this setup, the theoretical *bisection bandwidth* [12] is 1 Gb/s, which should be sufficient for this configuration. However, our preliminary investigations showed that the uplink circuitry of the IBM 8275-326 switch has some performance limitation. In order to provide the correct judgment on the performance of all implementations, we have performed some baseline measurements to determine the best achievable throughput across these GE uplinks.

By changing the configuration to the 10x2 setting, we measure the achieved *aggregated bandwidth* across the hierarchical network by having multiple concurrent bi-directional data exchanges. Figure 6 shows the results of this baseline study. The peak aggregated bandwidth achieved on this setting is 103 MB/s with 12 concurrent bi-directional flows across the uplink ports. Beyond that, the communication performance starts to deteriorate gradually. With the same software and hardware settings, but replacing the hierarchical network with a single IBM 8275-326 switch, we can achieve a linearly scaled aggregated bandwidth, which is labeled as "local" in

the graph. This demonstrates that the limitation is on the uplink circuitry, not on other components. With this baseline measurement, we have a solid foundation to justify on the expected communication efficiency across the problematic uplink ports, such that we have

$$\text{Best cross switch data exchange time} = \frac{\text{Total cross switch volume}}{103\ MB/s} \tag{7}$$

Take the 8x2 configuration as an example, the total cross-switch volume on the *k*-item complete exchange is $2k*mtu*(16-d_1)*d_1$ bytes. Thus, the best timing in delivering this volume of data across the uplink connection is $\frac{128*k*mtu}{103}$seconds. Assumed that an efficient communication schedule should be able to arrange all local and cross-switch communications be happened concurrently. Therefore, the execution time of the *k*-item complete exchange should be bounded by the best cross-switch data exchange time. Then, the best achieved per-node bandwidth for this *k*-item complete exchange operation on the 8x2 configuration is bounded by $\frac{k*mtu*15}{\frac{128*k*mtu}{103}} = 12.07$ MB$/s$.

After understanding about the performance limitation of the network, we carry on with our analysis. With the 8x2 configuration, the theoretical computed value of W_g is 10; however, when considered together with implementation issue, such as the existence of control packets with the GBN reliable support, the calculated value of W_g is $\left\lfloor 45 \div \frac{(16-8)*8*2}{15} \right\rfloor = 5$. We have measured the performance of the synchronous shuffle algorithm with this global windowing setting, and the results are presented in Figure 7. Five sets of measurements are shown in the graphs. They are the synchronous shuffle with global windowing and contention-aware permutation (sync+GW+CA), generalized pairwise exchange (pair), generalized pairwise exchange with contention-aware permutation (pair+CA), the original MPICH implementation and the generalized pairwise exchange MPI implementation (pair-MPI). The results show that the contention-aware synchronous shuffle exchange performs the best amongst all tested implementations in this configuration. When compared to the expected best achievement, the synchronous shuffle shows its effectiveness in utilizing the network pipelines as well as avoiding the congestion loss, since it reaches 93% of the best achieved performance.

However, we find that the performance of the DP pairwise exchange implementation has degraded considerably under this hierarchical configuration when compared to its performance on the single-switch case (Figure 5b). Initially, the performance of the DP pairwise implementation (labeled as “pair”) increases with the increase in message length until the maximum capacity of the uplink ports has reached. After that, the performance is affected by the congestion loss problem. However, our GBN reliable protocol could only recover from the loss with long message exchanges. This is being shown as the slow increased in the achieved bandwidth after experiencing the congestion loss problem. To investigate on whether the contention-aware permutation scheme would also benefit the pairwise exchange algorithm, we have applied the same contention-aware permutation on the DP pairwise implementation. The measured results (labeled as “pair+CA”) show that this augmentation exhibits a similar

behavior as compared to the pure pairwise exchange.

On the other hand, it is interested to see that the performances of the two MPI implementations do not have significant performance changes, except that the original MPICH implementation has considerable decreased in performance on exchanging small messages. This could be the result of contention over the uplinks as the MPICH implementation does not carefully schedule those communication events. This indicates that under this hierarchical configuration, it demands for a better communication scheme to coordinate the communication events.

4.3 24-Node Hierarchical Configurations - 8X3 and 6X4

To construct a 24-node cluster with our hardware resources, we can arrange the hierarchical network in two different configurations:

1. 8X3 - With this configuration, the computed value of W_g is 4 and the experimental results are shown in Figure 8.

2. 6X4 - With this configuration, the computed value of W_g is 5 and the experimental results are shown in Figure 9.

We have performed the same set of tests on these two configurations as compared to the 8X2 setup. When comparing their expected best achievements on these two configurations, which are of 9.25 MB/s on the 8X3 configuration and of 10.96 MB/s on the 6X4 configuration, we see that the 6X4 configuration has a better throughput performance. Once again, we see that the synchronous shuffle performs the best on both setups; however, it only reaches 89% and 88% of the expected best achievements on the 8X3 and 6X4 configurations respectively. A possible explanation on the performance degradation is due to the use of small global window settings, which may reduce the pipelining efficiency. However, increase the global window size would increase the congestion loss probability, this could create an adverse effect on the overall performance.

