Journal of Algorithms **42,** 135–152 (2002)
doi:10.1006/jagm.2001.1204, available online at http://www.idealibrary.com on IDEAL®

Broadcast Scheduling Optimization for Heterogeneous Cluster Systems

Pangfeng Liu

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.
E-mail: pangfeng@csie.ntu.edu.tw

Received October 23, 2000

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 PC or workstation cluster that provides high computing power within a limited budget. However, a cluster may consist of different types of processors, and this heterogeneity within a cluster complicates the design of efficient collective communication protocols. This paper shows that a simple heuristic called *fastest-node-first* (FNF) (1998, M. Banikazemi, V. Moorthy, and D. K. Panda, *in* "Proceedings of the International Parallel Processing Conference") is very effective in reducing the broadcast time for heterogeneous cluster systems. Despite the fact that the FNF heuristic fails to give the optimal broadcast time for a general heterogeneous network of workstations, we prove that FNF always gives the optimal broadcast time in several special cases of clusters. Based on these special case results, we show that FNF is an approximation algorithm that guarantees a competitive ratio of 2. From these theoretical results we also derive techniques to speed up the branch-and-bound search for the optimal broadcast schedule in HNOW. © 2002 Elsevier Science

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 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 Megabits per second of bandwidth with inexpensive Fast Ethernet adaptors and hubs.

In parallel with the development of inexpensive and standardized hardware components for NOW, system software for programming on NOW is also advancing rapidly. For example, the *Message Passing Interface* (MPI) library has evolved into a standard for writing message-passing parallel codes [1, 8, 12]. An MPI programmer uses a standardized high-level programming interface to exchange information among processes, instead of native machine-specific communication libraries. An MPI programmer can write highly portable parallel codes and run them on any parallel machine (including a network of workstations) 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 the parallel 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 workstations* (HNOW) becomes complicated since these processors will differ in performance in computation and communication. 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 consists of "leftovers" from the previous installation, and "newcomers" that are recently purchased. The issue of heterogeneity is both scientific and economic.

Every workstation cluster, be it homogeneous or heterogeneous, requires efficient collective communication [3]. For example, a barrier synchronization is often placed between two successive phases of computation to make sure 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 of 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 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.

This paper shows that a simple heuristic called *fastest-node-first* (FNF), introduced by Banikazemi et al. [3], is very effective in designing broadcast protocols for heterogeneous cluster systems. Despite the fact that the FNF heuristic does *not* guarantee optimal broadcast time for every heterogeneous network of workstation, we show that FNF does give the optimal broadcast time for several special cases of HNOW. First we show that there exists an optimal broadcast schedule in which all the fastest processors receive the broadcast messages before all of the others (called the fastest-node-first principle). Consequently, when there are only two classes of processors, FNF always gives the optimal broadcast time. This result is very useful in practice since most clusters consist of a small number of classes of processors.

In addition, we show that when the communication time of any processor p in the cluster is a multiple of any faster processor q, then p should be scheduled *after* q. Consequently, FNF gives the optimal broadcast time in such clusters. This result by itself is not very practical since most clusters do not have such a property. However, based on this result, we show that FNF is actually an approximation algorithm that guarantees a broadcast time within *twice* the optimum for *any* cluster.

Besides the theoretical results, we also introduce new search techniques derived from the theoretical results. For example, we can use the fastest-node-first principle, combined with a monotonic cost property in [3], to dramatically reduce the search space. These techniques are useful in practice when one has a cluster consisting of more than two types of processors, and the FNF performance guarantee described above is consider insufficient.

The rest of the paper is organized as follows. Section 2 describes the communication model in our treatment of broadcast problems in HNOW. Section 3 describes the fastest-node-first heuristic for broadcast in HNOW. Section 4 gives the theoretical results. Section 5 discusses techniques in the heuristic search for the optimal broadcast schedule, and Section 6 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 of 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 the LogP model [14] use a set of parameters to capture the communication costs. In addition the Postal and LogP models 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 assumes that the processors are connected by an arbitrary network. It has been shown that even when every edge has a unit communication cost (denoted the Telephone model), finding an 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, 18–20].

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 as in homogeneous postal and LogP model. An approximation algorithm for multicast is given, with a competitive ratio $\log k$, where k is the number of destinations 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 reduction with competitive ratio 2 is given in [17]. We adopt the simple model from [3] for its simplicity and the high level of abstraction of network topology. Other models for heterogeneous clusters include [6, 15].

