Reduction Optimization in Heterogeneous Cluster Environments*

Pangfeng Liu
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan, R.O.C.
pangfeng@cs.ccu.edu.tw

Da-Wei Wang
Institute of Information Science, Academia Sinica
Nankang, Taipei, R.O.C.
wdw@iis.sinica.edu.tw

Abstract

Network of workstation (NOW) is a cost-effective alternative to massively parallel supercomputers. As commercially available off-the-shelf processors become cheaper and faster, it is now possible to build a cluster that provides high computing power within a limited budget. However, a cluster may consist of different types of processors and this heterogeneity complicates the design of efficient collective communication protocols. For example, it is a very hard combinatorial problem to find an optimal reduction schedule for such heterogeneous clusters. Nevertheless, we show that a simple technique called slowest-node-first *(SNF) is very effective in designing efficient reduction protocols for heterogeneous clusters. First, we show that SNF is actually an approximation algorithm with competitive ratio two. In addition, we show that SNF does give the optimal reduction time when the cluster consists of two types of processors, and the ratio of communication speed between them is at least two.* Combined with a previous approximation algorithm for broadcast [17], we propose an all-reduction algorithm which sends the reduction answer to *all* processors, with competitive ratio 5. Finally we apply these theoretical results to branch-and-bound search and show that they can reduce the search time of the optimal reduction schedule by a factor of 500.

Keywords: Heterogeneous workstation cluster, scheduling optimization, reduction protocol, slowest-node-first heuristic, branch-and-bound search.

*The preliminary version of this paper appeared in the 14th International Parallel and Distributed Processing Symposium, 2000

1 Introduction

Network of workstation (NOW) is a cost-effective alternative to massively parallel supercomputers [2]. As commercially available off-the-shelf processors become cheaper and faster, it is now possible to build a PC or workstation cluster that provides high computing power within a limited budget. 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 cluster 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.

In parallel to the development of inexpensive and standardized hardware components for network of workstations, system softwares for programming on NOW are also advancing rapidly. For example, the *Message Passing Interface* (MPI) library has evolved into a standard for writing message-passing parallel code [1, 8, 12]. MPI programmers use a standardized high-level programming interface to exchange information among processes, instead of a native machine-specific communication library. MPI programmers can write highly portable parallel codes and run them on any parallel machine (including network of workstation) that has MPI implementation.

Most of the literature on cluster computing emphasizes *homogeneous* clusters – clusters consisting of the same type of processors. However, we argue that heterogeneity is one of the key issues that must be addressed in improving performance of NOW. First 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 communication performance from one another. Second, 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 will consist of "leftovers" from the previous installation, and "new comers" that are recently purchased. The issue of heterogeneity is both scientific and economic.

Any workstation cluster, be it homogeneous or heterogeneous, requires efficient collective communications [3]. For example, a barrier synchronization is often placed between two successive phases of computation to ensure that all processors finish the first phase before anyone goes to the next. In addition, a scatter operation distributes input data from the source to all the other processors for parallel processing, then a global reduction operation combines the partial solutions obtained from individual processors into the final answer. The efficiency of these collective communications will affect the overall performance, sometimes dramatically.

Heterogeneity of a cluster complicates the design of efficient collective communication protocols. 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 a homogeneous NOW every processor requires the same amount of time to transmit a message. For example it is straightforward to implement a reduction operation on P processors as $\log P$ phases, and in each phase we reduce the number of processors that have received the combined results from other processors by half. In a heterogeneous environment it is no long clear how we should proceed to complete the same task.

This paper shows that a simple heuristic called *slowest-node-first* (SNF) is very effective in designing reduction protocols for heterogeneous cluster systems. Despite the fact that SNF heuristic does not guarantee the optimal reduction time, we show that SNF is actually an approximation algorithm with competitive ratio two. In addition, we show that SNF does give the optimal reduction time when the cluster consists of two types of processors and the communication speed ratio between them is at least two.

It is a very hard optimization problem to find an optimal reduction schedule for heterogeneous clusters. In practice we often resort to dynamic programming or branch-and-bound search[3]. This paper shows that we can reduce the number of nodes we have to search by the theoretical results developed in the proof of SNF results. We conduct experiments to show that these techniques can improve the search efficiency by a factor of 500.

The rest of the paper is organized as follows. Section 2 describes the communication model in our treatment of reduction problem in HNOW. Section 3 describes the concept of earliest possible schedule and Section 4 describes the slowest-node-first heuristic for reduction in heterogeneous cluster. Section 5 states the theoretical results. Section 6 describes how the theoretical results help improve the heuristic search efficiency and Section 7 concludes.