As for the two MPI implementations, their measured performances look similar to the performance observed in the 8X2 configuration. In addition, we observe that as the cluster size has increased from 16 nodes to 24 nodes, the contention problem of the original MPICH implementation on small message exchanges is getting worse than the 8X2 configuration. Nevertheless, all these findings support our belief that conventional communication libraries are restrained by the high software overheads.

As for the DP pairwise implementations, their measured results exhibit different performance behaviors on these two configurations. On a configuration that supports less aggregated bandwidth (8X3), we find that we have experienced severe performance loss starting at small size message exchanges. But, with a less restrictive configuration (6X4), the losses start to appear only on medium size message exchanges. After that, the performance slowly increases when exchanging longer messages. On the other hand, we find that the add-on contention-aware

scheme (pair+CA) performs better on the 8x3 configuration when compared to the pure pairwise implementation. A possible explanation to this observation is that the contention-aware permutation is more effective when operates on a more stringent configuration.

4.4 32-Node Hierarchical Configuration - 8x4

Carry on with the same set of experiments with the global windowing parameter (W_g) set to 3, the measured results are shown in Figure 10. Same as other experiments, the synchronous shuffle algorithm shows its clear advantage over other implementations when running on this configuration. In particular, it achieved per-node bandwidth peaks at 7.67 MB/s, which is around 92% of the expected best achievement (8.32 MB/s) on this configuration.

As for the two MPI implementations, their performances seem to have no big difference as compared to other configurations. However, the performance of our DP pairwise exchange implementation suffers considerably when exchanging small to medium size messages, as the results show that the pairwise MPI implementation outperforms the DP pairwise implementations on this message range. This indicates that our GBN reliable protocol is not working as effectively as the TCP protocol except with large message exchanges. Once again, we observe that the add-on contention-aware permutation on the pairwise implementation has slight performance improvement on the current configuration. When compared this finding with the results on other configurations, we believe that the contention-aware scheme works better on a more stringent configuration. This observation shows that contention-aware permutation alleviates the congestion build-up at the uplink ports; however, it still has to work together with the global windowing scheme in order to avoid congestion loss.

5 Related Work

The uses of models in designing algorithms, in particular communication algorithms, are not uncommon in the parallel community [13, 14, 22, 23, 27]. These algorithms are mostly shown to be efficient or optimal with respect to the based models using paper and pen or simulations. For example, Karp et al. [14] had designed an optimal *k*-item broadcast algorithm based on the LogP model [8]. This algorithm is presented as a communication schedule where the root sends each data item only once, but alternates among recipients in order to retain the logarithmic depth of a tree broadcast. The principle behind this communication schedule matches with our synchronous shuffle exchange algorithm, as both schedules try to arrange the communications such that a processor would only at most send out one data item and receive one data item in one communication step. However, as demonstrated in our experimental studies, these types of communication schedules may be too idealistic, which demand to have logical synchronism in order to achieve the optimal bounds; therefore, it is hard to achieve the claimed performance on a real platform unless more control of the communications is enforced.

Similar to our contention-aware permutation scheme, Nupairoj et al. [22] has reported of using architecture-dependent characteristics of a system to ensure the efficiency of their optimal multicast algorithm, which is built on top of an architecture independent parameterized model [23]. This is because in real network, network contention is likely to occur if concurrent message transmissions are not scheduled properly. Their approach makes use of the topology information, e.g. mesh and multistage networks, to order nodes in the multicast tree to avoid network contention. This supports our belief that optimal performance achievable on parallel machines can be first designed by using realistic architecture independent model, and then perform performance tuning of an implementation on the based of some architecture dependent characteristics.

Donaldson et al. [11] also reported that their BSPlib package, a communication library for BSP [34] programming, is using information related to the global state to improve the collective use of the communication system. Due to the synchronous nature of the BSP machines, communications are constrained to take place only in certain stages. This constraint, although limits its flexibility, can be transformed to make enhancement on communications as each processor can infer the global state of the communication in which it is involved. For example, a BSP process knows which other processes are about to communicate, and how much they plan to send, and then can estimate the global network loading; this in turn, can be used to determine the ideal schedule and transmission rate. Their objective is similar to our approach, but we are working on an asynchronous model, which makes use of the buffering information and communication pattern to devise the corresponding information.

Instead of using the buffer resources as a guide to monitor the congestion problem, other approaches have been proposed to explicitly manage the allocation of scarce resources to different purposes. For example, Roy et al. [26] propose the uses of Quality of Service (QoS) mechanisms at the application level to manage contention, so as to improve the performance of MPI applications. They argue that if appropriate mechanisms can be provided for expressing application requirements, for arbitrating between different requirements, for enforcing allocations, and for providing feedback to applications concerning achieved performance, then applications can adapt their behavior according to resource availability. However, their approach requires that the underlying network supports the QoS mechanisms.