The model is defined as follows. A heterogeneous cluster is defined as a collection of processors $p_0, p_1, \ldots, p_{n-1}$, in which each processor is capable of point-to-point communication with any other processor in the cluster. Since we are interested in the communication capability only, each processor is characterized by its speed of sending messages. Formally, we define a non-negative *transmission time* of a processor to be the time it needs to send a unit of message to any other processor. Note that by this definition the time required to transmit a message is determined by the sender.

The communication model requires that the sender and receiver processors cannot engage in multiple message transmissions simultaneously. That is, a sender processor must complete its data transmission to a receiver before sending the next message to anyone else. This restriction is due to the fact that every processor and communication network have limited bandwidths; therefore we would like to exclude from our model the unrealistic algorithm that a processor simply sends the broadcast message to all of the other processors at the same time. Similarly, the model prohibits the simultaneous receiving of multiple messages by any processor. That is, the model disallows the unrealistic implementation of a reduction operation by

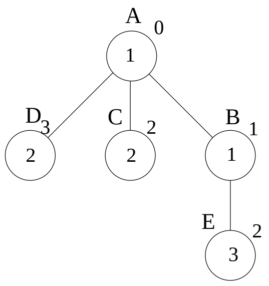

FIG. 1. A broadcast tree for five processors.

having one processor receive the messages from all of the other processors simultaneously. Although in practice many message-passing libraries provide non-blocking send and receive primitives, these simultaneous message transmissions are eventually serialized at the hardware level.

After defining the communication model, we can define other terminologies for the broadcast problem in a heterogeneous system. We define a broadcast tree as follows. Each node in the broadcast tree represents a processor in the cluster, and the root of the tree is the source processor for the broadcast. The children of a tree node p are the processors that receive the broadcast message from p. The *ready time* of a processor c is the time in which c completes receiving the broadcast message from the parent of c and is *ready* to send out messages of its own. In other words, the ready time of a processor c is the time that the parent of c started sending the message to c, plus the transmission time of the parent of c. Figure 1 illustrates a broadcast tree for a cluster of five processors, with transmission times 1, 1, 2, 3, 2, respectively. The number inside a tree node is its transmission time, and the number next to it is its ready time. Note that since all of the messages sent from the same source are serialized, the receive times of two siblings differ by at least the transmission time of their parent.

3. FASTEST-NODE-FIRST TECHNIQUE

It is difficult to find the optimal broadcast tree that minimizes the total broadcast time in a heterogeneous cluster; therefore a simple heuristic called *fastest-node-first* (FNF) is proposed in [3] to find a reasonably good broadcast schedule. The heuristic works as follows. In each iteration the algorithm chooses a sender from the set of processors that have received the broadcast message (denoted by A) and a receiver from the set that have not (denoted by B). The algorithm picks the sender s from A so that s will finish this transmission as early as possible, considering all of the transmissions that have been scheduled so far, and chooses the receiver r as the

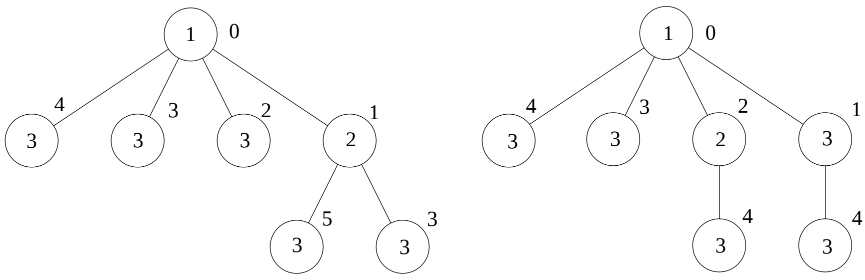

FIG. 2. An counterexample that FNF always produces the optimal broadcast time.

processor that has the minimum transmission time in B. Then *r* is moved from B to A and the algorithm iterates to find the next sender/receiver pair. The intuition behind this heuristic is that by sending the message to those fast processors first, it is likely that the messages will propagate more rapidly.

The FNF technique is very effective in reducing broadcast time. The FNF has been shown in simulation to find the optimal broadcast time, with high probability, when the transmission times are randomly chosen from a given table [3]. The FNF technique also delivers good communication efficiency in actual experiments. In addition, FNF is simple to implement and easy to compute.