2 Communication Model

There have been two classes of models for collective communication in homogeneous cluster environments. The first group of models assumes that all the processors are fully connected. As a result it takes the same amount of time for a processor to send a message to any other processor. For example, both the Postal model [5] and LogP model [14] use a set of parameters to capture the communication costs. In addition the Postal and LogP model assume that the sender can engage in other activities after a fixed startup cost, during which the sender injects the message into the network and is ready for the next message. Optimal broadcast scheduling for these homogeneous models can be found in [5, 14]. The second group of models assume that the processors are connected by an arbitrary network. It has been shown that even when every edge has a unit communication cost (denoted as the Telephone model), finding a optimal broadcast schedule remains NP-hard [9]. Efficient algorithms and network topologies for other similar problems related to broadcast, including multiple broadcast, gossiping and reduction, can be found in [7, 10, 11, 13, 16, 18, 19, 20].

Various models for heterogeneous environments have also been proposed in the literature. Bar-Noy et al. introduced a heterogenous postal model [4] in which the communication costs among links are not uniform. In addition, the sender may engage another communication before the current one is finished, just like homogeneous postal and LogP model. An approximation algorithm for multicast is given, with a competitive ratio $logk$ where k is the number of destination of the multicast [4]. Banikazemi et al. [3] proposed a simple model in which the heterogeneity among processors is characterized by the speed of sending processors. Based on this model, an approximation algorithm for broadcast with competitive ratio 2 is given in [17]. We adopt the simple model from [3] for its simplicity and the high level abstraction of network topology. Other models for heterogeneous clusters include [6, 15].

The communication model of a heterogeneous environment is defined as follows. The system consists of n processors $\{p_0, ..., p_{n-1}\}$, each is capable of direct point-to-point communication to one another. A processor p_i is characterized by its transmission time $t(p_i)$, i.e. the time it takes for p_i to send a message to any other processor.

In order to make the communication model realistic, we further assume that two communications cannot overlap, i.e. neither the sender nor the receiver of an ongoing communication can engage in any other communication at the same time. This assumption is based on that in practice most of the processors in a clusters only have limited resource for communication.

We define the reduction problem with the communication model we just described. Suppose each processor in the system has a unit of information, and we would like to combine all these informations into the final answer, how do we schedule the processors so that the reduction takes the least amount of time? For example, each processor may have a number and we want to compute the total sum of these numbers by message passing. For a homogeneous system a simple tree algorithm can be used so that during each iteration half of the processors send their information to the other half where the informations are combined. The algorithm takes $\log n$ rounds to combine all the informations since the number of active processors reduces by half per iteration. However, due to the variance in communication speed in a heterogeneous environment, this algorithm cannot guarantee minimum reduction time.

We formally define the reduction as a scheduling problem. We observe that during the reduction process exactly $n-1$ messages are sent by $n-1$ different processors – each processor except the final reduction destination, sends out exactly one message. Based on this observation, we define a reduction schedule as a mapping function from a processor to the time that it starts sending its message. Formally we define the scheduling function s so that for any processor p (except the reduction destination), p starts and completes sending its message at time $s(p)$ and $s(p)+t(p)$ respectively. The interval $[s(p), c(s,p)]$ is defined as the *transmission window* of p, where $c(s,p) = s(p) + t(p)$ is the *completion time* of p under the schedule s.

To show that it is not beneficial for the reduction destination to send out any message, we define that a processor p can *reach* the redcution destination d if and only if there exists a series of transmission windows $w_1, w_2, ..., w_k$, and the following properties are true.

1. The source of w_1 is p.
2. The destination of w_k is d.
3. The destination of w_i is the source of w_{i+1}, for $1 \le i < k$.

And we define that a schedule is "correct" if and only if all the processors (except the destination d) can reach d. It is easy to see that if a schedule is correct, then we can remove all the transmission windows with d as the source, and all the other processors can still reach d. Therefore, without lose of generality, we assume that the destination *will not* send any messages in our scheduling.

Due to the constraint that a processor can only participate a single communication at any given time, we must distinguish valid schedule from invalid ones. To formalize this constraint, we define two sets of processors for any schedule at any given time. Let $A(s,t)$ be the number of processors that are still actively sending their messages, and $C(s,t)$ be the number of processors that have completed sending their messages at time t under schedule s.

$$\begin{aligned} A(s,t) &= |\{p_i : s(p_i) \leq t < c(s,p_i)\}| \\ C(s,t) &= |\{p_i : t \geq c(s,p_i)\}| \\ 2A(s,t) + C(s,t) &\leq n \end{aligned} \tag{1}$$

A schedule function s is *valid* if and only if inequality 1 is true. The inequality says that at any time t, the total number of senders and receivers, and those processors that have sent out their messages, should not be more than the number of processors in the system. Figure 1 show a valid schedule and the corresponding A and C functions.

