Contention-free Complete Exchange Algorithm on Clusters*

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

To construct a large commodity cluster, a hierarchical network is generally adopted for connecting the host machines, where a Gigabit backbone switch connects a few commodity switches with uplinks to achieve scaled bisectional bandwidth. This type of interconnection usually results in link contention and has congestion developed at the uplink ports. Moreover, the non-deterministic delays on scheduling communication events in clusters accelerate the building up of congestion amongst these uplink ports, which lead to severe packets drop and hinder the overall performance. In this paper, we focus on the practical design of high-speed complete exchange algorithm on a commodity cluster interconnected by a hierarchical Ethernet-based network. By exploiting some architectural characteristics of the interconnection in optimizing the performance of a complete exchange algorithm, we introduce a congestion control mechanism - global windowing that monitors and regulates the traffic load, together with a permutation scheme - reorder scheme that effectively alleviates the congestion problem. We evaluate our algorithm and compare its performance with other algorithms in a PC cluster connected by various types of switches, including Gigabit Ethernet, input-buffered and shared-memory Fast Ethernet switches.

1. Introduction

Commodity supercomputing is one of the targets in building clusters. Being as one form of message passing machines, the performance of clusters depends largely on the performance of the interconnection network. With the introduction of low-latency communication support [3, 4, 9], software overheads induced in communication have been significantly reduced. This allows applications to push data faster into the network, as well as users have more control on scheduling communication events.

However, having the capability to drive the network in higher speed does not guarantee to achieve good performance. Contention problems can happen in host node, network link and switch, which adversely affect the overall performance. Node contention happens when multiple data packets are contended for the receive channel of a node, while link contention occurs when two or more packets share a communication link. And switch contention is induced by the unbalance of traffic flow through the switch, which results in overflow of the switch buffer.

Complete exchange, also named as all-to-all personalized communication, is a typical example of showing how contention problems are crucial to the overall performance. This is a collective operation that takes place with a set of processes, and each process has a distinct set of data to transmit to every other process in the system. A common approach to avoid contention loss is to design a communication schedule that prevents building up of congestion, which results in packet drop. However, some contention-free schedules demand on using a tightly synchronized scheme [2, 12], which brings on another type of overheads, the synchronization overhead. In cluster environment, due to the distributed nature, it is difficult to impose such a lock-step schedule. For example, cluster node has its own local clock and process scheduler, and there is no coordination in scheduling those communication events. In addition, most of the synchronization operation is implemented by software means. Thus, this further impedes on normal data communication and contends for network resources.

In this paper, we propose an efficient communication schedule for running the complete exchange operation on clusters, which are interconnected by a hierarchical Ethernet-based network. The key feature of this communication scheme is the *proactive* approach in handling congestion. We try to avoid contention in the first place by having a communication schedule which prevents contention at the node and switch. If congestion does develop, the communication scheme regulates the traffic to avoid further building up of congestion.

This congestion control scheme is different from tra-

*This research was supported by Hong Kong RGC grant 10201701 and HKU CRGC grant 10203009.

ditional congestion control schemes. Conventional mechanisms for controlling congestion are based on end-to-end windowing schemes [8], however, they are not suitable for collective operations in high-speed networks. First, 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. Inevitably, packets are lost and performance suffers. Second, the feedback information from the network is usually outdated due to the propagation delay, and hence, any reactive action taken may be too late. Third, end-to-end windowing only provides isolated 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. However, in cluster computing, the traffic pattern is predictable in the case of collective operations on a bounded-size enclosed network.

Our complete exchange algorithm on hierarchical network is derived from an algorithm, which is developed on a theoretical non-blocking network [11]. By introducing a *global windowing* concept to all participating nodes, they are responsible to monitor and regulate the traffic loads to avoid congestion. Based on architectural features like the network buffering capacity and the balancing of upstream and downstream flows, we derive the global windowing scheme and the reorder scheme, which transform the algorithm to work efficiently on the hierarchical network.