Despite its efficiency in scheduling broadcasts in heterogeneous systems, the FNF heuristic does not guarantee an optimal broadcast time [3, 6]. A simple example is shown in Fig. 2, and a more complicated one is given in [6]. The number inside a tree node indicates its transmission time, and the number next to it is its ready time. Let *p* be the only processor with transmission time 2 in this cluster. According to the FNF principle, the root processor will first send the message to *p* before the rest of the processors. The resulting broadcast tree (on the left) has a total communication time of 5. On the other hand, an optimal scheduling is for the root to send the message to *p* in the second round, as indicated by the tree on the right in Fig. 2. The optimal broadcast tree requires only four time steps, one less than the tree by FNF on the left.

4. THEORETICAL RESULTS

Despite the fact that FNF cannot guarantee an optimal broadcast time, we show that FNF is optimal in some special cases of heterogeneous clusters. Based on the results on these special cases, we show that FNF has a completive ratio of 2.

We will need the following two theorems from [3] to prove the optimality of FNF in the special cases of heterogeneous systems.

THEOREM 4.1 [3]. *There exists an optimal broadcast tree T in which all processors send message without delay. That is, for all processor p in T, starting from its ready time, p repeatedly sends a message with a period of its transmission time until the broadcast ends.*

THEOREM 4.2 [3]. *There exists an optimal broadcast tree T in which every processor has a transmission time no less than the transmission time of its parent.*

Note that from the definition of optimality in Theorem 4.2 we consider the optimal broadcast schedule from *all* possible sources. We will follow this definition and investigate the optimality when the source is given in Section 4.4.

With Theorem 4.1, we can simply discard those trees that will delay messages and still find the optimal schedule. The proof of Theorem 4.2 follows from the observation that by exchanging a child node with a parent that has a longer transmission time, the final broadcast time will not increase. As a result we assume, without lose of generality and throughout this paper, that every processor in an optimal broadcast tree has a transmission time no less than its parent's.

Since there is no delay within the broadcast tress, we can represent a broadcast tree as a sequence of processors sorted in their ready time. Recall the sets A and B in the description of FNF. Since no delay is allowed, any scheduling method must schedule s, the processor in A that could have completed a transmission at the earliest time, to send a message immediately. Formally we define $S = (s_0, \ldots, s_{n-1})$ to be a sequence of n processors sorted in their ready time, i.e., the processors will be moved from B to A in the order defined by S. Therefore, for FNF the processors will appear in S in non-decreasing transmission time order, i.e., the processors will receive the broadcast according to their transmission time. Let $r(s_i)$ denote the ready time of s_i, then the total broadcast time of S (denoted by $T_{\mathrm{brdcst}}(S)$) is by definition $r(s_{n-1})$. A broadcast sequence S is *optimal* if and only if for any other permutation of S (denoted by S'), $t_{\mathrm{brdcst}}(S) \leq t_{\mathrm{brdcst}}(S')$. Note that by this definition we consider the schedule for all possible broadcast sources.

Let $t(p)$ be the transmission time of a processor p, and let $NS_S(p, t)$ be the *number* of messages successfully *sent* at and before time t by p in the sequence S. Formally, $NS_S(p, t)$ is the minimum non-negative integer k such that $r(p) + k * t(p) \geq t$, for $t \geq r(p)$. Following this notation, we can define the ready time $r(s_i)$ recursively by the following equations:

$$r(s_0) = 0$$

$$r(s_i) = \min\left\{ t \,\middle|\, \sum_{j=0}^{i-1} NS_S(s_j, t) \geq i \right\}, \qquad 1 \leq i \leq n-1. \tag{1}$$

4.1. *Fastest Nodes First*

We first establish the lemma that all of the fastest processors should send messages before all others. Without loss of generality, we assume that the transmission time of the fastest processors is 1. Consider an optimal sequence $S = (s_0, s_1, \dots, s_{n-1})$. From Theorem 4.2 we can argue that $t(s_0) = 1$, and $r(s_i)$ must be an integer if $t(s_i) = 1$ since only a fastest processor can send a message to a fastest processor.

Suppose there are fastest processors appearing after slower processors in S. Let $p = s_j$ be the first such processor in S. We show that among the slower processors appearing before p, one of them (denoted by q) became ready one time step ahead of p, i.e., $r(q) = r(p) - 1$.

LEMMA 4.1. *Let $S = (s_0, s_1, \dots, s_{n-1})$ be an optimal broadcast sequence, and let $p = s_j$ be the first fast processor appearing after slower processors in S. If $r(s_j) = t$, then there exists an $i < j$ such that $t(s_i) > t(s_j)$ and $r(s_i) = t - 1$.*

Proof. First we show that there exists a set of processors with ready time $t - 1$. Since p is a fastest processor, t must be an integer by Theorem 4.2. In addition, the root of the tree must be a fastest processor, and it will send out messages at integer time steps, including $t - 1$.