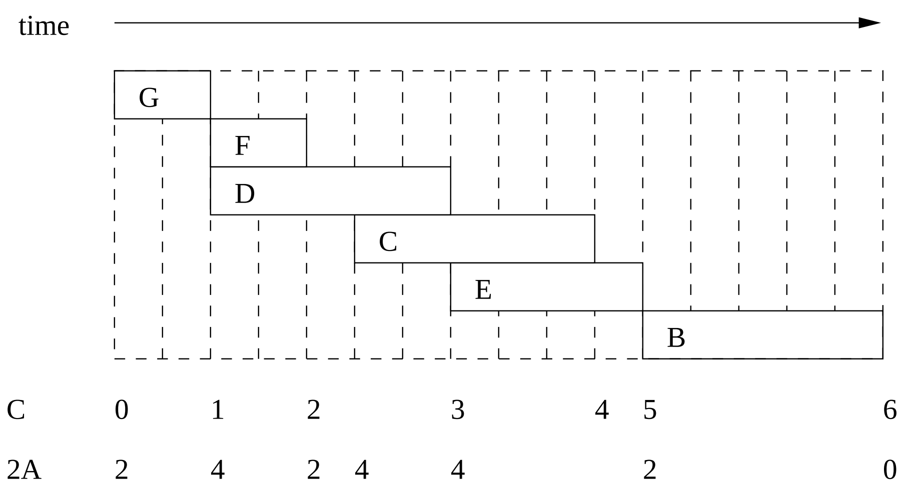

Figure 1: An valid schedule example showing the A and C functions. The system consist of seven processors – indicated by the uppercase letters from A to G. The transmission time for these seven processors are 10, 5, 5, 5, 4, 2, 2, respectively.

3 Earliest-Possible Schedule

This section describe a technique called *earliest possible* scheduling that can "normalize" all the possible valid schedules. We use this canonical form to simplify the discussion of finding an optimal schedule.

An earliest possible (EP) schedule is one in which all the communications are initiated as earlier as possible. A new communication can be initiated as soon as the number of *free processors* reaches 2 – one for the sender and one for the receiver. Let $F(s,t)$ denote the number of processors that we are free to use to initiate a new communication at time t under schedule s. That is, $F(s,t) = n - CA(s,t) - C(s,t)$, where $CA(s,t) = |\{P_i | s(P_i) < t < s(P_i) + t(P_i)\}|$ is the number of processor that are in the middle of sending and receiving messages[1]. As a result all the first $\lfloor n/2 \rfloor$ processors will start sending messages at time 0, and the rest of the processors will start sending messages as soon as the number of free processors reaches two, as illustrated by Figure 2. The algorithm EP will assign non-decreasing start time to the processors in the order they appear in the input processor sequence P.

```
Algorithm EP(P)
{
  i = 1; time = 0; free = n;
  Active = empty set; Complete = empty set;
  while (i ≤ n-1) {
    do while (free ≥ 2) {
      set the start time of the ith processor in P
      to time;
      add the ith processor in P into Active;
      i = i + 1;
      free = free - 2;
    }
    find the set of processors p that has the smallest
    completion time in Active;
    time = the completion time of p;
    move p from Active to Complete;
    free = free + the number of elements in p ;
  }
}
```

Figure 2: The earliest possible scheduling algorithm.

By definition an earliest possible schedule s has the following properties.

1. $n - 1 \leq 2A(s,t) + C(s,t) \leq n$.
2. For any processor p that $s(p) > 0$, there exist another processor q such that $s(p) = c(s,q)$.

Note that it is trivial to convert any valid schedule into an EP schedule without increasing the total time – we just move the transmission window of each processor to the earliest possible time. As a result EP schedule servers as a canonical form in which we will limit our search of an optimal schedule. Since the EP algorithm can completely determine a schedule once the order of processors in the sequence P is

[1] $CA(s,t)$ is different from $A(s,t)$, which includes processors that will start sending and receiving message at time t.

fixed, we only have to consider up to $(n-1)!$ different schedules. Figure 3 shows the EP schedule that is converted from the schedule in Figure 1.

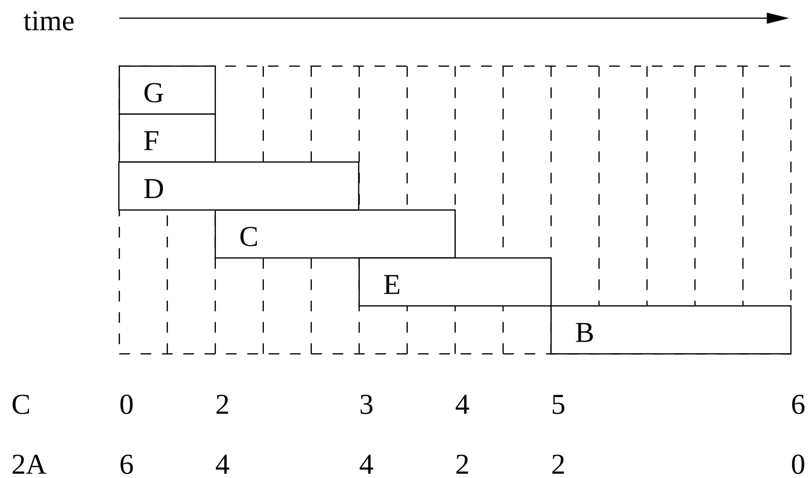

Figure 3: The EP schedule converted from the schedule in Figure 1

3.1 Sender Dependency and Destination Assignment

Given a EP schedule, we need to specify the destination for all messages in order to program a cluster to perform a reduction. We will show that under any EP schedule, it is always possible to do so without violating the constraint that a processor cannot participate more than one communication simultaneously.