The rest of the paper is organized as follows. Section 2 lays down the architectural characteristics of the Ethernet-based hierarchical network, and defines the problems associated with this type of network. In Section 3, we provide a simple abstract model of the network to aid our analysis. Section 4 presents a bandwidth-optimal complete exchange algorithm on Ethernet-based hierarchical networks and non-blocking networks. The experimental results of this algorithm are presented in Section 5. Finally, the conclusions are presented in Section 6.

2. Hierarchical Network

Ethernet-based network is the most widely used local area networking (LAN) technology. Although standard Ethernet has limited bandwidth in supporting cluster computing, its enhanced versions, such as Fast Ethernet (FE) and Gigabit Ethernet (GE), provide sufficient bandwidth with a steady upgrade path in building commodity clusters. Therefore, many self-made clusters are using FE as the base of their interconnections, e.g. Avalon [1], ICEBox [6], KLAT2 [7], VALHAL [13], etc.

There are generally two approaches in building a Ethernet-based cluster with a few hundreds nodes. First, using a single high-performance, high port density chassis switch to connect all machines [5]. Second, using a hierarchical network, in which cluster nodes are connected to FE switches and using the GE as a backbone network to interconnect all FE switches. Given that the backbone capacity of the interconnection network is greater than the demanding bandwidth of the whole cluster, both approaches support a fully connected network with similar performance. In reality, they are suffered from some architectural constraints that limit their actual performance. Our previous study of complete exchange on a single router switch [11] has revealed that the buffering mechanism used within the switch could hinder its actual performance, and we have proposed a sub-optimal algorithm in dealing with the situation.

On the hierarchical network, connections between different technologies are bridged by one or more uplink ports. Since both FE and GE are mutated from the standard Ethernet, they are using the same mechanism in switching packets. Packet received on an ingress port is switched to the corresponding egress port according to the destination MAC address found at the head of each packet. The switch uses its address lookup table to make this forwarding decision.

The requirement of having higher channel bandwidth for the uplinks limits the switching technique adopted on this type of interconnection. As cut-through switching is only possible for ports operate at the same data rate, this 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 as store-and-forward switching causes cumulation of upstream and downstream packets over those uplink ports.

Apparently, even under a node contention-free schedule of the complete exchange algorithm, sharing of uplinks is needed. For instance, all cross-switch traffics are going via the uplinks to the GE switch, packets have to contend for the shared uplinks even though they are from distinct sources and to distinct destinations. In theory, under a node contention-free schedule, the distribution of data packets should be well balanced, thus, 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 distributed nature of the clusters does not guarantee to adhere to a tightly synchronized schedule. For example, 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, the buffering mechanism used within the switches could affect their overall performance. While there are many variations in switch architectures, most switches fall into one or a combination of three basic types: input-buffered, output-buffered and shared-buffered [14].

For the input-buffered architecture, incoming packets

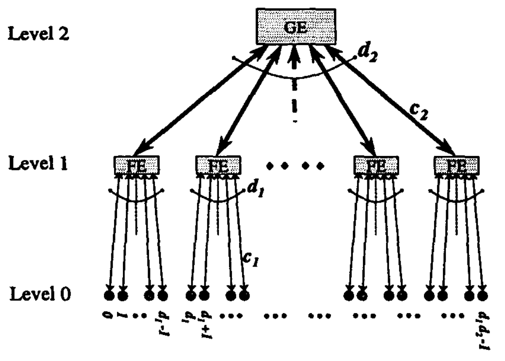

Figure 1. Interconnection topology of the two-level switch hierarchy

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.

3. System Model

In our model, a cluster is defined as a collection of autonomous machines that are interconnected by a switch-based network. To simplify the discussion, a two-level switch hierarchy as depicted in Figure 1 is discussed. For this tree topology, 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 uplink is capable of handling 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 $d_1 d_2 c_1$.

Switches are the basic building blocks of this hierarchical network. We assume they are packet-switched, pipelined network, and operate in full-duplex configuration. Buffers are provided in the switches for temporary storage, but the amount of buffers is assumed to be finite. All cluster nodes communicate via this switch-based network and assume to be homogeneous. In this study, we assume that each node equips with one network adapter, which supports concurrent send and receive operations.