We prove the lemma by contradiction. Let us assume that all processors that became ready at $t - 1$ are fastest processors, and assume w to be one of them. We will consider two cases. First we assume that there is no slower processor with ready time between $t - 1$ and t. As a result p will not be the first fast processor appearing after slow processors since w became ready at time $t - 1$, and the slower processor appearing before p must appear before w as well.

In the second case, there does exist a set of slower processors (denoted by Q) with ready time between $t - 1$ and t. Let P be the set of processors that sent messages to processors in Q. We argue that all of the processors in P are slow processors since all of the fast processors send messages at integer time. Therefore, the ready time of any processor in P is before $t - 1$ since its transmission time is greater than 1. However, we know that a fastest processor w is ready at time $t - 1$, which became ready after those slower processors in P. As a result p cannot be the first fastest processor appearing after slower processors either. ∎

Let S be an optimal sequence and let $p = s_j$ denote a fastest processor that appears after the slower processor $q = s_i$ with ready time $r(p) - 1$ in Lemma 4.1. We show that by exchanging p with q in S, i.e., letting $S' = (s_0, \ldots, s_{i-1}, p, s_{i+1}, \ldots, s_{j-1}, q, s_{j+1} \ldots, s_{n-1})$, we can prevent the total broadcast time from increasing. In other words, S' has the same optimal broadcast time as S. First we establish the new ready time for p and q after the exchange.

LEMMA 4.2. *By modifying S into S' as described above, the ready time of p is made earlier from t to $t - 1$, and the ready time of q is delayed from $t - 1$ to $t' \leq t$. As a result $NS_{S'}(p, T) + NS_{S'}(q, T) \geq NS_S(p, T) + NS_S(q, T)$, for $T \geq t$.*

Proof. Since the first i processors of S' are the same as in S, the ready time of p in S' is $t - 1$, the same as the ready time of q in S. Similarly the ready time of s_m in S', for $i < m < j$, is the same as in S because whether p or q became ready at time $t - 1$, it will not send any message until time t. On the other hand, the ready time of q in S' is delayed by one time step. Now consider the new NS function for S'. Since p is moved forward one time step, an interval as long as its transmission time, $NS_{S'}(p, T) = NS_S(p, T) + 1$ for $T \geq t$. On the other hand, q is delayed by one time step, which is less than its own transmission time. As a result $NS_S(q, T) \leq NS_{S'}(q, T) \leq NS_S(q, T) + 1$ for $T \geq t$, and the lemma follows. ∎

After establishing the effects of exchanging the two processors on the new NS function, we argue that the ready time of the last $n - j$ processors will not be delayed from S to S'. We prove this statement by induction, and the following lemma serves as the induction base.

LEMMA 4.3. *The ready time of s_{j+1} in S' is no later than in S.*

Proof. The lemma follows from Lemma 4.2 and the fact that the ready time of the first $j + 1$ processors in the sequence is not changed, except p and q. Here we use the subscript to indicate whether the NS function is defined on S or S', and for ease of notation we remove the same second parameter t from all occurrences of NS functions.

$$
\begin{aligned}
r_{S'}(s_{j+1}) &= \min\left\{ t \,\middle|\, \sum_{l=0}^{j} NS_{S'}(s_l) \geq j+1 \right\} \\
&= \min\left\{ t \,\middle|\, \left(\sum_{l=0,\, l\neq i}^{j-1} NS_{S'}(s_l) \right) + NS_{S'}(p) + NS_{S'}(q) \geq j+1 \right\} \\
&= \min\left\{ t \,\middle|\, \left(\sum_{l=0,\, l\neq i}^{j-1} NS_{S}(s_l) \right) + NS_{S'}(p) + NS_{S'}(q) \geq j+1 \right\}
\end{aligned}
$$

$$\le \min\left\{t \,\middle|\, \left(\sum_{l=0,\, l\neq i}^{j-1} NS_S(s_l)\right) + NS_S(p) + NS_S(q) \ge j+1\right\}$$
$$= r_S(s_{j+1}).$$

∎

Now we complete the induction.