Lemma 1 *For any earliest possible schedule we can assign the destinations for all processors so that no two overlapping transmission windows have any sender or receiver in common.*

Proof. First we would like to establish the dependency among the processors. Note that in algorithm EP it requires two free processors to start a new transmission. As a result we define the *predecessors* of a sender processor to be the two senders that must complete before the new transmission can start[2]. This dependency forms a binary tree among all sender processors.

Now we can start filling in the destinations for all senders. First we assign the reduction destination d as the destination of the sender that has the latest completion time. Then for a sender/receiver pair s and r, we assign s and r to be the destination of the two predecessors of s. Since both predecessors complete sending their messages before s starts, no overlapping transmission windows will have processors in common. ■

Figure 4 shows an example of how to find the destination for the schedule in Figure 3. Note that the assignment is not unique since we can assign s and r to either of the two predecessors.

[2]Note that when n is an odd number, one of the processor will have only one predecessor.

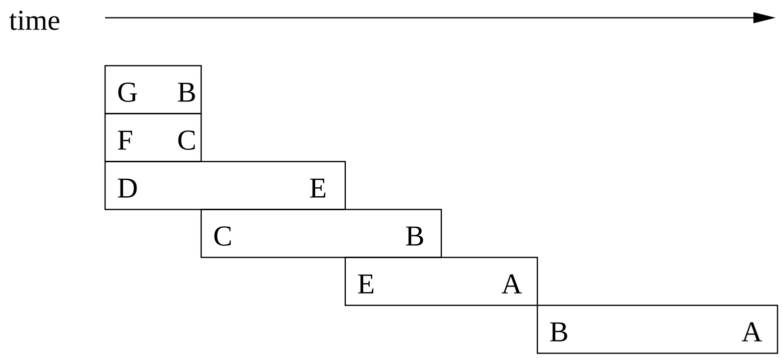

Figure 4: Find the destinations for all senders in Figure 3. Note that A is the destination processor.

4 Slowest-Node-First Scheduling

We introduce a simple scheduling method called *slowest-node-first* (SNF) for the reduction problem. SNF simply sorts the processors in P in non-increasing order, and gives the sorted sequence $P = (p_1, p_2, ...p_{n-1})$[3] to algorithm EP. In the next section we show that this simple technique is very effective in obtaining a good reduction schedule. Figure 5 shows the slowest-node-first schedule with the same cluster as in Figure 3.

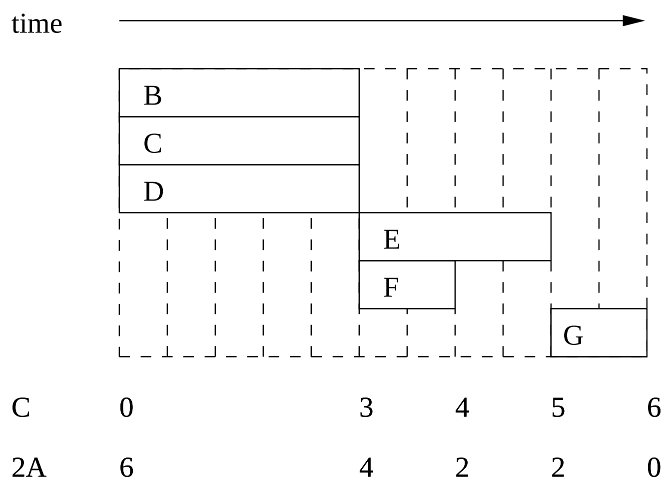

Figure 5: Slowest-node-first scheduling result from the same cluster as in Figure 1 and 3.

The rationale of having the slowest processors to send messages first is as follows. At the beginning of the reduction process, we would like to overlap as many communication windows as possible. Intuitively we let all the slow processors send first so that they will overlap with each other, instead of having to wait for each other at the end of the reduction.

[3] $t(p_i) \geq t(p_j)$ if $i < j$

5 Main Results

Before we describe the main results, we describe an exchange lemma that clarifies our intuition that slow processors should send first, as we did in SNF scheduling.

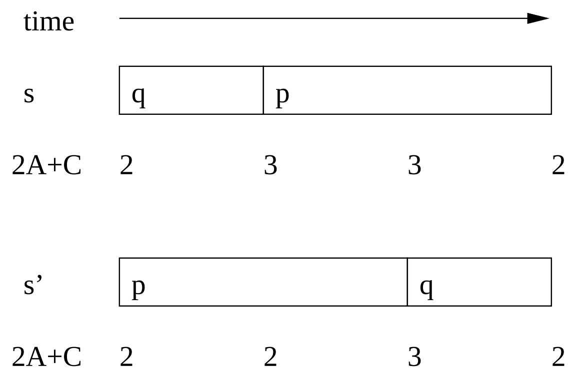

Figure 6: An illustration on the contribution to the sum of C and $2A$.