6 Conclusion

The performance of communication networks is the limiting factor of most existing clusters. Improvement of the communication performance by well scheduling of communication events could leverage the overall performance of the clusters. Based on the architecture and communication model of cluster, we observe that to achieve optimal result, the network pipes must be fully utilized. Any waiting stage would stall the pipelines and decrease the overall communication efficiency. This is achieved by the synchronous shuffle exchange algorithm which has adopted a

node contention-free schedule at the packet level.

Theoretically, under a contention-free schedule on a complete-connected network, all packets arrived to different input ports are destined to different output ports; therefore, can be routed instantaneously. However, in reality, no global clock is implemented to coordinate all cluster nodes and their events. Thus, their operations are not lock-step synchronized. Any variations in communication schedules will result in drifting from the theoretical harmony. The consequence is packets start to cumulate in the network due to experience of conflicts. Under this situation, the buffering architecture of the switch plays a critical role in the performance issue.

To avoid congestion loss, we propose the use of synchronous shuffle exchange algorithm with congestion control scheme and contention-aware permutation. With the proactive approach in handling congestion, this algorithm makes use of architectural characteristics to avoid congestion build-up in the first place and reduces congestion whenever it happens. We derive a global window scheme from information on the network buffer capacity, which forces each node to limit their traffic loads and ensures a fair sharing of network resources that avoids congestion overflow. We also make use of information on the network topology to derive a contention-aware permutation in generating a communication schedule, which avoids contention at the node and at the switch, as well as creating a more evenly distributed traffic pattern on the network. This improves the synchronism of the traffic information exchange between cluster nodes, and hence, improves the effectiveness of the global windowing scheme in monitoring the network. The proposed algorithm is implemented on a 32-node cluster with different network configurations. And the results have showed that it can efficiently utilize the network as well as effectively control the congestion problem.

We believe that the hierarchical network model is a practical design to construct large-scale clusters. With this system configuration, clusters can be scaled up to hundreds or thousands of nodes, and support enough bandwidth for high-speed communication. Our research can be used in any combination of Ethernet-based switched networks, which we belief, over 60% of the self-made clusters are based on. And the concept is applicable to future technologies, such as 10 Gigabit Ethernet, which simply extends the topology to multi-level hierarchy.

References

[1] Ammon Barak, Ilia Gilderman, and Igor Mctrik. Performance of the Communication Layers of TCP/IP with the Myrinet Gigabit LAN. *Computer Communication*, 22(11), Jul 1999.

[2] D. Bertsekas, C. Ozveren, G. Stamoulis, P. Tseng, and J. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11:263–275, 1991.

[3] S. Bokhari. Multiphase complete exchange: a theoretical analysis. *IEEE Transactions on Computers*, 45(2):220–229, February 1996.

[4] S. Bokhari. Multiphase complete exchange on paragon, sp2, and cs-2. *IEEE Parallel and Distributed Technology*, 4(3):45–59, Fall 1996.

[5] S.H. Bokhari and D.M. Nicol. Balancing contention and synchronization on the Intel Paragon. *IEEE Concurrency*, 5(2):74–83, 1997.

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

[7] G. Chiola and G. Ciaccio. Gamma: a low-cost network of workstations based on active messages. In *Proceedings of the 5th EUROMICRO workshop on Parallel and Distributed Processing PDP'97*, January 1997.

[8] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[9] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[10] V. Dimakopoulos and N. Dimopoulos. A theory for total exchange in multidimensional interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 9(7):639–649, July 1998.

[11] S. Donaldson, J. Hill, and D. Skillicorn. Exploiting Global Structure for Performance on Clusters. In *Proceedings of IPPS/SPDP'99*, pages 176–182, 1999.

[12] Kai Hwang and Zhiwei Xu. *Scalable Parallel Computing*. McGraw-Hill, 1998.

[13] G. Iannello. Efficient Algorithms for the Reduce-Scatter Operation in LogGP. *IEEE Transactions on Parallel and Distributed Systems*, 8(9):970–982, 1997.

[14] R. Karp, A. Sahay, E. Santos, and K. Schauser. Optimal broadcast and summation in the logp model. In *Proceedings of Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 142–153, June 1993.

[15] Jonathan Kay and Joseph Pasquale. The importance of non-data touching processing overheads in TCP/IP. In *Proceedings of ACM SIGCOMM 93*, pages 259–268, 1993.

[16] K. Keeton, T. Anderson, and D. Patterson. Logp quantified: The case for low-overhead local area networks. In *Hot Interconnects III: A Symposium on High Performance Interconnects*, Aug. 1995.

[17] C. Lam, C. Huang, and P. Sadayappan. Optimal Algorithms for All-to-all Personalized Communication on Rings and Two Dimensional Tori. *Journal of Parallel and Distributed Computing*, 43:3–13, 1997.

[18] M. Lauria, S. Pakin, and A. Chien. Efficient layering for high speed communication: Fast messages 2.x. In *Proceedings of the 7th High Performance Distributed Computing Conference (HPDC7)*, July 1998.

[19] C.M. Lee, A.T.C. Tam, and C.L. Wang. Directed point: An efficient communication subsystem for cluster computing. In *International Conference on Parallel and Distributed Computing Systems (IASTED)*, Oct. 1998.

