Empirical Studies of Collective Communication in Heterogeneous Cluster Systems

Li-Ru Chen
Department of Computer Science and Information Engineering
National Chung Cheng University

Pangfeng Liu
Department of Computer Science and Information Engineering
National Taiwan University
pangfeng@csie.ntu.edi.tw

August 1, 2002

Abstract

All-to-all personalized communication is a very important collective communication pattern in high performance computing applications. The communication pattern is regular and can be efficiently implemented in a homogeneous cluster system. However, due to economical consideration, in today's high-performance cluster computing environments, heterogeneity is very common and processors could have very different computation and communication profile, even in the same cluster. As a result new algorithms must be deployed in order to achieve good communication efficiency in heterogeneous environments. This paper discusses scheduling algorithms for Heterogeneous Networks of Workstations (so called HNOW) systems, and designs simulators to conduct experiments on them. We want to make communication events with all-to-all personalized and all-to-some communication efficiently scheduled, and the final goal is to reduce the completion time of the communication schedule for heterogeneous networks of workstations.

1 Introduction

With the speed of microprocessors increasing and cost decreasing, high-performance workstations/PCs have become the building blocks of choice for high-performance parallel computing systems. While traditional parallel machines typically exploit custom high-performance networks, modern commodity networks (such as Fast Ethernet, ATM, HiPPI, Servernet, and Myrinet) are becoming widely available at a reasonable cost. High performance parallelism is achieved by dividing the computation into manageable subtasks, and distributing these subtasks to the processors within the cluster.

These off-the-shelf high-performance processors provide a much higher performance-to-cost ratio so that high performance clusters can be built inexpensively. In addition, the processors can be conveniently connected by industry standard network components. For example, Fast Ethernet technology provides up to 100 Mega bits per second of bandwidth with inexpensive Fast Ethernet adaptors and hubs. The conjugation of high-speed networks and high-performance microprocessors makes networks of workstations/PCs (so called NOW) appealing vehicles for cost-effective parallel computing. As a whole, clusters are gradually becoming an alternative to MPPs and supercomputers in many areas of applications [1].

Most of the literature on cluster computing emphasizes on *homogeneous* cluster – a cluster consisting of the same type of processors. However, we argue that heterogeneity is one of the key issues that must be addressed in improving parallel performance of NOW. Firstly, it is always the case that one wishes to connect as many processors as possible into a cluster to increase parallelism and reduce execution time. Despite the increased computing power, the scheduling management of such a *heterogeneous network of workstation* (HNOW) becomes complicated since these processors will have different performances in computation and communication. Secondly, since most of the processors that are used to build a cluster are commercially off- the-shelf products, they will very likely be outdated by faster successors before they become unusable. Very often a cluster consists of "leftovers" from the previous installation, and "new comers" that are recently purchased. The issue of heterogeneity is both scientific and economic. Due to the commodity nature of workstations and networking equipments, LAN environments are gradually becoming heterogeneous. The heterogeneity could be due to the difference in processing speed and communication capability of the workstations, or coexistence of multiple network architectures or communication protocols. Therefore, in today's high-performance cluster computing environments, heterogeneity is common and its extent will continue to grow in the future. This trend is forcing networks of workstations to be redefined as Heterogeneous Networks of Workstations (HNOW).

The collective communication, where all processors contribute data to get a result that may arrive at one or all processors, provides important functionality for many applications, and thus efficient implementations of the collective communication is crucial for achieving maximum performance in message-passing systems.

All-to-all personalized communication is common in high performance computing applications. For example, let us consider the matrix transposition problem [2]. At the beginning, each row of a two-dimensional matrix is distributed among all processors. Since now the transposed matrix would its columns distributed among all processors, each processor must use all-to-all personalized communication to send different data to all the processors.

2 Problem Description

All-to-all personalized communication is common in HPC application, such as most noticeable matrix transposition and the fast Fourier transform, etc [3]. For all-to-all personalized communication,each node sends different messages to each of the other nodes. The communication pattern is regular and can be efficiently implemented in a homogeneous cluster system where all processors have similar communication and computation capabilities.

For all-to-all personalized communication, each node sends different messages to each of the other nodes. This is very different from the all-broadcast operation, in which each node in the system sends the same message to all the other nodes. The all-to-all personalized communication is also different from the scatter operation, in which one node sends different messages to each of the other nodes. We can think that all-to-all personalized communication is equal to all scatter – each node scatters different messages to all the others.

Our model allows each node to participate a sending and a receiving operation at the same time, but does not allow multiple sending or receiving [2]. When a node sends a message to another node, or receives a message from another node, both are treated as communication events. As a result for a P-node system, there are P^2 communication events for all-to-all personalized communication. The message lengths of all the communication events are not necessarily the same, and consequently will not require the same amount of time to finish. In order to reduce the completion time of the communication schedule, i.e. the completion time of the last communication event, we must make all the communication events efficiently scheduled. We can know that the completion time of the communication schedule cannot be less than the sum of sending times or receiving times for any processor.

The scheduling problem could be complicated due to the presence of servers in the systems, which will incur a lot of traffics. For example, there could be WWW or ftp servers in the cluster, which will contribute a large amount of traffics and cause hot spots for the system. In addition, under large variation of network conditions, run-time load cannot be determined in advance, therefore it is important to consider the case that the transmission times are very different, even when the processors are homogeneous. We would like to find not a fixed schedule, but one that is capable of adapting to changing network conditions.