Lemma 2 *Let s be a valid schedule and processor p starts right after processor q ends, i.e. $s(p) = c(s, p)$. If $t(p) > t(q)$ then we can exchange p and q so that the modified schedule s' in which $s'(p) = s(q)$ and $s'(q) = c(s', p)$ is also valid.*

Proof. From Figure 6 we observe that the contribution of p and q to the sum of C and A functions is always higher in s than in s', therefore if s can satisfy Inequality 1, so can s'. ■

From Lemma 2 it follows that in the search of an optimal reduction schedule we can neglect those schedules that have a slower sender p with a faster predecessor q. There are two cases to consider. First if $s(p) = c(s, q)$ then by Lemma 2 we can switch p and q. On the other hand, if there is a gap between the transmission windows of p and q, then we can delay the window of q until it touches p's window. We can do so because there is no new transmission initiated between $s(p)$ and $s(q) + t(q)$. As a result we will consider only those schedules that all the senders have predecessors of smaller or equal transmission time.

Corollary 1 *There exists an optimal schedule such that the predecessors of every senders have an equal or higher transmission time than the sender.*

5.1 Competitive Ratio

We now consider a special class of clusters in which the transmission time of every processor is a power of 2. Without lose of generality we assume that the fastest processor has a transmission time of 1, and the slowest one has 2^k. We will call this kind of cluster a *power 2* cluster and show that SNF gives an optimal scheduling for all power 2 clusters.

Theorem 1 *The slowest-node-first method gives an optimal schedule for all power 2 clusters.*

Proof. We show that every schedule for a power 2 cluster can be converted into the SNF schedule without increasing the total reduction time. We reschedule all the slowest processors as early as possible, until there is no faster processor ahead of them. Then we reshcdule the second slowest processors the same way, and repeat this rescheduling until the final schedule is the same as SNF.

When the transmission time of every processors is a power of 2, we claim that the start time of every processor is a multiple of its transmission time. Since every processor has a predecessor that ends when it starts, we locate a "chain" of predecessors all the way back to time 0. In addition, every one of these predecessors has a transmission time of equal or larger power of 2. Therefore the start time, which is the sum of the transmission time of these predecessors, is also a multiple of its transmission time.

Consider a slowest processor p, and the set F of faster processors that starts before p. Let q be the processor completes last in F. If q finishes right when p starts, then by Lemma 2 we can exchange p and q. If there is a gap between $s(p)$ and $s(q) + t(q)$, then any processors starts within this gap must end within this gap as well since all the processor must start and end at the multiple of its transmission time. Therefore q will not be the processor with the largest completion time in F. Similarly we argue that any processor completes in that gap must also start in that gap, therefore we can safely delay the transmission window of q so that q ends when p starts. By Lemma 2 we can then exchange p and q and the theorem follows. ■

Theorem 2 *The slowest-node-first schedule has a total reduction time no greater than twice of the time of an optimal schedule.*

Proof. Let s be an optimal schedule for a cluster P. Without lose of generality we assume that the fastest processor in P has a transmission time 1. We first convert P into a power 2 cluster by increasing the transmission time of every processor p to $2^{\lceil \log t(p) \rceil}$. We call this new cluster P'.

We argue that there exists a schedule s' for P' in which every processor p starts at time no later than $2s(p)$. The claim follows from a simple induction that if every processor p in P' waits for the two predecessors q and r defined by s, then it can start at time no later than $2s(p)$, since both p and q starts no later than $2s(q)$ and $2s(q)$, and their transmission time at most double in P'.

Now we have constructed a new schedule s' for P' that has a reduction time at most twice of s. Since s' is a schedule for a power 2 cluster, its reduction time will be no less than the optimal schedule s^* produced by SNF for P'. Finally the SNF schedule s'' on P is at least as fast as the SNF schedule s^* for P', since every processor starts earlier in P than in P' because both of its two predecessors can start earlier in P, and sends message faster. Therefore the reduction time of s'' is no more than s^*, which in turn is no more than twice of s. ■

5.2 Competitive Ratio for All-reduction

There is a similar problem called *all-reduction*, in which the final reduction answer should be made known to all processors. This problem can be naively solved by performing a reduction then followed by a broadcast. It is easy to see that the an all-reduction is at least as expensive as the fastest broadcast from any source processor, or a fastest reduction into any destination processor.

In a previous paper [17] we showed that starting from a fastest processor as the source, we can perform broadcast in no more than twice of the fastest broadcast, with an technique called *fastest-node-first* [3]. In addition, we have already shown that with *slowest-node-first* scheduling, we can perform reduction to a slowest destination with time no more than twice the time of the any fastest reduction. Combining these two algorithms, we can perform an all-reduction within *5* times of the optimal time, with an extra factor of one by transferring the result from the slowest processor to the fastest one between the two algorithms.

Theorem 3 *There exists an algorithm that performs all-reduction with competitive ratio 5.*

5.3 Two Types of Processors case

In this section we prove that SNF does give the optimal reduction time when the cluster consists of two types of processors, and the communication speed ratio between them is at least two. We also find a counterexample which shows that if the communication speed ratio between them is less than two then SNF is not optimal.