LEMMA 4.4. *The ready time of s_l in S' is no later than in S, for $j+1 \le l \le n-1$.*

Proof. We complete the proof by the induction step. Assume that the receive time of s_{j+m} in S' is no later than in S, for $1 \le m \le n-j-1$. Again for ease of notation we remove the same second parameter t from all occurrences of NS functions.

$$\begin{aligned}
r_{S'}(s_{j+m+1}) &= \min\left\{t \,\middle|\, \sum_{l=0}^{j+m} NS_{S'}(s_l) \ge j+m+1\right\} \\
&= \min\left\{t \,\middle|\, \left(\left(\sum_{l=0,\, l\neq i}^{j-1} NS_{S'}(s'_l)\right) + NS_{S'}(p) \right.\right. \\
&\qquad \left.\left. + NS_{S'}(q) + \sum_{l=j+1}^{j+m} NS_{S'}(s'_l)\right) \ge j+m+1\right\} \\
&\le \min\left\{t \,\middle|\, \left(\left(\sum_{l=0,\, l\neq i}^{j-1} NS_S(s_l)\right) + NS_S(p) \right.\right. \\
&\qquad \left.\left. + NS_S(q) + \sum_{l=j+1}^{j+m} NS_{S'}(s'_l)\right) \ge j+m+1\right\} \\
&\le \min\left\{t \,\middle|\, \left(\left(\sum_{l=0,\, l\neq i}^{j-1} NS_S(s_l)\right) + NS_S(p) \right.\right. \\
&\qquad \left.\left. + NS_S(q) + \sum_{l=j+1}^{j+m} NS_S(s_l)\right) \ge j+m+1\right\} \\
&= r_S(j+m+1).
\end{aligned}$$

∎

The last inequality follows from the induction hypothesis that all of the processors from s_{j+1} to s_{j+m} have an earlier ready time in S' than in S, so they will have a larger NS function and a smaller t to satisfy the equation in (1). One immediate result from Lemmas 4.3 and 4.4 is that for any broadcast sequence, including the optimal ones, making the fastest processors ready as early as possible will never increase the total broadcast time. Now we have the following theorem.

THEOREM 4.3. *There exists an optimal broadcast sequence in which all of the fastest processors appear before all of the other processors.*

4.2. *Special Cases*

We consider two special cases in which FNF guarantees a minimum broadcast time. First we consider the case that there are only two classes of processors in the cluster. The second case is that the transmission time of any slower processor is a multiple of any faster processors.

4.2.1. *Two Classes of Processors*

THEOREM 4.4. *The FNF algorithm gives an optimal broadcast time when the number of classes of processors is two, but does not guarantee the optimal broadcast time when the number of classes of processors is three.*

Proof. Given any optimal broadcast sequence consisting of two classes of processors, we can always make the ready time of a faster processor earlier should it appear after any slower processors, and the resulting sequence is still optimal. We can repeat this process until no such faster processor exists, and the resulting sequence is the same as the one given by FNF. The second part of the theorem follows from Fig. 2. ▮

In practice it is very likely that a cluster consists of only a small number of types of processors since they are often purchased in batches. This result ensures that the FNF algorithm can achieve an optimal broadcast time when the number of classes of processors is two. For clusters consisting of more processor types, FNF has also been proven effective through simulations [3].

4.2.2. *Multiple of Transmission Time*

The FNF algorithm also gives an optimal broadcast time when the transmission time of any slower processor in the cluster is a multiple of any faster processors. Without loss of generality, let us again assume that the transmission time of the fastest processors is 1. First we show that Lemma 4.1 is true for all processors, instead of only for the fastest ones, for such clusters.

LEMMA 4.5. *Let $S = (s_0, s_1, \ldots, s_{n-1})$ be an optimal broadcast sequence for a cluster where the transmission time of any processor is a multiple of any faster processor. Suppose there exists a processor $p = s_j$ that becomes ready after a slower processor in S; then there exists an $i < j$ such that $q = s_i$ is a slower processor and $r(q) = r(p) - 1$.*

Proof. The proof is similar to the proof of Lemma 4.1 and is in fact easier since now the ready times of all processors are integers. We consider the first processor p that appears after a slower processor. Similar to the argument in Lemma 4.1, we argue that there exists a set of processors that become ready one step ahead of p because the root of the broadcast tree is a fastest processor. If any such processor is slower than p then the lemma follows. If this is not the case, the processor that is slower than p but appears before p will be ready at time $r(p) - 2$ or earlier, and p will not be the first processor that appears after a slower processor. ∎

Similarly, we argue, as in Lemma 4.5, that it is always possible to switch a processor p with a slower processor that became ready one step ahead of p. This modification will not increase the total broadcast time, as indicated by the following lemma. Again notice that this is true for *any* processor, not just only for the fastest ones, as in Lemma 4.5.