The scheduling management of such a *heterogeneous network of workstation* (HNOW) becomes complicated since these processors have very different performances in computation and communication. Heterogeneity of a cluster complicates the design of efficient collective communication protocols as a result. When the processors send and receive messages at different rates, it is difficult to synchronize them so that the message can arrive at the right processor at the right time for maximum communication throughput. On the other hand, in homogeneous NOW every processor requires the same amount of time to transmit a message. For example, it is straightforward to implement a broadcast operation as a series of sending and receiving messages, and in each phase we double the number of processors that have received the broadcast message. In a heterogeneous environment it is no longer clear how we should proceed to complete the same task.

Our problem can be formulated as follows: Given a matrix of communication costs from source to destination processors, find a good communication schedule that minimizes the total all-to-all personalized or all-to-some communication time. The system model allows a processor to participate a sending and a receiving operation simultaneously, but does not allow multiple sending or receiving. We will describe our experimental approaches on this problem in the following sections.

3 Algorithms

There are many related scheduling algorithms for toal exchange, or all-to-all personalized communication in the literature, which will be described as follow. We will also describe our randomized algorihtms for all-to-all personalized and all-to-some communication.

3.1 Matching-Based Algorithm

We describe a simple matching-based algorithm that scedules the all-to-all personalized boradcast [2]. The communciation model allows a processor to participate a sending and a receiving operation at the same time, but does not allow multiple sending or receiving. We construct a bipartite graph with P vertices on both sides. The vertices on the left side are senders, and the vertices on the right side are receivers. The edge from V_l on the left side to V_r on the right side is assigned a weight, which is equal to the cost of the communication from P_l to P_r. In each round of the algorithm, we find a maximum perfect matching in the bipartite graph and remove the edges of the matching from the graph. Since those edges removed are part of a matchiing, there is no contention among these P communication events. Since there is a well-known algorithm for finding a maximum weight complete matching with time complexity $\mathrm{O}(P^3)$ [4], and the schedule has P rounds, the total time complexity for finding all the maximum weight matchings is $\mathrm{O}(P^4)$. The inituition of the matching-based algorithm is that it is likely that the communication events scheduled within the same round have similar communication times.

3.2 Greedy Algorithm

We also describe a greedy algorithm for scheduling all-to-all communication [2]. Firstly, the communication events of each sender node are ordered in decreasing order of the communication times. In each round we find a destination node for a sender from its ordered list of receiver. The chosen destination must satisfy the following two conditions – the destination is the first node in its ordered list that has not been selected by the sender, and the destination is not the destination of the other senders in the same round. If we cannot find the destination for a sender it will be idle in this round, but has the highest priority in picking the destination in the next round. If there is no sender idle in this round, the last sender will be the first node to pick the destination node in next round. The complexity of the greedy algorithm is $\mathrm{O}(P^3)$.

3.3 Open Shop Algorithm

We also describe the open shop algorithm in this section [2, 5]. The algorithm maintain three data structures – a receiver set for every sender, and two arrays for the available times of the senders and receivers.

The open shop algorithm first finds the earliest sender that can send messages in future, then it scan the receiver set of the sender to find the destination that can receive the message at the earliest possible time. If all of the destination nodes in the receiver set are busy, the sender will sit idle. If there are two or more senders that are available at the same time, we select any of them at random. The complexity of this algorithm is $O(P^3)$ since there are P^2 communication events to schedule, and we take $O(P)$ time to select a communication event.

3.4 Caterpillar Algorithm

The total exchange scheduling problem is NP-Complete [2, 6, 7, 8] , therefore it is unlikely that we could find an optimal algorithm, hence various heuristic were developed to give reasonalbly good scheduling. One of these heuristic algorithm is called *caterpillar* algorithm [9], which is widely used in tightly coupled homogeneous systems.

We assume that there are N nodes in the system, and the caterpillar algorithm generates a schedule with N steps. In step j $(0 \leq j < N)$, each node P_i $(0 \leq i < N)$ sends a message to $P_{(i+j) mod N}$, as shown in Table 1. From now on we will illustrate these scheduling algorihtms by their corresponding timing diagrams, as shown in Table 1. Each processor will send messages to all the others according to the indeices in its column.

P_0	P_1	P_2	P_3	P_4
0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

Table 1: The sending sequence from the caterpillar algorithm for all-to-all personalized communication.

The caterpillar algorithm can also be used in all-to-some problem. The order that each node sends messages is increasing with labels of nodes, as shown in Table 2.

Assume that there are 10 source nodes and 5 receiver nodes – $\{P_3, P_4, P_5, P_6$, and $P_7\}$.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
3	3	3	3	4	5	6	7	3	3
4	4	4	4	5	6	7	3	4	4
5	5	5	5	6	7	3	4	5	5
6	6	6	6	7	3	4	5	6	6
7	7	7	7	3	4	5	6	7	7

Table 2: The sending sequence from the caterpillar algorithm for All-to-some personalized communication.

3.5 Randomized Algorithms

Similar to the caterpillar algorithm, the randomized algorithm we would like to test simply determine sending sequence by random. The intuition is that by randomly assigning the order the processors sending messgaes, we will not have messages queued for a single processor simutaneously.

Random order Each node selects a receiver at random in each step.

Random_Starting_Point_Baseline The Random_Starting_Point_Baseline algorithm first chooses a random permutation for each processor as the starting point in the timing diagram (see Table 3 for an illustration). After we decide the first destination, each node then follows the order as in the caterpillar algorithm.

- All-to-all personalized communication
  In the first row, we randomly permute {0,1,2,3,4} and place the result {0,4,3,2,1} in the first row of the timing diagram. Then we following the caterpillar pattern.