Lemma 3 *Let s be a valid schedule, p be a slow processor and q_1, q_2, q_3 be fast processors. If $s(p)$ is within the transmission window of q_1, q_1 stars right after q_2, and q_2 starts right after q_3, then there exists a schedule s' with following properties.*

1. *$s'(p) = s(q_3)$, i.e. process p starts sending message in s' the time process q_3 starts sending message in s.*
2. *$s'(q_3) = s(p)$, i.e. process q_3 starts sending messages in s' the time process p starts sending message in s.*
3. *Process q_1 starts right after q_2 and q_1 ends in s' at the same time as p ends in s.*

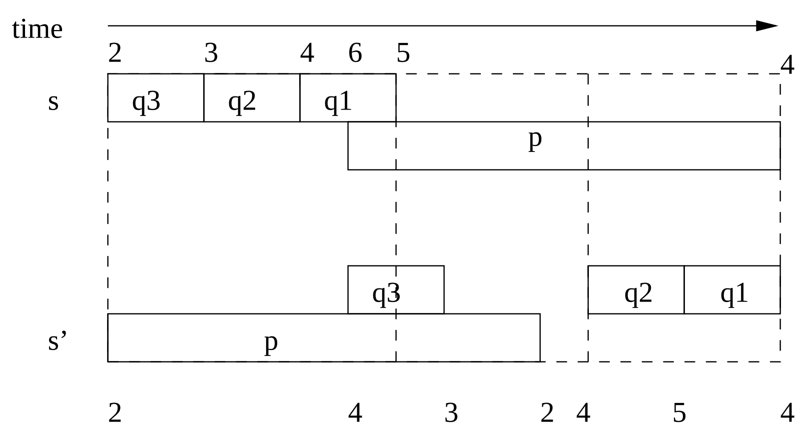

Figure 7: Rearrange the communication windows of p and q_1, q_2, q_3.

Proof. From Figure 7 we observe the followings:

1. $2A(s, s(p)) + C(s, s(p)) = 6$ because p is at least twice as slow as q_1.
2. $s'(q_2) < c(s', p)$, i.e. p ends before q_2 starts in s'.

From the above observation we conclude that at any time t the value of $2A + C$ is always equal or higher in s than in s', and the theorem follows. ■

According to Lemma 2 we know that there exists an optimal schedule in which no slow processor will start *right after* a fast processor. Therefore, we can assume for every slow processor, its predecessors are slow processors as well. As a a result the starting time of every slow processor is a multiple of its transmission time, and we can layer the slow processors according to their starting time as follows.

Definition 1 *Layer one processors are the set of slow processors starting at time 0. Layer $i+1$ processors are the set of slow processors starting right after layer i processors.*

Theorem 4 *SNF is optimal when there are only two types of processors and the communication speed ratio between them is at least two.*

Proof. Assume that there exists an optimal solution in which there exits a slow processor that starts later than a fast processor. Let p be the one in the earliest layer among such slow processors, and q be the one that has the latest completion time among all those fast processors before p. We show that it is always possible to reschedule p so that it starts earlier than q without increasing the total time.

There are three cases to consider – the processor p can start *before*, *at*, or *after* the processor q ends. For the second case we apply Lemma 2 and switch p and q. For the third case, since q finishes last among all the fast processors starting before p, there will be no processor starting after q ends and before p starts. As a result we can delay the transmission window of q so that q ends right when p starts, then we apply Lemma 2 to move p earlier.

We now consider the first case. Since every processor has a predecessor that ends when it starts, we locate a "chain" of predecessors for q all the way back to time 0. For the processor q we need at least two levels of predecessors and both of them must be fast ones, since the ratio of communication speed is at least two and the slow processors must start at a multiple of its transmission time. By Lemma 3 we find a new schedule when p starts before q. We repeatedly reschedule the slow processors p earlier until no fast processor before it, and the theorem follows. ■

Figure 8 shows a counterexample that SNF always gives optimal reduction time. This cluster has 4 slow processors with transmission time x, and 8 fast processors with transmission time 1. The alternative schedule requires $2x + 1$ time when $1.5 \leq x < 2$, or 4 when $1 < x < 1.5$. In contrast SNF requires $x + 3$ time for both cases, and has a longer reduction time for for all x between 1 and 2. As a result the bound of two in Theorem 4 is tight.