LEMMA 4.6. *By switching p with q in Lemma* 4.5, *the ready time of p is moved forward from t to $t-1$, the ready time of q is delayed from $t-1$ to t, and $NS_{S'}(p, T) + NS_{S'}(q, T) \geq NS_S(p, T) + NS_S(q, T)$, for $T \geq t$.*

Proof. Let us consider the change to NS function from q's point of view. Since q is delayed by only one time step, NS_S is at most greater than $NS_{S'}$ by 1, and this decrease only happens at time interval $[r(q) + kt(q), r(q) + kt(q) + 1)$, where k is a positive integer and $r(q)$ is the ready time of q in S. Note that this interval includes the time $r(q) + kt(q)$ but not $r(q) + kt(q) + 1$. However, during this interval $NS_{S'}(p)$ will be larger than $NS_S(p)$ by one since $t(q)$ is a multiple of $t(p)$ and p became ready one step earlier in S' than in S. This increase compensates for the decrease due to q and the lemma follows. ∎

With Lemma 4.6 in place we have the following theorem.

THEOREM 4.5. *The FNF algorithm gives an optimal broadcast time when the transmission time of any slower processor in the cluster is a multiple of any faster processors.*

4.3. *Competitive Ratio Analysis*

Theorem 4.5 by itself is not very useful in practice since most clusters do not have such nice transmission time properties. However, we can use Theorem 4.5 to show that FNF is actually an approximation algorithm of competitive ratio 2. This somehow explains that in simulations FNF always produces very good schedules (within 1% of the optimal [3]).

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 $(k > 0)$. We will call this kind of cluster a *power* 2 cluster. From Theorem 4.5 it immediately follows that FNF produces the optimal broadcast time for all power 2 clusters.

By increasing the transmission time of processors, we can transform a heterogeneous cluster into a power 2 cluster. We increase the transmission time of each processor p to $2^{\lceil \log t(p) \rceil}$, i.e., the smallest power of 2 that is not less than the original transmission time. We will show that FNF, optimal for the transformed cluster, also gives a schedule within twice of the optimal time for the original cluster.

THEOREM 4.6. *The FNF scheduling has a total reduction time no greater than twice that of the optimal schedule.*

Proof. Let S be an optimal broadcast sequence for a heterogeneous cluster C, and let C' be the power 2 cluster transformed from C. Let T and T' be the broadcast times of S for C and C', respectively, i.e., before and after the power 2 cluster transformation. We argue that this increase in transmission time will at most double the total broadcast time, i.e., $T' \leq 2T$. We can use a simple induction on i to argue that s_i, which is ready at time $r(s_i)$ for C, becomes ready no later than $2r(s_i)$ for C'. The induction step follows from the fact that all of the previous s_j for $j < i$ become ready no later than $2r(s_j)$ for C', and their transmission time at most doubles from C to C'.

Now we apply FNF scheduling on C' and let T'' be the resulting broadcast time. Since C' is a power 2 cluster, it immediately follows from Theorem 4.5 that T'' is no more than T'. Finally, we apply the same FNF scheduling on C and let T^* be the resulting broadcast time. T^* should be no more than T'' since the transmission time of each corresponding processor is higher in C' than in C. As a result T^* is no greater than T'', which is no greater than T', which is no more than $2T$. ∎

4.4. *Broadcast for Specified Source*

The previous sections describe theoretical results for the broadcast problem in which the source of the broadcast can be any processor. That is, the *optimal* schedule is the fastest one among all possible schedules considering all possible sources, and, as suggested by Theorem 4.2, there exists an optimal schedule in which the source is the fastest processor. In practice, however, the application usually will specify the source of the broadcast, i.e., during the computation a particular processor has to broadcast important information for other processors to proceed. We should show that, under the constraint that the source is given, which is not necessarily a fastest processor, Theorem 4.6 is still valid.

We will use the same notation as in the general case, but with the following modification. First, we still define a schedule $S = (s_0, \ldots, s_{n-1})$ as a sequence of processors, and s_0 is the specified source for the broadcast. Note that we can no longer assume that s_0 is the fastest processor. Let $t(s_i)$ still be the transmission time of s_i, and assume the fastest processor has a transmission time 1. To simplify the notation we will assume that the time will start at $-t(s_0)$, so that the ready time of the processor that receives the first message from the source s_0, i.e., s_1, will be 0. We first show that for any power 2 cluster, there exists an optimal schedule in which s_1 is the fastest processor.