P_0	P_1	P_2	P_3	P_4
0	4	3	2	1
1	0	4	3	2
2	1	0	4	3
3	2	1	0	4
4	3	2	1	0

Table 3: The sending sequence from the Random_Starting_Point_Baseline algorithm for all-to-all personalized communication.

- All-to-some personalized communication
  The same random algorithm can be applied on all-to-some problem with a simple modification. Assume that there are 10 source nodes and 5 receiver nodes – {P_3,P_4,P_5,P_6, and P_7}. Each node first selects the first receiver at random, and then follow the caterpillar, as shown in Table 4.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
5	6	7	4	3	6	3	5	4	5
6	7	3	5	4	7	4	6	5	6
7	3	4	6	5	3	5	7	6	7
3	4	5	7	6	4	6	3	7	3
4	5	6	3	7	5	7	4	3	4

Table 4: The sending sequence from the Random_Starting_Point_Baseline algorithm for All-to-some personalized communication.

One_Random_Starting_Point_Baseline This algorithm is very similar to the previous Random_Starting_Point_Baseline. First we select a random permutation of all processors and place it in the first column of the timing diagram (Table 5). The we cyclicly shift the first column and place the result in the second column. It is equivlent to the caterpillar order, but only in horizontal direction. Please refer to Table 5 for details.

- All-to-all personalized communication:
  We place a random permutation of {0,1,2,3,4} into the first column ({4,0,3,1,2} in this case), then we follow the caterpillar horizontally. Refer to Table 5 for details.

P_0	P_1	P_2	P_3	P_4
4	0	1	2	3
0	1	2	3	4
3	4	0	1	2
1	2	3	4	0
2	3	4	0	1

Table 5: The sending sequence from the One_Random_Starting_Point_Baseline algorithm for all-to-all personalized communication.

- Similarlly this algorithm can be modified to solve the all-to-some problem. Let's assume that there are 10 nodes in the system and the receiver nodes are {P_3,P_4,P_5,P_6,P_7}. In the first column we place a random permutation ({6,3,7,5,4} in this case), then we follow the caterpillar horizontally.Refer to Table 6 for details.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
6	7	3	4	5	6	7	3	4	5
3	4	5	6	7	3	4	5	6	7
7	3	4	5	6	7	3	4	5	6
5	6	7	3	4	5	6	7	3	4
4	5	6	7	3	4	5	6	7	3

Table 6: The sending sequence from the One_Random_Starting_Point_Baseline algorithm for All-to-some personalized communication.

4 Our Experimental Approach

4.1 Communication Model

Our model allows each node to participate a sending and a receiving operation at the same time, but does not allow multiple sending or receiving [2]. For all-to-all personalized communication and all-to-some communication, we design simulators to execute four scheduling algorithms - including Random order, Caterpillar order, Random_Starting_Point_Baseline, One_Random_Starting_Point_Baseline.

In our model, each node has a sending overhead and receiving overhead [10, 11]. The sending overhead represents the initial start-up time, which is the time interval between a message is moved from its local memory to the network; and the receiving overhead is a time interval from a message is moved from local network buffer to its local memory. The network latency is the overhead for a message to travel through the network. The communication time is the sum of the sending overhead of the sender, the network latency, and the receiving overhead of the receiver. And we assume that the communication time for local transmission is zero. For ease of notation we adopt the following terminologies.

S.O. Sending overhead of the sender.

L Network latency.

R.O. Receiving overhead of the receiver.

4.1.1 Synchronous model

We will consider two communication models – synchronous and asynchronous model.

Synchronous model
Each sender must wait until the message has been received by the current receiver, then the sender can start sending another messages to other receivers. The synchronous model is shown Figure 1.

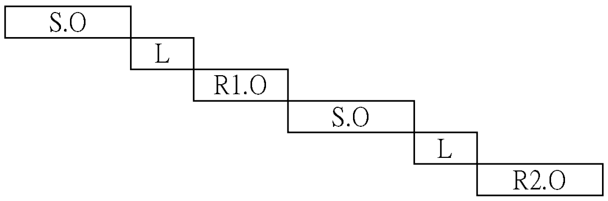

Figure 1: The communication behaviors of the synchronous model.

Asynchronous model
In the asynchronous model the sender can send another messages to the other receivers just after the sending overhead, that is, it does not need to wait until the receiver complete receiving the message. The asynchronous model is shown Figure 2.

4.2 Implementation of the Simulators

The simulator first, according to various scheduling algorithms described earlier, determines an order among the destination by which each sender sends messages to, then the simulator calculates the final completion time for each schedule. The simulation details are as follows:

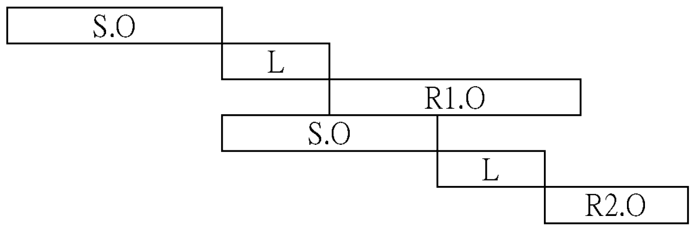

Figure 2: The communication behaviors of the asynchronous model.

4.2.1 Schedule generation

The simulator goes through the following steps to obtain a schedule.