6 Heuristic Search

The previous section describes the theoretical results that guarantees the optimality of SNF method for special cases, and provides performance guarantee for general

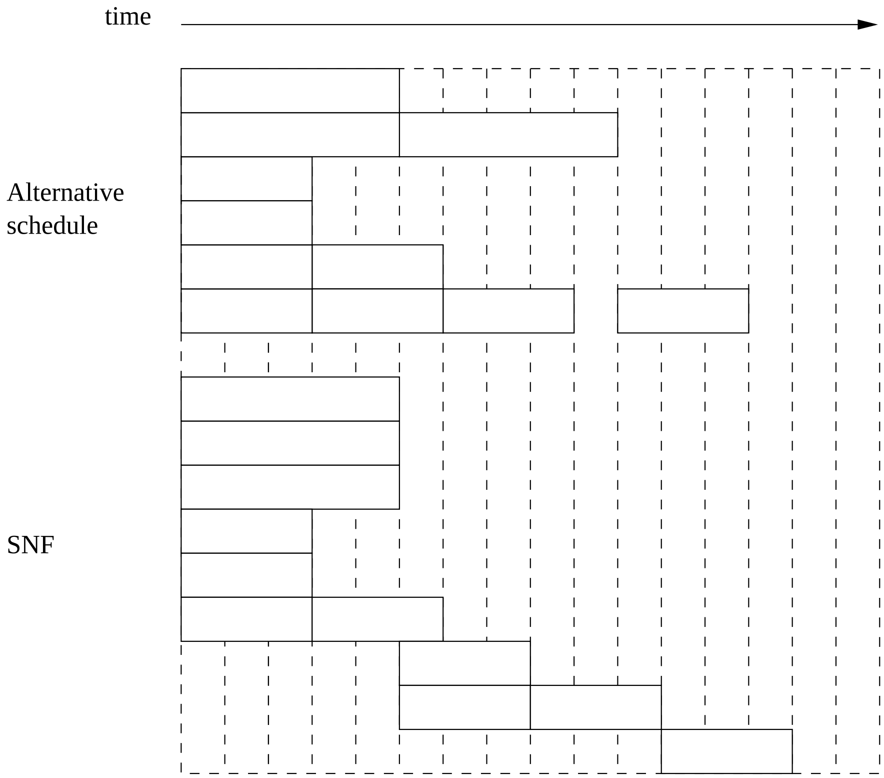

Figure 8: A counterexample that SNF always gives optimal reduction time in a cluster consisting of two types of processors.

cases. However, in practice one may wish to find the optimal reduction schedule for a particular cluster that contains more than two kinds of processors. In such cases we have to search for the optimal schedule since SNF does not guarantee optimality. This section describes the techniques that we used to speed up the search process.

As described in Section 3, every reduction schedule can be converted into a earliest-possible schedule, which can be represented by a sequence of processors. As a result we can find an optimal reduction schedule among these $(n-1)!$ possible sequences, where n is the number of processors in the cluster. However, for a typical cluster $(n-1)!$ is such a large number that we apparently cannot try all these permutations, even by a branch-and-bound procedure. To overcome this problem, we conduct experiments to show that by Corollary 1 we can dramatically reduce the search space.

We use three techniques to reduce the number of sequences we have to consider. First of all, we examine the sequences in such an order that those sequences with slower processors appearing first will be examined first. Formally we define the *priority* of a sequence to be the number processors that have longer or equal transmission time than the next processor in the sequence. In other words, the SNF schedule has the highest priority and will be considered first.

The second technique is to apply Corollary 1 so that when a fast processor is scheduled to send the message to a slower processor, we can prune that subtree immediately. In addition, it is possible for several senders to send messages simultaneously so that more than one processor can receive at the same time. In that case if any sender is faster than any of those possible receivers then we can drop this partial solution completely.

Finally, we use the standard branch-and-bound technique to explore the search tree. If the cost of a partially examined sequence is already larger than the current optimal, the entire subtree is pruned. This technique is most effective when the difference among processor speed is large.

We conduct the experiments on a Pentinum 3-450 PC running FreeBSD 3.2 UNIX. The PC has 128Mbytes memory and we use gcc 2.7.2-1 to compile the code. The input cluster configurations for our experiments are generated as follow. We assume that the number of classes in a cluster is 3. This assumption is practical since processors are usually purchased in batches and the number of batches is usually small. We vary the cluster size from 10 to 21. For each processor we randomly assign a communication speed from the three possible values. For each cluster size we repeat the experiments for 50 times and compute the average for the quantities we measured.

We quantify the *search ratio* of an algorithm as the percentage of the entire search tree the algorithm has to examine in order to find the optimal solution. As a result, an algorithm that scans n tree nodes before finding the optimal one the search ratio is $\frac{n}{N}$, where N is the total number of nodes in the entire search tree.

Table 1 compares the efficiency of our algorithm with a simple branch-and-bound search. Guided by various heuristics described earlier, our algorithm searches much fewer tree nodes than the generic branch-and-bound method, and consequently runs much faster. For large clusters consisting up to 21 nodes our algorithm runs about five hundred times faster than the generic algorithm, and can find the optimal solution within 15 second, while the generic algorithm runs more than two hours.

The first two columns in Table 1 indicate the average number of nodes and leaves of the search trees generated. The next three columns are the number of tree nodes examined, the search time (in second), and the search ratio from our algorithm. The next three columns are from a generic branch-and-bound algorithm. The last column shows the performance ratio between these two algorithms.