LEMMA 4.7. *Let C be a power 2 cluster. There exists an optimal schedule $S = (s_0, \ldots, s_{n-1})$ such that s_1 is the fastest processor.*

Proof. Without loss of generality we assume that the source s_0 is not the fastest processor. Let S be any optimal schedule and let q be the second processor in S. We consider the case where q is not the fastest processor in C, i.e., $t(q) > 1$. From this assumption we argue that the ready time of the fastest processor, denoted by p, has a ready time that is earliest at $\min(t(s_0), t(q)) > 1$. Now we switch p and q in S and let p be the second processor in the sequence S. Similar to Lemma 4.2, we argue that the increase of the NS function from p is more than enough to compensate for the decrease of q since p has a shorter transmission time. As a result the modified schedule is also optimal, and the lemma follows. ∎

With Lemma 4.7 in place we can argue that there will be processors ready at time 0, 1, 2, and so on since s_1 has a transmission time 1. Now we can proceed to the following lemma, which is similar to Lemmas 4.5 and 4.6.

LEMMA 4.8. *Let $S = (s_0, s_1, \ldots, s_{n-1})$ be an optimal broadcast sequence for a power 2 cluster, in which s_0 is the specified source and s_1 is the fastest processor. Let $p = s_j$ be the first fast processor appearing after any slower processor other than s_0.*

1. *If $r(p) = t$, then there exists an $i < j$ such that $q = s_i$ and $t(q) > t(p)$ and $r(q) = r(p) - 1$.*

2. *By switching p with q, the ready time of p is moved forward from t to $t - 1$, the ready time of q is delayed from $t - 1$ to t, and for the resulting new schedule S', we have $NS_{S'}(p, T) + NS_{S'}(q, T) \geq NS_S(p, T) + NS_S(q, T)$, for $T \geq t$.*

Proof. The proof is also similar to Lemmas 4.5 and 4.6. We notice that from Lemma 4.7 we are certain that there exists a set of processors that become ready one time step ahead of p. Then it follows that one of these processors must be a slower processor, otherwise p will not be the first faster processor appearing after slower ones. The second part of the lemma follows from a similar argument in Lemma 4.6. Notice that we exclude s_0 in the lemma since we cannot exchange the order of the specified source. ∎

Finally, we conclude that FNF is also optimal for a power 2 cluster when the source is given and establish the following competitive ratio.

THEOREM 4.7. *The FNF scheduling has a total reduction time no greater than twice that of the optimal schedule when the source is given.*

5. HEURISTIC SEARCH

The previous section describes the theoretical results that guarantee the optimality of the FNF method under special cases and provide a performance guarantee for general cases. However, in practice one may want to find the optimal broadcast 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 FNF does not guarantee optimality. This section describes the techniques that we used to speed up the search process.

As described in Section 3, any broadcast tree can be converted into 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 of these permutations, even by a branch-and-bound procedure. To overcome this problem, we conduct experiments to show that by using Theorem 4.2 in [3] and Theorem 4.3 in this paper 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 faster processors appearing first will be examined first. Formally we define the *priority* of a sequence to be the number processors that have transmission times shorter than or equal to that of the next processor in the sequence. In other words, the FNF schedule has the highest priority and will be considered first. In addition, from Theorem 4.3 we know that we can ignore all sequences in which fastest processors are not at the beginning and still find the optimal schedule. This dramatically reduces the search space since now we only have to schedule those processors that are not from the fastest processor group.

The second technique is to apply Theorem 4.2 so that when a slow processor is scheduled to send the message to a faster processor, we can stop the search at that subtree immediately. In addition, it is possible for several senders to complete simultaneously so that more than one processor can be the receiver at the same time. In that case if any sender is slower 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, then the entire subtree is pruned. This technique is most effective when the difference among processor speeds is large.

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

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

Table 1 compares the efficiency of our algorithm with a simple branch-and-bound search. The first two columns 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 seconds), 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. Guided by various heuristics described