We analyze the performance of the complete exchange algorithms based on a communication model discussed in [10]. 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 communication protocol in use. 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[1]. 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 in-

[1]For a simple switch, we only have one B_L value, either associates with the whole switch if it is a shared-buffer switch, or associates to a switch port if it is an input-buffered or output-buffered switch. For a switch with uplink module, we may have two B_L values which depend on the architecture used in the uplink module. One is associated to the switch/port as above, and the another is associated to the uplink port.

duced 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 :

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

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.

4. Synchronous Shuffle Exchange Algorithm

Figure 2 presents the synchronous shuffle exchange algorithm, which is proposed in [11] and is a bandwidth-optimal algorithm on any non-blocking network. 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 [11].

If every operation is executed on schedule, the permutation scheme of the synchronous shuffle exchange can be finished in minimal time. The following formula is the predicted communication cost for this complete exchange algorithm on a non-blocking switch based on our system model,

$$T_{sync} = O_s + kg(p-1) + L - g + O_r + U_r \qquad (2)$$

However in reality, as this schedule induces intensive communications and demands on logical synchrony, any non-deterministic delays between events could break the synchronism and result in congestion developed in the switch. For example, non-coordinated process scheduling would introduce randomness. We have shown in our previous study that not all switches can withstand such an intensive communication pattern for an extended period of time.

The assumption of logical synchrony on all cluster nodes is generally too idealistic on the case of commodity clusters, which have not hardware synchronization support. To impose this synchrony, explicit synchronization

```
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 = $\varphi_s$(myid, $i_s$)
      from = $\varphi_r$(myid, $i_r$)
      if (send_item_to(to$_s$,to) == success)
         inc $i_s$
      endif
      if (recv_item_from(from$_r$,from) == success)
         inc $i_r$
      endif
   endfor
endfor
```

Figure 2. The Synchronous Shuffle Exchange algorithm

operations can be used. 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 the synchronization operation. Since performance loss is caused by oversubscribing the network that induces packet loss at the bottleneck region, the best solution to avoid contention loss is to prevent oversubscription to the network. That can be done by applying a global congestion window on each node to ensure fair sharing of resources among cluster nodes.

4.1. Global Window Congestion Control

The conjecture behind the contention problem induced by the synchronous shuffle exchange algorithm is the non-deterministic delays on communication events. With the hierarchical network, two more sources of delay could contribute to this non-determinism: a) the queuing delay at the uplinks and b) the difference of network latencies between nodes within a switch and across switches. To achieve optimal performance on the hierarchical network, sharing of uplinks, thus having link contention, is a fact that we have to face. Although mild contention increases network delay, it does not hurt the performance much unless the congestion persists for a long period of time, which results in buffer overflow. Therefore, a congestion control scheme is needed to prevent overloading the network.

We adopt a proactive approach in congestion control, as reactive schemes are not suitable for high-speed communication. Based on the network capacity, each node is assigned with a predefined resource limit, and force them to regulate their traffic loads below this limit. This ensures that no source will exceed its allowed traffic capacity and avoids congestion loss. For our complete exchange algorithm, all cluster nodes are assigned with a global window (W_g) at the beginning of the operation.

During the execution flow, at i^{th} communication step, a process is sending a data packet to another process according to a 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 dramatically. The above observation implies that to avoid overflowing the network buffers, we need to regulate the number of outstanding packets (η). The basic principle behind this scheme is simple. If a process 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. In [9], we have explained how to evaluate the B_L parameter of the switch from a user perspective. 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.

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 FE 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 packets directed to each FE switch's 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

$$W_g = \left\lfloor \frac{B_L}{\nu} \right\rfloor \tag{4}$$

4.2. Reorder Scheme

However, knowing the value of W_g is a necessary but not sufficient condition to avoid congestion loss. This is because a) W_g is derived from taking the average traffic load, and b) 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

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

Figure 3. An example permutation in which global windowing alone fails to regulate the traffic. (The bottom matrix represents the induced cross-switch traffics.)