[20] MPICH. MPICH-A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich/.

[21] Extreme Networks. http://www.extremenetworks.com/products/.

[22] Nupairoj, Ni, Park, and Choi. Architecture-Dependent Tuning of the Parameterized Communication Model for Optimal Multicasting. In *IPPS: 11th International Parallel Processing Symposium*, pages 578–582. IEEE Computer Society Press, 1997.

[23] J.-Y.L. Park, H.-A. Choi, N. Nupairoj, and L.M. Ni. Construction of Optimal Multicast Trees Based on the Parameterized Communication Model. In *Proceedings of the 1996 International Conference on Parallel Processing*, pages 180–187, Aug 1996.

[24] R. Ponnusamy, R. Thakur, A. Choudhary, and G. Fox. Scheduling regular and irregualar patterns on the CM-5. In *Proceedings of Supercomputing '92*, pages 394–402, November 1992.

[25] Loic Prylli and Bernard Tourancheau. BIP: A New Protocol Designed for High Performance Networking on Myrinet. In *IPPS/SPDP Workshops*, pages 472–485, 1998.

[26] A. Roy, I. Foster, W. Gropp, N. Karonis, V. Sander, and B. Toonen. MPICH-GQ: Quality-of-Service for Message Passing Programs. In *Proceedings of the IEEE/ACM SC2000 Conference*, 2000.

[27] S. Shibusawa, H. Makino, S. Nimiya, and J. Hatta. Scatter and Gather Operations on an Asynchronous Communication Model. In *ACM Symposium on Applied Computing*, Mar 2000.

[28] M. Sidi, W. Z. Liu, I. Cidon, and I. Gopal. Congestion control through input rate regulation. *IEEE Transactions on Communications*, 41(3):471–477, 1993.

[29] Y.J. Suh and S. Yalamanchili. All-to-all communication with minimum start-up costs in 2D/3D tori and meshes. *IEEE Transactions on Parallel and Distributed Systems*, 9(5):442–458, 1998.

[30] Anthony T.C. Tam and Cho-Li Wang. Realistic Communication Model for Parallel Computing on Cluster. In *Proceedings of the 1st IEEE International Workshop on Cluster Computing (IWCC'99)*, December 1999.

[31] Anthony T.C. Tam and Cho-Li Wang. Efficient Scheduling of Complete Exchange on Clusters. In *the ISCA 13th International Conference On Parallel And Distributed Computing Systems (PDCS-2000)*, August 2000.

[32] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: A operating system coordinated high performance communication library. In *High-Performance Computing and Networking '97*, 1997.

[33] Yu-Chee Tseng and Sandeep K. S. Gupta. All-to-all personalized communication in a wormhole-routed torus. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):498–505, 1996.

[34] L. Valliant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, Aug. 1990.

[35] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the Nineteenth International Symposium on Computer Architecture*. ACM Press, 1992.

[36] M. Welsh, A. Basu, and T. Von Eicken. Low-latency communication over Fast Ethernet. In *Lecture Notes in Computer Science*, volume 1123, 1996.

[37] Y. Yang and J. Wang. Optimal all-to-all personalized exchange in self-routable multistage networks. *IEEE Transactions on Parallel and Distributed Systems*, 11(3):261–274, 2000.

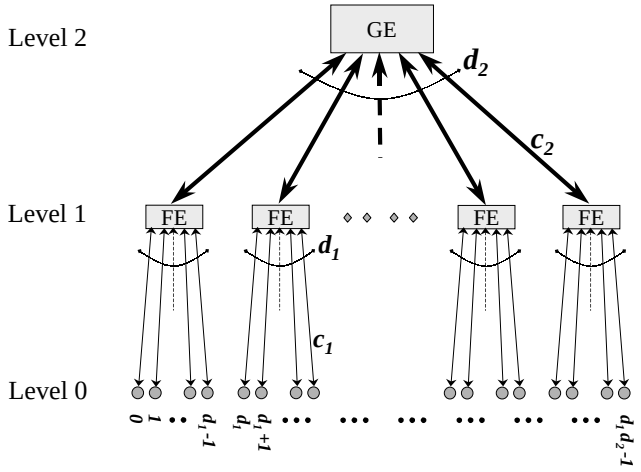


Figure 1: Interconnection topology of the two-level switch hierarchy

Algorithm 1 Synchronous Shuffle Exchange

```
for (s = 1 to k) & (r = 1 to k) in parallel do
    for (i_s = 1 to p-1) & (i_r = 1 to p-1) do
        to = φ_s(myid, i_s)
        from = φ_r(myid, i_r)
        status = send_item_to( to_s, to )
        if (status = SUCCESS) then
            inc i_s
        fi
        status = recv_item_from( from_r, from )
        if (status = SUCCESS) then
            inc i_r
        fi
    endfor
endfor
```