1. We first obtain the number of receivers. There are two input parameters – the total number of nodes and the number of receiver nodes in the system. If the communication pattern is all-to-all personalized communication, the number of receivers is equal to the total number of nodes. If the pattern is an all-to-some and there are 100 nodes in the system, then we select 10% of 100 nodes as receivers. In addition, for all-to-some, we must choose the receivers from the cluster nodes.
2. Assign sending and receiving overhead of each node, and determine the value of the network latency.
3. Determine the order that each node sends messages from the given scheduling algorithm. This step may require random number generation.

4.2.2 Schedule simulation

The simulator does through the following steps to calculate the total communication time. For a synchronous model simulation we need to do the following:

- Each sender must wait until the message is received by the current receiver, then each sender can resume sending messages to the other receivers.
- For a P node system we use two P-element arrays – sendavail and recvavail, to maintain the available time of the senders and receivers.
- We use event trigger to find the sender that can participate in future send operation at the earliest time. If the receiver that the sender wants to send a message is busy, the sender sits idle. We can know if the receivers are busy by the recvavail array. If the receiver that the sender wants to send the message is idle, the message is scheduled. Then we update sendavail

and recvavail. The process repeats until all of the senders have sent messages to all of its receivers. The time that the last communication event is completed is the completion time of the communication schedule.

Notice that If there are two or more senders that can participate in future send operation at the earliest time, we select any of them at random. But we may have different completion times of the communication schedule, despite the fact that the order that each node sends messages is fixed. For example, in Figure 3 both P_1 and P_2 want to send messages to P_4 at the same time [2]. The choice of letting whether P_1 or P_2 to be scheduled first will affect the final completion times. Similarly in Figure 4 P_1 and P_2 want to send messages to P_0 at the beginning. The completion times will be different,depending on which processor we choose as the sender.

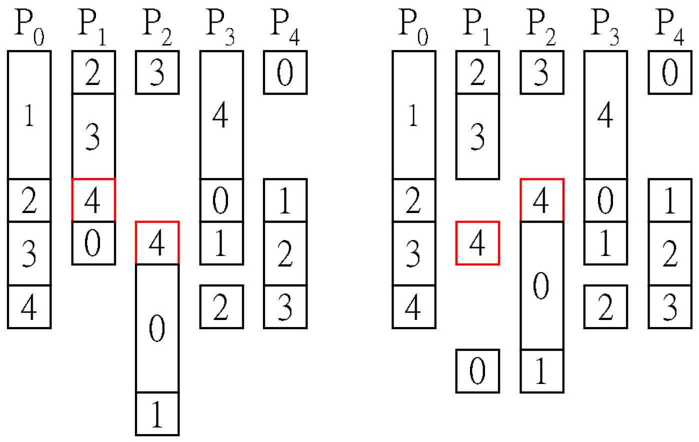

Figure 3: Fixed_order_contention_example1

For an asynchronous model simulation we need to perform the following:

- For asynchronous model, after the sending overhead of each sender, each sender can send messages to the next destination. Thus, when each sender sends the messages, we know when the messages arrive at the receivers and we just wait until then. In contrast, in synchronous model each sender cannot know when the message is received and when it can send the next message to the next receiver.
- For asynchronous model, when each sender sends the messages to all of the receivers, we can know when the messages arrive at the receivers. For each receiver, we use event trigger to find the message that will arrive in the future at the earliest time to receive it. We repeatedly use event trigger until all the messages from all senders are received by all receivers.

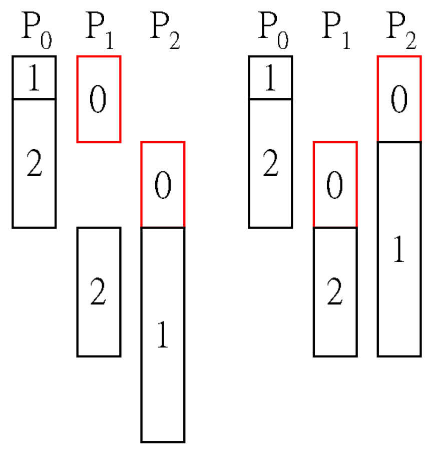

Figure 4: Fixed_order_contention_example2

- The completion time of the last receiver that finishes receiving all the messages is the completion time of the communication schedule.

In Figure 5 P_0 wants to send messages to the set $\{P_1, P_2, P_3\}$ in order. The sending overhead of P_0 is 2 and the network latency is 1.

5 Experimental Results and Observation

We describe our experimental results in this chapter. First we describe the simulation parameters and the terminology to simply the dicsussion.

The simulation parameters are determined as follows: We assume that, for every node, the sending overhead and receving overhead are the same. Then we divide these nodes into three categories according to its communciation costs, which are 1, 5 and 10. We also assume that the network latency is set to 1. For ease of explanation we use RSPB to denote the Random_Starting_Point_Baseline algorithm, and ORSPB for One_Random_Starting_Point_Baseline.

5.1 Experiment Guildlines

We classify our experiments into four categories, as shown in Table 7. We consider both all-to-all personalized communication and all-to-some scatter, and both synchronous and asynchronous model. For each of these four combinations we run simulation for all four algorithms and compare their schedules.