7 Conclusion

This paper shows that the slowest-node-first scheduling is a very efficient reduction protocols for heterogeneous cluster systems. We show that SNF is an approximation algorithm with competitive ratio two. Combined with a previous approximation algorithm for broadcast [17], we propose an all-reduction algorithm which sends the reduction answer to *all* processors, with competitive ratio 5. In addition, we show that SNF does give the optimal reduction time when the cluster consists of two types of processors, and communication speed ratio between them is at least two. Finally we apply these theoretical results to branch-and-bound search and show that they can reduce the search time by a factor of 500.

It will be interesting to extend this technique to other communication protocols and models. For example, in our model the communication time is determined solely by the sender. In a more practical and complex model the communication

P	search tree		SNF search			generic branch-and-bound			search time
	N	L	n	time	n/N	n	time	n/N	ratio
10	7852	2406	475	0.002	6.049%	5642	0.05	71.854%	25
11	21544	6586	824	0.005	3.825%	15330	0.17	71.157%	34
12	55243	16785	2151	0.01	3.894%	39367	0.48	71.262%	48
13	163186	49695	3715	0.03	2.277%	115419	1.64	70.728%	54
14	440848	134154	9908	0.07	2.247%	310731	4.76	70.485%	68
15	1141417	344573	14845	0.12	1.301%	804970	13.61	70.524%	113
16	3824647	1160027	46688	0.41	1.221%	2683898	48.48	70.174%	118
17	10379374	3147823	70259	0.66	0.677%	7248825	144.32	69.839%	218
18	29444739	8902547	187034	1.85	0.635%	20576306	444.51	69.881%	240
19	78392528	23692980	293681	3.28	0.375%	54742994	1115.72	69.832%	340
20	189185942	56704807	848633	10.43	0.449%	132562430	2476.89	70.069%	237
21	600236924	180412877	1187418	15.28	0.198%	420191444	8173.73	70.004%	544

Table 1: The comparison of two search programs.

time may be a function of both the send and the receiver [6]. In addition, it will be worthwhile to investigate the possibility to extend the analysis to similar protocols like parallel prefix, or all-to-all broadcast. These questions are very fundamental in designing collective communication protocols in heterogeneous clusters, and will certainly be the focus of further investigations in this area.

8 Acknowledgment

The authors thank Mr. Tzu-Hao Sheng for implementing the heuristic search programs. This work is supported in part by National Science Council of Taiwan, under grant number NSC 89-2213-E-194-009.

References

[1] *Message Passing Interface Forum.* Mar 1994.

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

[3] M. Banikazemi, V. Moorthy, and D.K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of International Parallel Processing Conference*, 1998.

[4] A. Bar-Noy, S. Guha, J. Naor, and Schieber B. Multicast in heterogeneous networks. In *Proceedings of the 13th Annual ACM Symposium on theory of computing*, 1998.

[5] A. Bar-Noy and S. Kipnis. Designing broadcast algorithms in the postal model for message-passing systems. *Mathematical Systems Theory*, 27(5), 1994.

[6] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *Proceedings of the International Conference on Distributed Computing Systems*, 1999.

[7] M. Dinneen, M. Fellows, and V. Faber. Algebraic construction of efficient networks. *Applied Algebra, Algebraic Algorithms, and Error Correcting codes*, 9(LNCS 539), 1991.

[8] J. Bruck et al. Efficient message passing interface(mpi) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, Jan 1997.

[9] M. R. Garey and D. S. Johnson. *Computer and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman, 1979.

[10] L. Gargang and U. Vaccaro. On the construction of minimal broadcast networks. *Network*, 19, 1989.

[11] M. Grigni and D. Peleg. Tight bounds on minimum broadcast networks. *SIAM J. Discrete Math.*, 4, 1991.

[12] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi: a message passing interface standard. Technical report, Argonne National Laboratory and Mississippi State University.

[13] S. M. Hedetniemi, S. T. Hedetniem, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks.*, 18, 1991.

[14] R. Karp, A. Sahay, E. Santos, and K. E. Schauser. Optimal broadcast and summation in the logp model. In *Proceedings of 5th Ann. Symposium on Parallel Algorithms and Architectures*, 1993.

[15] R. Kesavan, K. Bondalapati, and D. Panda. Multicast on irregular switch-based networks with wormhole routing. In *Proceedings of International Symposium on high performance computer architecture*, 1997.

[16] A. L. Liestman and J. G. Peters. Broadcast networks of bounded degree. *SIAM J. Discrete Math.*, 1, 1988.

[17] P. Liu and T. Shen. Broadcast scheduling optimization for heterogeneous cluster systems. In *Proceedings of the 12th Annual ACM Symposium on Parallel Architectures and Algorithms*, 2000.

[18] D. Richards and A. L. Liestman. Generalization of broadcast and gossiping. *Networks*, 18, 1988.

[19] J.A. Ventura and X. Weng. A new method for constructing minimal broadcast networks. *Networks*, 23, 1993.

[20] D. B. West. A class of solutions to the gossip problem. *Discrete Math.*, 39, 1992.