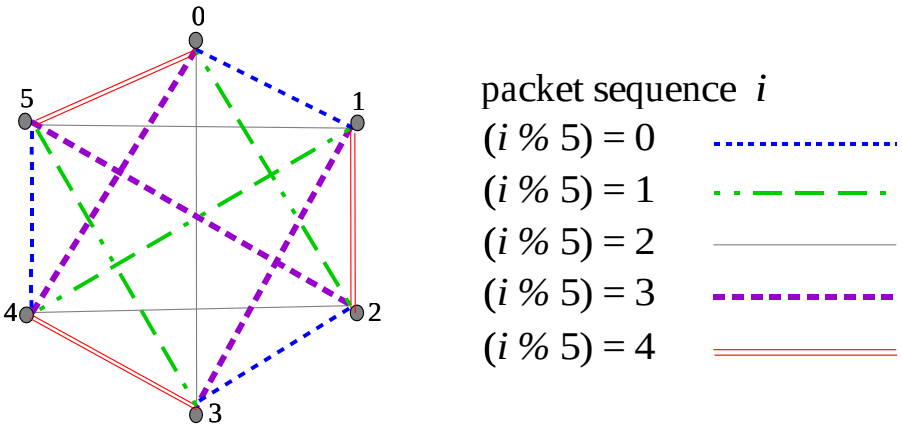


Figure 2: The resulting packet transmission sequences by the synchronous shuffle exchange algorithm using edge-color permutation on a 6-node cluster

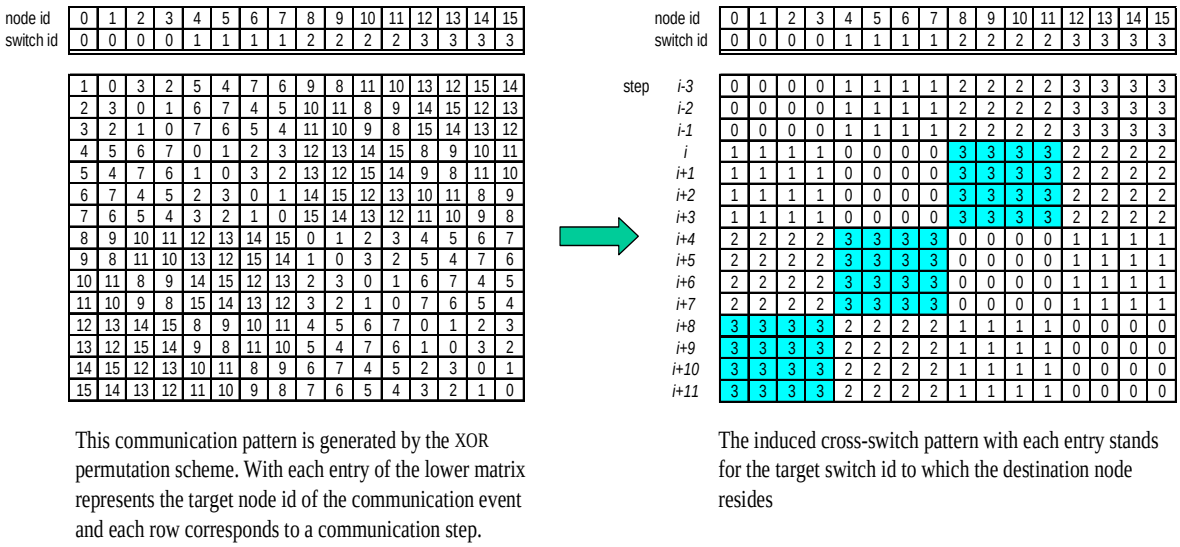


Figure 3: An example permutation in which global windowing alone fails to regulate the traffic.

Switch/uplink	Architecture	B_L
Alcatel PR2200	Shared-buffered	820
IBM 8275-326	Input-buffered	43
IBM GE uplink	Input-buffered	45

Table 1: The B_L parameter of different switches in our experimental setup

node id	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15
switch id	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
	4	0	12	8	5	1	13	9	6	2	14	10	7	3	15	11
	8	12	0	4	9	13	1	5	10	14	2	6	11	15	3	7
	12	8	4	0	13	9	5	1	14	10	6	2	15	11	7	3
	1	5	9	13	0	4	8	12	3	7	11	15	2	6	10	14
	5	1	13	9	4	0	12	8	7	3	15	11	6	2	14	10
	9	13	1	5	8	12	0	4	11	15	3	7	10	14	2	6
	13	9	5	1	12	8	4	0	15	11	7	3	14	10	6	2
	2	6	10	14	3	7	11	15	0	4	8	12	1	5	9	13
	6	2	14	10	7	3	15	11	4	0	12	8	5	1	13	9
	10	14	2	6	11	15	3	7	8	12	0	4	9	13	1	5
	14	10	6	2	15	11	7	3	12	8	4	0	13	9	5	1
	3	7	11	15	2	6	10	14	1	5	9	13	0	4	8	12
	7	3	15	11	6	2	14	10	5	1	13	9	4	0	12	8
	11	15	3	7	10	14	2	6	9	13	1	5	8	12	0	4
	15	11	7	3	14	10	6	2	13	9	5	1	12	8	4	0

After applying the contention-aware mapping to re-map each node's logical id, a new communication pattern is generated with the same XOR permutation scheme.