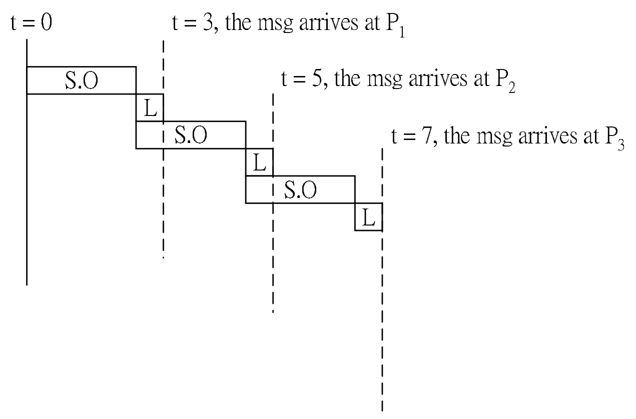
t = 0
t = 3, the msg arrives at P_1
t = 5, the msg arrives at P_2
t = 7, the msg arrives at P_3
S.O
L
S.O
L
S.O
L

Figure 5: An example of asynchronous model simulation.

We would like to compute the completion time of the communication schedule with each scheduling algorithm. First, we assign simulation parameters (the sending and receiving overhead of each node) to the system. The we generate the schedule according to different algorithms. Note that except for the caterpillar algorithm, each scheduling algorithm do require computation to determine the sending order.

In general we experiment with each scheduling algorithm for 100 times and compute the average completion time of these 100 runs. For caterpillar order the order is fixed. However, if there are two senders that could send messages to the same receiver at the same time, we select one of them at random. As a result the completion time from the caterpillar algorithm is also a random variable. If we select a bad sender it may delay some critical nodes and result in longer completion time. By taking the average of a lrage number of runs we hope to get a more reliable completion time.

Let N be the number of nodes in the system. For all-to-all personalized communication, we experiment with clsuter size $N = 30, 40, 50, 60, 70, 80$. For all-to-some communication, we experiment with $N = 100$, and select 10%, 20%, 30%, 40%, 50%, 60% of these 100 nodes as receivers.

5.1.1 Synchronous all-to-all personalized communication

Figure 6 plots the relation between the average synchronous all-to-all personalized communication completion time and the number of total nodes in the system for all four algorithms. Figure 7 plots the variances of synchronous all-to-all personalized communication of four scheduling algorithms.

Observation From Figure 6, the completion time of Random is larger than the completion time of ORSPB. The completion times of RSPB and caterpillar order are almost the same and smaller than the completion time of ORSPB.

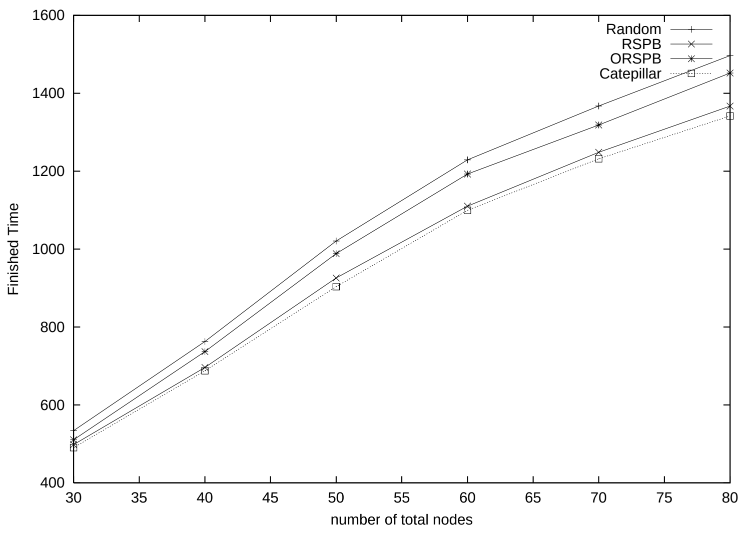

Figure 6: The average completion time of synchronous all-to-all personalized communication of four scheduling algorithms.

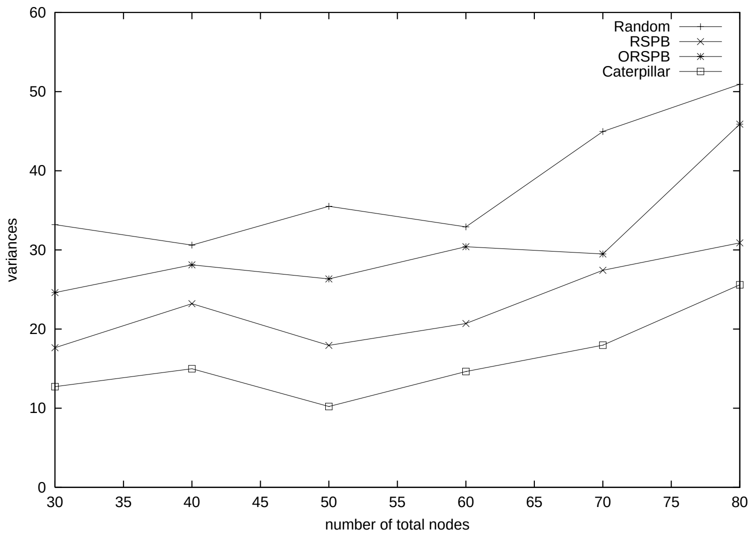

Figure 7: The variances of synchronous all-to-all personalized communication of four scheduling algorithms.