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 still experience contention loss even though the 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 originated from switch 2 and headed 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 this contention until after step i+8 when the global windows of processes in switch 3 become saturated. 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.

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, 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 permutation is by the *reorder scheme*, which is a rearrangement of an existing permutation. Observed that the original per-

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

Figure 4. The resulting communication pattern after applying the reorder mapping scheme.

```
Set $\eta$ = 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 = $\varphi_s(\phi(physical\ id), i_s)$
      from = $\varphi_r(\phi(physical\ id), i_r)$
      if ($\eta < W_g$) &&
         (send_item_to(to$_s$, to) == success)
         inc $i_s$
         inc $\eta$
      endif
      if (recv_item_from(from$_r$, from) == success)
         inc $i_r$
         dec $\eta$
      endif
   endfor
endfor
```

Figure 5. Synchronous Shuffle Exchange algorithm with global windowing and reorder scheme

mutation is obtained by some simple functions (φ) which operates on inputs such as current communication step and node id. A simple method to rearrange the original permutation is by redefining a mapping between logical node id to its physical id. One example of such reorder scheme (ϕ) can be as follows:

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

Carry on with the previous example, if we apply the xor permutation on the reordered 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, and hence, greatly relieves the contention at the uplink ports and improves the effectiveness of our congestion control scheme.

Based on the global windowing scheme and the reorder scheme, we have modified the synchronous shuffle exchange algorithm to work efficiently on the two-level hierarchical network, and the modified algorithm is given in Figure 5.

Switch/uplink	Architecture	B_L
Alcatel PR2200	Shared-buffered	820
Intel 510T	Shared-buffered	1990
IBM 8275-326	Input-buffered	45
Intel GE uplink	Shared-buffered	1990+510
IBM GE uplink	Input-buffered	45

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

5. Experimental Results

Our experimental platform is a cluster consists of 16 standard PCs running Linux 2.0.36. Each cluster node equips with a 450MHz Pentium III processor with 512KB L2 cache, 128MB of main memory, a Digital 21140A FE card and is connected to a FE Switch. We use the Directed Point (DP) communication system [8] to drive the network and conduct all our experiments. We have implemented a simple Go-Back-N protocol to support limited reliability on DP. In our studies, we have 4 FE switches and one GE switch to set up various configurations in evaluating 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 up at most 4 ports in our experiments. Two FE switches are from IBM, which are 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 switch to connect to the Gigabit backbone. Another pair of FE switches are the Intel 510T switches, which are revealed as shared-buffered architecture. The 510T is a 24-port switch with only 2.2 Gbps backbone capacity. Both switches are also equipped with add-on GE modules for connecting to the PR2200. Table 1 summaries all the buffer parameters of the above switches, which are used in our algorithm to compute the global windows (W_g) for different network configurations.

5.1. Configuration One - 16x1

In Figure 6, we measured three complete exchange implementations on the 16-node cluster interconnected by an input-buffered switch (8275-326). They are the synchronous shuffle with global windowing (sync+GW), original synchronous shuffle (sync), and the popular pairwise exchange (pairwise) [11]. The experiment is conducted with each node sending k packets of size 1492 bytes to every node in the cluster, which is ranged from k=1 to 2000. We used a relative metric, called *achieved bandwidth*, to quantify the efficiency of the algorithm in utilizing the network.

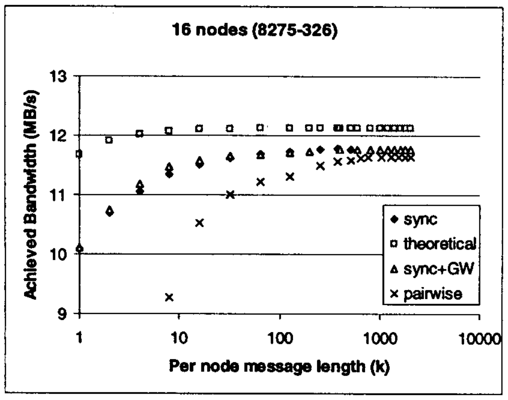

Figure 6. Performance of Synchronous Shuffle Exchange on a single input-buffered switch.