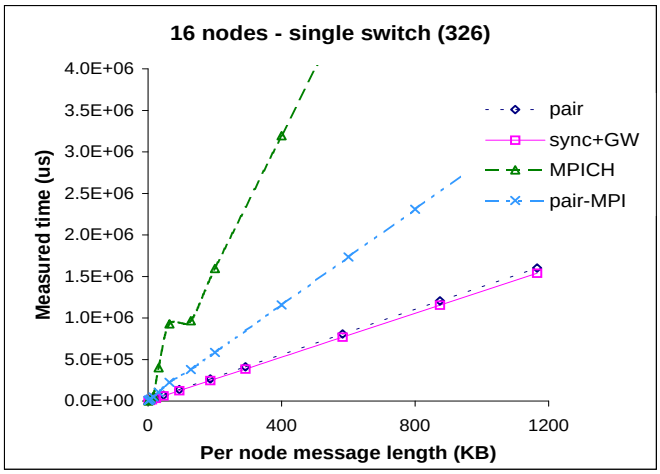
node id	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15
switch id	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
	1	1	1	1	0	0	0	0	3	3	3	3	2	2	2	2
	2	2	2	2	3	3	3	3	0	0	0	0	1	1	1	1
	3	3	3	3	2	2	2	2	1	1	1	1	0	0	0	0
	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
	1	1	1	1	0	0	0	0	3	3	3	3	2	2	2	2
	2	2	2	2	3	3	3	3	0	0	0	0	1	1	1	1
	3	3	3	3	2	2	2	2	1	1	1	1	0	0	0	0
	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
	1	1	1	1	0	0	0	0	3	3	3	3	2	2	2	2
	2	2	2	2	3	3	3	3	0	0	0	0	1	1	1	1
	3	3	3	3	2	2	2	2	1	1	1	1	0	0	0	0
	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
	1	1	1	1	0	0	0	0	3	3	3	3	2	2	2	2
	2	2	2	2	3	3	3	3	0	0	0	0	1	1	1	1
	3	3	3	3	2	2	2	2	1	1	1	1	0	0	0	0

The induced cross-switch pattern with each entry stands for the target switch id to which the destination node resides

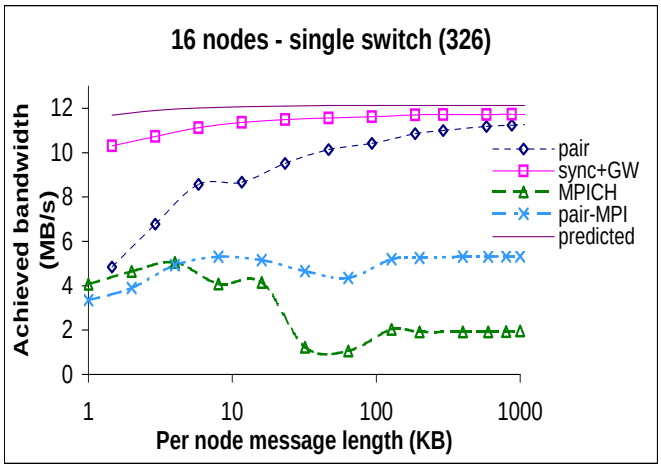
Figure 4: The resulting communication pattern after applying the contention-aware permutation scheme.

Algorithm 2 Contention-aware Synchronous Shuffle Exchange Algorithm

```
set η = 0
for (s = 1 to k) & (r = 1 to k) in parallel do
    for (i_s = 1 to p-1) & (i_r = 1 to p-1) do
        to = φ_s( ϕ(physical id), i_s)
        from = φ_r( ϕ(physical id), i_r)
        if (η < W_g) then
             status = send_item_to( to_s, to )
             if (status = SUCCESS) then
                 inc i_s
                 inc η
             fi
        status = recv_item_from( from_r, from )
        if (status = SUCCESS) then
            inc i_r
            dec η
        fi
    endfor
endfor
```



(a) Measured execution time



(b) Achieved bandwidth

Figure 5: Performance of contention-aware synchronous shuffle exchange on a single input-buffered switch. (Legends: sync - synchronous shuffle; pair - pairwise; GW - global windowing)

Algorithm 3 Generalized Pairwise Exchange Algorithm

```
round = odd(p) ? p : p−1
for i = 1 to round do
    partner = edgeColor( i, myid, p)
    if (partner = −1)
        No match partner, idle on this round
    else
        for (s = 1 to k) & (r = 1 to k) in parallel do
            status = send_item_to( partner_s, partner)
            if (status = Success) then
                inc s
            endif
            status = recv_item_from( partner_r, partner)
            if (status = Success) then
                inc r
            endif
        endfor
    endif
endfor
```