Communication Pattern	Algorithm	Model
all-to-all	1. Random order 2. Caterpillar order 3. Random_Starting_Point_Baseline 4. One_Random_Starting_Point_Baseline	Synchronous
all-to-all	1. Random order 2. Caterpillar order 3. Random_Starting_Point_Baseline 4. One_Random_Starting_Point_Baseline	Asynchronous
all-to-some	1. Random order 2. Caterpillar order 3. Random_Starting_Point_Baseline 4. One_Random_Starting_Point_Baseline	Synchronous
all-to-some	1. Random order 2. Caterpillar order 3. Random_Starting_Point_Baseline 4. One_Random_Starting_Point_Baseline	Asynchronous

Table 7: Four groups of the experiments

Explanation For an N node system, the sending order that each node follows has N destinations. If there are two or more source nodes that send messages to the same node in each step, there will be contentions. We hope that each node can send a message in each step so that no node is idle, therefore the completion time of the communication schedule may be reduced. However, the sending orders from Random have relatively high probabilities to have contention in each step[1]. If we think of the sending orders from Random as a set, most of the set are orders with contentions in some of the steps and very few elements of this set are without contention. Sending orders from Random will certaily result in contentions and lengthen the completion time. In general it is possible to reduce the completion time of the communication only when an sending order causes no contention in every step[2]

5.1.2 Asynchronous all-to-all personalized communication

Figure 8 plots the relation between the average asynchronous all-to-all personalized communication completion time and the number of total nodes in the system for all four algorithms. Figure 9 plots the variances of asynchronous all-to-all personalized communication of four scheduling algorithms.

Observation For a given N, we observe from Figure 8 that the completion time of each of the four scheduling algorithms is almost the same. When we try to increase N, the completion time of each scheduling algorithm increases proportionally.

[1]The probability of having at least one contention is $(1 - N!/N^N)$, where N is the number of processors

[2]Nevertheless, even if there is no contention in every step, it is still possible that contention may occur since the sending overhead and receiving overhead of each node is not the same.

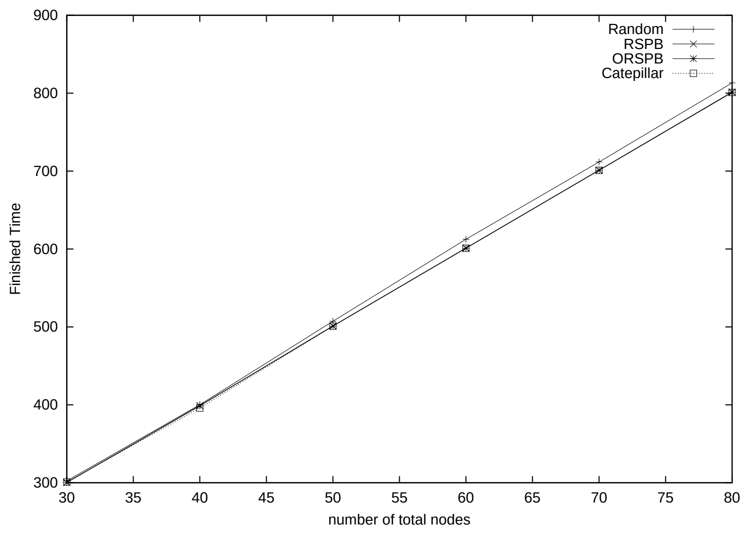

Figure 8: The average completion time of asynchronous all-to-all personalized communication of four scheduling algorithms.

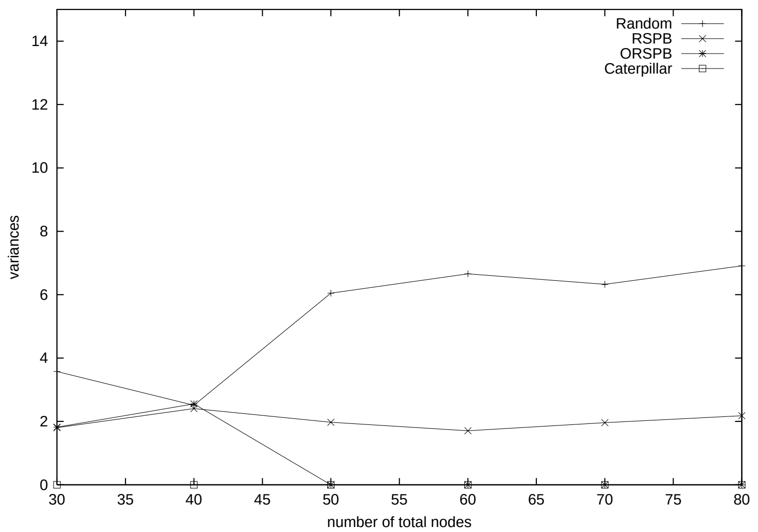

Figure 9: The variances of asynchronous all-to-all personalized communication of four scheduling algorithms.

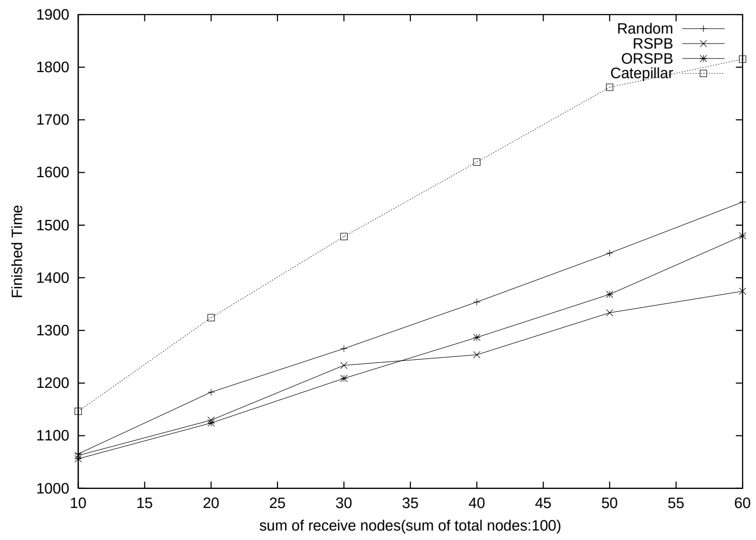

Figure 10: The average completion time of synchronous all-to-some communication of four scheduling algorithms.