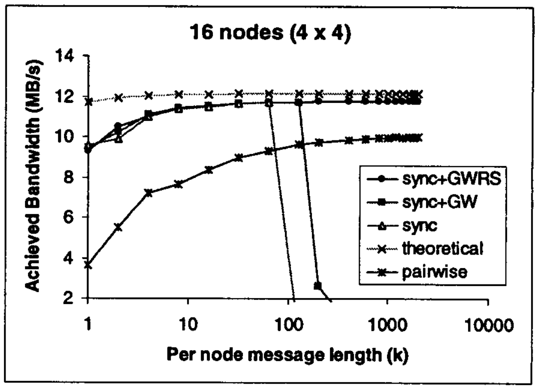

Figure 7. Performance of Synchronous Shuffle Exchange on the 4x4 configuration - 4 nodes connect to a FE switch, which is connected to the PR2200.

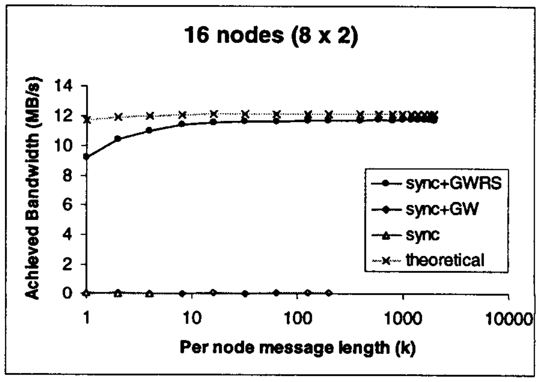

Figure 8. Performance of Synchronous Shuffle Exchange on the 8x2 configuration - 8 nodes connect to the IBM 326, which is connected to the PR2200.

This metric is computed by dividing the total data sizes injected into the network by each node with the measured communication time.

The results show that both versions of the synchronous shuffle algorithms have similar performance, which is peak at 97% of the available bandwidth. When compared to the theoretical performance (eqt. 2), the synchronous shuffle exchange algorithm has its efficiency ranged from 87% to 97% of the theoretical bandwidth. When compared with the pairwise exchange, which has a higher synchronization overhead, the results show that the synchronous shuffle algorithm can effectively mask away those synchronization overhead and achieve better performance, especially when exchanging small size messages. However, the performance of the original algorithm degrades significantly after k>512, which corresponds to the total message length of 11 MByte per node. Meanwhile, with the addition of the global windowing scheme to the synchronous shuffle algorithm, it continues to operate efficiently as the problem size scales.

5.2. Configuration Two - 4x4

With a different network configuration, we compare four different implementations: synchronous shuffle with global windowing and reorder scheme (sync+GWRS), original synchronous shuffle (sync), original algorithm with global windowing only (sync+GW) and the pairwise exchange (pairwise) algorithm. The results are shown in Figure 7. As we are using a two-level configuration, there are total 5 switches with different architectures. When comparing the buffer capacities of those switches, the obvious limitation is on the uplink module of the IBM 326 switch. With the corresponding B_L value, we compute the W_g factor, which is equal to 14.

The results show that synchronous shuffle exchange with global windowing and reorder scheme performs consistently and efficiently in this test. It effectively masks away the synchronization and contention overheads, which appears to have a more significant influence on the pairwise exchange algorithm when executes on this hierarchical network. When compared with the original algorithm, we see that the performance of the original algorithm starts to degrade at k>64, which means there is significant contention build-up on the uplink ports. By applying the global windowing without reorder scheme, we can slightly extend the contention tolerance of the synchronous shuffle algorithm until k>128. This experiment shows that both global windowing and reorder scheme are required to monitor on the congestion problem.