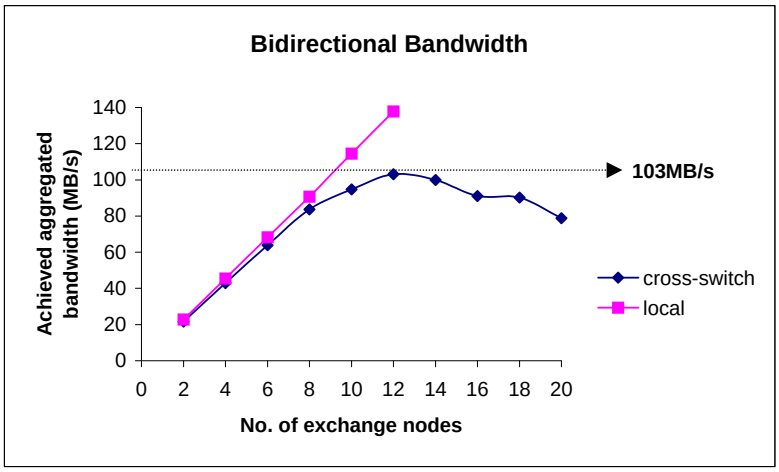
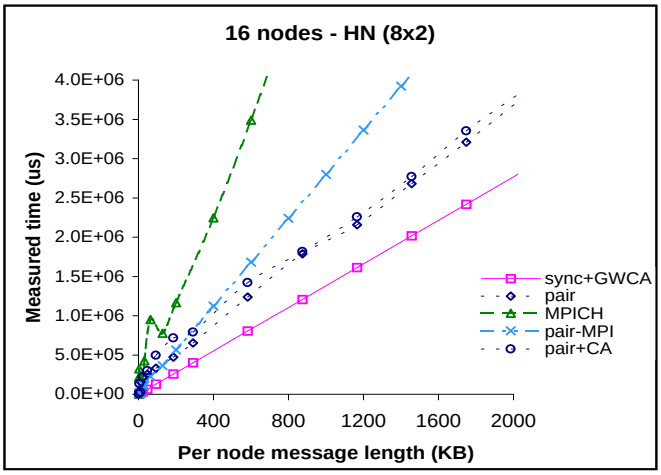
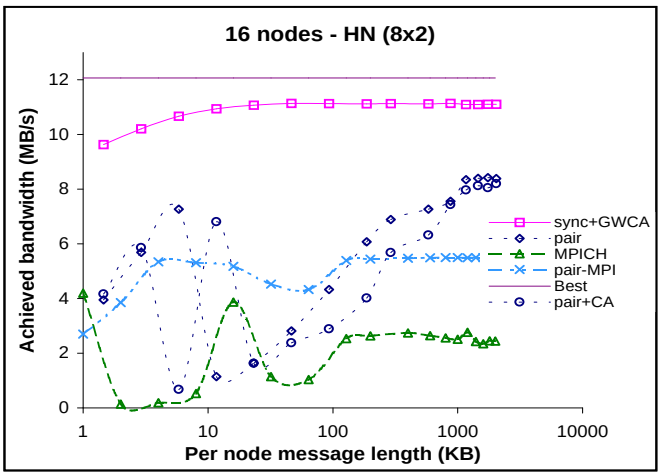


Figure 6: The achieved performance of the 10X2 hierarchical network under multiple bidirectional message exchanges. (Legends: cross-switch - measured aggregated bandwidth over the hierarchical network; local - measured aggregated bandwidth on the single switch)

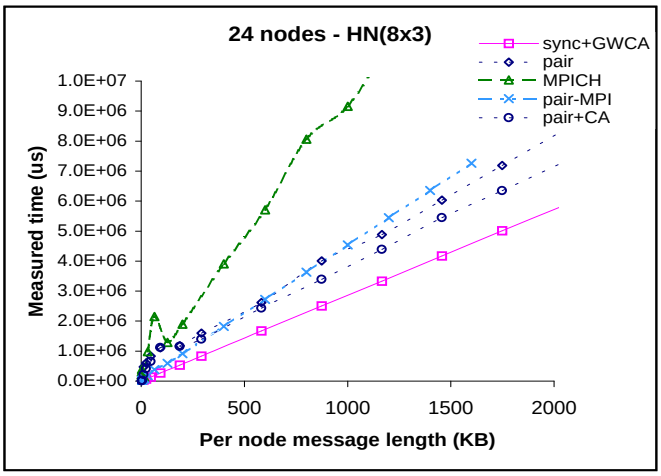


(a) Measured execution time

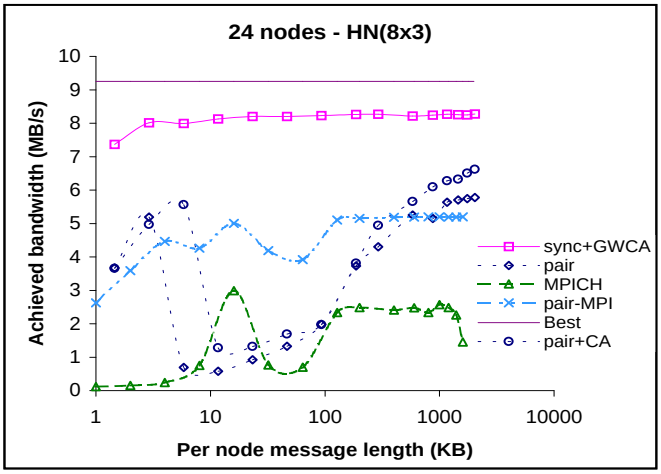


(b) Achieved bandwidth

Figure 7: The performance of contention-aware synchronous shuffle exchange on the 8x2 configuration - 8 nodes connect to each FE switch, which is connected to the PR2200. (Legends: GWCA- global windowing plus contention-aware permutation scheme; CA-contention-aware permutation scheme only)