Explanation In asynchronous model, each sender can send the next message to the next receiver immdeiately after the current sending overhead. As a result the efficiency is not that sensitive to whether there is contentions in the sending schedule, at least from the senders' point of view. If there are two or more messages waiting to be received at the same time (contention), it doesn't have effect on each receiver. For each receiver, it must receive these messages originally. Thus, when N is fixed, the completion times of the four scheduling algorithms are almost the same.

Observation When we increase N, the completion time of each scheduling algorithm increases proportionally.

Explanation There is an order with contentions in each step, it will not cause any delay. Each sender can still send the next message to the next receiver after the sending overhead. So, the only factor with an effect on the completion time of the communication schedule is N.

5.1.3 Synchronous all-to-some communication

Figure 10 plots the relation between the average synchronous all-to-some communication completion time and the number of total nodes in the system for all four algorithms. Figure 11 plots the variances of synchronous all-to-some communication of four scheduling algorithms.

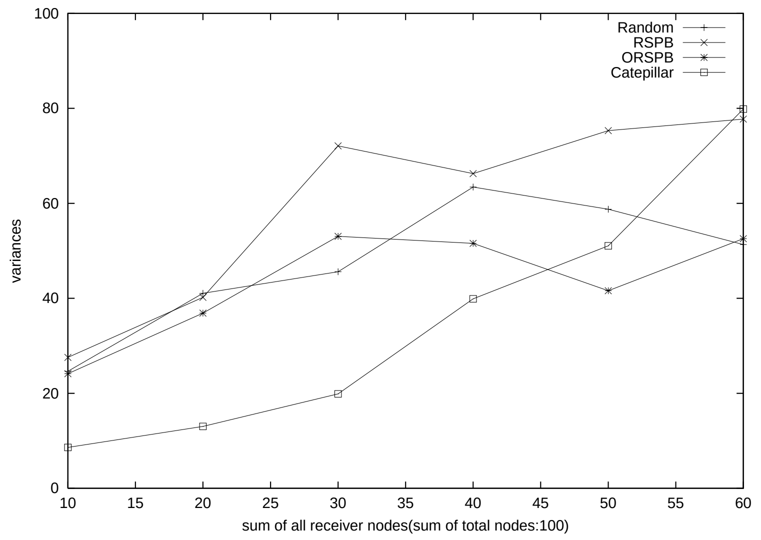

Figure 11: The variances of synchronous all-to-some communication of four scheduling algorithms.

Observation From Figure 10, we know that the completion time of the catepillar order is larger than the others and the completion times of RSPB and ORSPB are smaller than the others. On the other hand, the timing from RSPB and ORSPB are too close to determine which is better.

Explanation First we explain why the completion time of the catepillar order is larger than the others. Let's assume that we select 10% of 100 nodes as receivers. If there are 100 nodes in the system, only $P_{80} \sim P_{89}$ as receivers. If we use caterpillar order as the scheduling algorithm, in each step there will be 91 senders that want to send messages to the same receiver. That is, during the first step, $P_0 \sim P_{80}$ and $P_{90} \sim P_{99}$ all want to send messages to P_{80}. Duing the second step, $P_0 \sim P_{80}$ and $P_{90} \sim P_{99}$ want to send messages to P_{81}. That is, the caterpillar order makes it possible for many senders to send messages to the same receiver during each step. Because each receiver can receive most one message each time, caterpillar order causes serious delay.

Now we explain why the completion time of Random order is larger than RSPB and ORSPB. For all-to-some communication, the number of senders is larger than the number of receivers. There will be two or more nodes to send messages to the same node during each step. If we can distribute these senders among the receivers evenly, it is possible to effectively reduce the completion time of an all-to-some communication. For example, if there are 100 nodes in the system, among them only 10 are receivers. If only 10 out of 100 possible senders are sending messages to a receiver, it may be possible to reduce the completion time. Therefore the completion time of Random order is larger than RSPB and ORSPB since Random order does not distribute these senders among the receivers evernly. In contrast RSPB and ORSPB do distribute senders among the receivers more evenly.

5.1.4 Asynchronous all-to-some communication

Due to the minor difference between the timing results we use a table (Table 8) to illutrate the results. Figure 12 plots the variances of asynchronous all-to-some communication of four scheduling algorithms.

pertentage as recivers	Random order	RSPB	ORSPB	Caterpillar order
10%	992.27	992.21	992.76	994
20%	993.5	993.27	993.14	994
30%	994.84	994.83	993.96	996
40%	996.23	996.05	995.9	996
50%	997.93	997.36	996.41	1001
60%	999.2	998.91	997.78	1001

Table 8: The average completion time of asynchronous all-to-some communication of four scheduling algorithms.