5.3. Configuration Three - 8x2

In this configuration, we interconnect two IBM 326 switches to the PR2200, with each switch connects to 8 nodes. It is clear that the performance bottleneck would lie on the uplink ports. Base on this configuration, the computed value of W_g is 10, and the results are shown in Figure 8. Same as the previous two experiments, the modified synchronous shuffle algorithm performs the best.

However, both the original synchronous shuffle and the original algorithm with global windowing perform poorly in this test. This is because a) the Go-Back-N protocol is known to suffer badly when the error rate is high, and b) the GE module of the model 326 is not performing as good as it claims. By using a bi-directional exchange benchmark test, we revealed that the uplink ports could only support up to 5 active bi-directional channels without packet loss. It seems like the circuitry of the uplink modules cannot catch up with the Gigabit performance. We have tested with other configurations, such as 6x2 and 4x2, to corroborate with the above observations. This experiment shows that without the reorder scheme, the global windowing scheme cannot avoid congestion build-up on its own.

6. Conclusion

This paper presented an efficient implementation of complete exchange operation on a Ethernet-based hierarchical network. The hierarchical network is based on a Gigabit switch as the backbone to which all Fast Ethernet switches are connected. We demonstrated that the contention problems on such network - link, node, and switch contention, can severely affect the overall performance of the clusters. To avoid congestion loss on this type of network, we propose the synchronous shuffle exchange algorithm with congestion control scheme. This algorithm makes uses 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 reorder scheme in generating a communication schedule, which is both node and switch contention-free, as well as supports 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 hierarchical network model is a practical design to construct large-scale clusters. With this system configuration, clusters can be scaled up to hundreds of nodes, and support enough bandwidth for high-speed communication. Our research can be used in any combination of Ethernet-based switched network, which we belief, over 50% of the self-made clusters are based on. And the concept is applicable to future technologies, such as 10Gigabit Ethernet, which simply extends the topology to multi-level hierarchy.

References

[1] The Avalon Cluster. (`http://cnls.lanl.gov/avalon/`)

[2] S.H. Bokhari and D.M. Nicol, "Balancing Contention and Synchronization on the Intel Paragon", in *IEEE Concurrency* Vol. 5, No. 2, 1997, pp 74-83.

[3] G. Chiola and G. Ciaccio, "GAMMA: a Low-cost Network of Workstations Based on Active Messages", in the *5th EUROMICRO workshop on Parallel and Distributed Processing (PDP'97)*, January 1997.

[4] T. von Eicken, A. Basu, V. Buch and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing", in the *15th ACM Symposium on Operating System Principles*, December 1995.

[5] Extreme Networks. (`http://www.extremenetworks.com/products/`)

[6] The ICEBox Cluster. (`http://www.chpc.utah.edu/ice/`)

[7] The KLAT2 Cluster. (`http://aggregate.org/KLAT2/`)

[8] M. Sidi, W.Z. Liu, I. Cidon and I. Gopal, "Congestion Control Through Input Rate Regulation", in *IEEE Transactions on Communications*, Vol. 41, No. 3, March 1993, pp 471-477.

[9] C.M. Lee, A. Tam and C.L. Wang, "Directed Point: An Efficient Communication Subsystem for Cluster Computing", in *the International Conference on Parallel and Distributed Computing Systems (IASTED)*, October 1998.

[10] A.T.C. Tam and C.L. Wang, "Realistic Communication Model for Parallel Computing on Cluster", in the *1st IEEE Computer Society International Workshop on Cluster Computing (IWCC'99)*, December 1999.

[11] A.T.C. Tam and C.L. Wang, "Efficient Scheduling of Complete Exchange on Clusters", in the *ISCA 13th International Conference On Parallel And Distributed Computing Systems (PDCS2000)*, August 2000.

[12] Y.C. Tseng and S.K.S. Gupta, "All-to-All Personalized Communication in a Wormhole-Routed Torus", in IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 5, May 1996, pp 498-505.

[13] The VALHAL Cluster. (`http://www.fysik.dtu.dk/CAMP/valhal.html`)

[14] J. Walrand and P. Varaiya, High-Performance Communication Networks, Morgan Kaufmann Publishers, 1996.