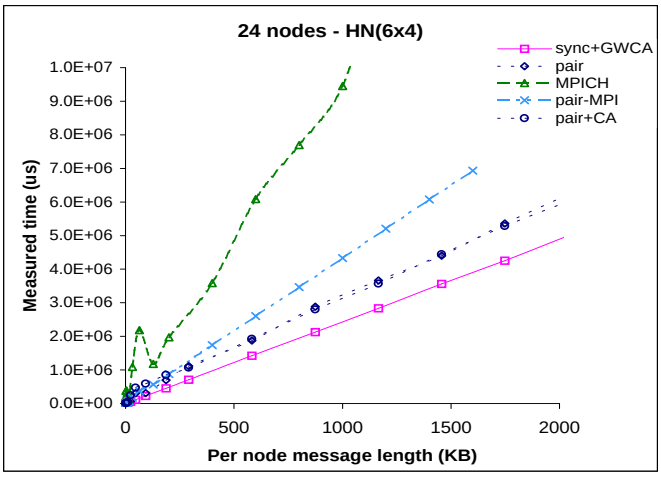


(a) Measured execution time

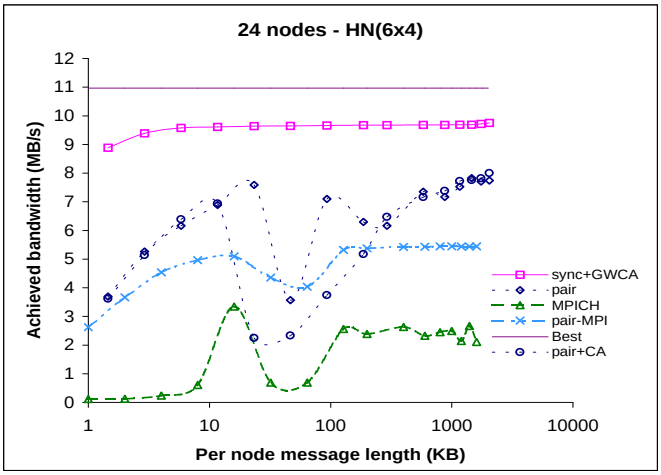


(b) Achieved bandwidth

Figure 8: Performance of different complete exchange implementations on the 8x3 hierarchical configuration

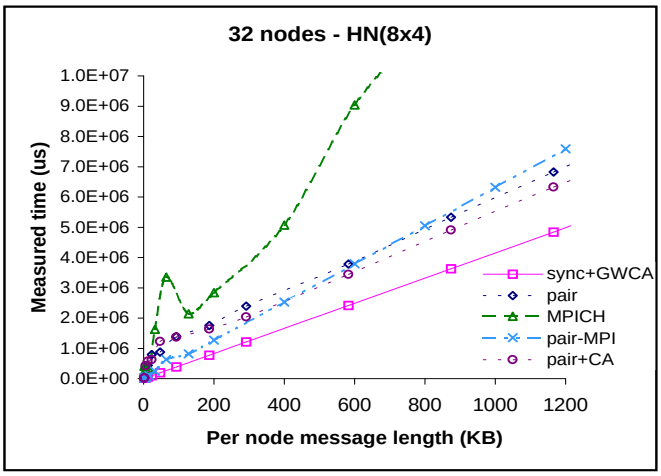


(a) Measured execution time

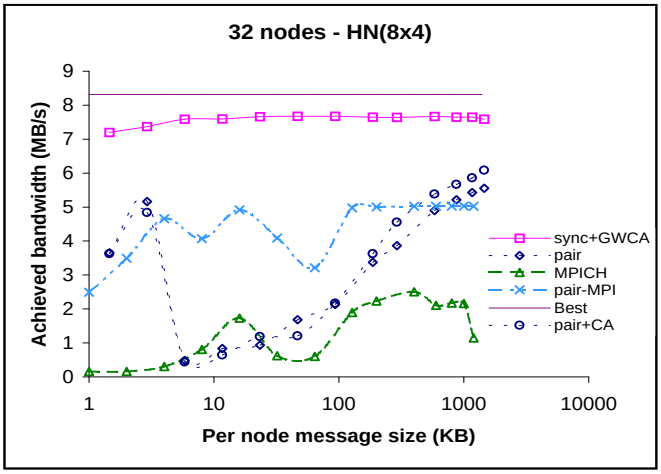


(b) Achieved bandwidth

Figure 9: Performance of different complete exchange implementations on the 6x4 hierarchical configuration



(a) Measured execution time



(b) Achieved bandwidth

Figure 10: Performance of different complete exchange implementations on the 8x4 hierarchical configuration