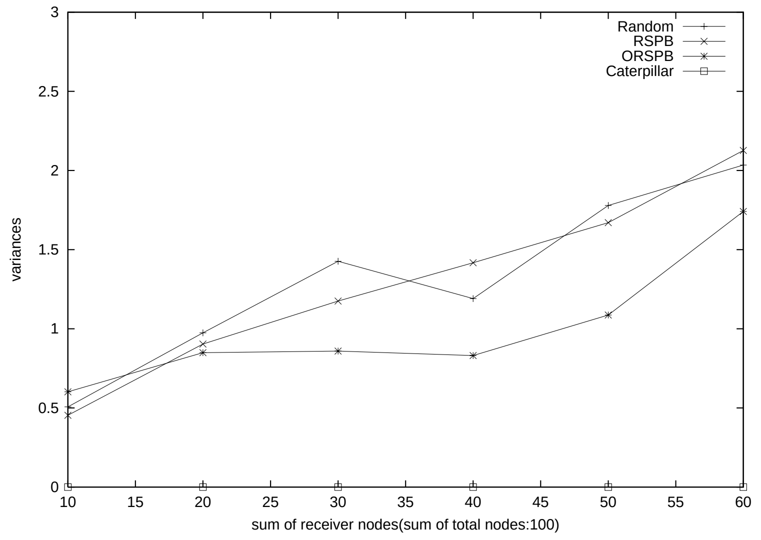

Figure 12: The variances of asynchronous all-to-some communication of four scheduling algorithms.

Observation From table 8 we know that when the number of receiver nodes is fixed, the completion times of four scheduling algorithms are similar. When we increase the number of receiver nodes, the completion time of each scheduling algorithm is almost unchanged.

Explanation We will explain when the number of receiver nodes is fixed, the completion times of four scheduling algorithms are similar. Let's assume that $N = 100$, and we select 10% of these 100 nodes as receivers. The sending overhead and receiving overhead of each node may be (1, 1) or (5, 5) or (10, 10). For each of these 100 senders, the message sent last must arrive at the last receiver before time = $(10 \times 10) + 1$, since there are 10 receivers, and the maximum value of the sending overhead is 10. (plus a network latency 1). Thus, before time = $(10 \times 10) + 1$, all messages that each sender sends to all receivers must have arrived. For each receiver, at time = $(10 \times 10) + 1 = 101$, all messages from each sender have arrived and are waiting to be received. For a receiver with receiving overhead 10, at time = 101, it could only have received at most 10 messages. However, there are 100 senders, therefore the orders produced by four scheduling algorithms are somewhat irrelavent. For all-to-some communciation in asynchronous model, at very early stage of the communication, all messages that each sender sends to all receivers have arrived and are waiting to be received. All the receivers are busy receiving messages. The orders the sender sends messages is not that important.

Next, when we increase the number of receiver nodes, is the completion time of each scheduling algorithm almost unchanged? Let's assume $N = 100$, and we select either 10% or 20% of the 100 nodes as receivers. The difference is that all messages that each sender sends to all receivers have arrived before time $= (10 \times 10) + 1$ or before time $= (20 \times 10) + 1$. For a receiver with receiving overhead 10, it can receive most 10 sender's messages at time = 101, or it could receive at most 20 messages at time = 201. Each receiver still needs to receive 100 sender's messages. Although we increase the number of receiver nodes, the number of the messages that each receiver needs to receive is still the same, and each receiver is still busy receiving messages at very early time in all-to-some and asynchronous model. Thus, the completion time of each scheduling algorithm is almost unchanged.

6 Conclusion

This paper discusses scheduling algorithms for heterogeneous networks of workstations (HNOW) systems, and designs simulators to conduct experiments on them. We want to schedule all-to-all personalized and all-to-some communication efficiently, and the final goal is to reduce the completion time of the communication schedule for heterogeneous networks of workstations in general.

From the experimental results we note that the completion time of all-to-all personalized communication or all-to-some in asynchronous model is not very sensitive to the sending order. The senders in asynchronous model is seldom delayed due to contention. When two senders send messages to the same receiver, each sender can send the next message after only a small period of sending overhead because it does not need to wait until the message is received. On the other hand, the completion time of all-to-all or all-to-some in synchronous model is sensitive to the order. When two senders want to send the messages to the same receiver, one of them must wait until the previous communication event finishes.

The current scope of this paper does not consider the asymmetric data transmission costs. That is, we assume that our cost matrix is symmetric, that is, the cost of sending one message from A to B is the same as from B to A. It is shown that the caterpillar order performs poorly in an asymmetric cost matrix, up to a factor of $O(P)$ from the optimal [2], where P is the number of processors. We did not encounter such phenomenon, due to our symmetric matrix assumption. The future work includes experimenting with asymmetric cost matrix, since this kind of communication pattern can be easily justified when there are servers in the system, and the data volume in a request and the corresponding reply is very different.

References

[1] T. Anderson, D. Culler, and D. Patterson. A case for networks of workstations (now). In *IEEE Micro*, pages 54–64, Feb 1995.

[2] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *Proceedings of the 7th IEEE Intnl. Symposium on High Performance Distributed Computing (HPDC)*, 1998.

[3] http://www.cacr.caltech.edu/publications/annreps/annrep95/compsci4.html.

[4] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt and Rinehart and Winston, 1976.

[5] D.B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3):617–632, June 1994.

[6] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. Manuscript, Dept. of EE-Systems, University of Southern California, May 1998.

[7] P. Brucker. *Scheduling Algorithms.* Springer, 1995.

[8] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.

[9] L.H. Jamielson, P.T. Mueller, and H.J. Siegel. FFT algorithms for SIMD parallel processing systems. *Journal of Parallel and Distributed Computing*, 3(1):48–71, March 1986.

[10] R. Libeskind-Hadas and J. Hartline. Efficient multicast in heterogeneous networks of workstations. In *Proceedings of 2000 International Workshop on Parallel Processing*, 2000.

[11] M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda, and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *Proceedings of International Workshop on Heterogeneous Computing*, 1999.