TABLE 1
Comparison of Two Search Programs

	Search tree		FNF search			Generic branch-and-bound			Search time
P	N	L	n	Time	n/N	n	Time	n/N	ratio
10	8895	2750	101	0.0005	1.135%	1876	0.0089	21.091%	17.9
11	26200	8094	132	0.0008	0.504%	4956	0.0278	18.916%	36.8
12	65666	20118	180	0.0012	0.274%	10406	0.0671	15.817%	57.7
13	182749	55812	364	0.0027	0.199%	25517	0.1857	13.963%	69.7
14	479988	146226	549	0.0046	0.114%	62318	0.5150	12.983%	111.9
15	1130166	342050	1111	0.0101	0.098%	144507	1.3335	12.786%	132.1
16	3156025	953521	2007	0.0200	0.064%	300785	3.0448	9.531%	152.1
17	9193712	2773002	3002	0.0349	0.033%	1029079	11.6970	11.193%	334.5
18	31946795	9688499	5721	0.0709	0.018%	2658041	32.9900	8.320%	465.3
19	86500614	26089395	6124	0.0825	0.007%	7098066	96.4370	8.206%	1169.6
20	220708439	66378464	12693	0.1821	0.006%	13926482	205.1097	6.310%	1126.2
21	658075130	198533991	27418	0.4348	0.004%	29399414	576.2480	4.470%	1324.3

earlier, our algorithm searches many fewer tree nodes than the generic branch-and-bound method and consequently runs much faster. For large clusters our algorithm runs about 1300 times faster than the generic algorithm and can find the optimal solution within a fraction of a second, even for clusters consisting of up to 21 nodes.

6. CONCLUSION

FNF is a very useful technique in reducing broadcast time. We show that in several special cases it always gives an optimal broadcast time. In simulations it can find the optimal solution with very high probability when the number of processors is small, and the transmission time is randomly chosen from a small table [3]. In practice it also delivers good performance in actual NOW systems. The schedule is easy to compute and can be updated incrementally.

This paper also derives a performance guarantee for the FNF algorithm for general heterogeneous clusters. We show that FNF guarantees the total time to be within twice the time from an optimal schedule. It will be more interesting if one can derive a bound on the difference, instead of on the factor, between the schedule from the proposed algorithm and the optimal one.

This paper also suggests techniques to speed up the search process of finding an optimal schedule. We combined three key techniques into the algorithm: scheduling all fastest nodes first, a sender cannot be slower than its receiver, and branch-and-bound. This combined approach dramatically reduces the search space and provides an optimal schedule within a fraction of a second, for clusters up to 21 processors.

There are many research issues open for investigation. For example, 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 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, all-to-all reduction, 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.

ACKNOWLEDGMENTS

The authors thank Dr. Da-Wei Wang for helpful discussions and Mr. Tzu-Hao Sheng for implementing the heuristic search program. This work is supported in part by the National Science Council of Taiwan, under Grant NSC 89-2213-E-194-009.

REFERENCES

1. "Message Passing Interface Forum," March 1994.
2. T. Anderson, D. Culler and D. Patterson, A case for networks of workstations (now), *IEEE Micro.* (1995), 54–64.
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, pp. 460–467.
4. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, Multicasting in heterogeneous networks, *in* "Proceedings of the 13th Annual ACM Symposium on Theory of Computing," 1998, pp. 448–453.
5. A. Bar-Noy and S. Kipnis, Designing broadcast algorithms in the postal model for message-passing systems, *Math. Systems Theory* **27**(5) (1994), 431–452.
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, pp. 15–24.
7. M. Dinneen, M. Fellows, and V. Faber, Algebraic construction of efficient networks, *in* "Applied Algebra, Algebraic Algorithms, and Error Correcting Codes," vol. 9(LNCS 539), pp. 152–158, 1991.
8. J. Bruck, D. Dolev, C. Ho, M. Rosu, and R. Strong, Efficient message passing interface(mpi) for Parallel computing on clusters of workstations, *J. Parallel Distributed Comput.* **40**, (1997), 19–34.
9. M. R. Garey and D. S. Johnson, "Computer and Intractability: A Guide to the Theory of NP-Completeness," Freeman, New York, 1979.
10. L. Gargang and U. Vaccaro, On the construction of minimal broadcast networks, *Network* **19** (1989), 673–689.
11. M. Grigni and D. Peleg, Tight bounds on minimum broadcast networks, *SIAM J. Discrete Math.* **4** (1991), 207–222.
12. W. Gropp, E. Lusk, N. Doss, and A. Skjellum, A high-performance, portable implementation of the mpi: a message passing interface standard, *Parallel Computing* **22** (1996), 789–828.
13. S. M. Hedetniemi, S. T. Hedetniem, and A. L. Liestman, A survey of gossiping and broadcasting in communication networks, *Networks* **18** (1991), 129–134.
14. R. Karp, A. Sahay, E. Santos, and K. E. Schauser, Optimal broadcast and summation in the logp model, *in* "Proceedings of 5th Annual Symposium on Parallel Algorithms and Architectures," 1993, pp. 142–153.
15. R. Kesavan, K. Bondalapati, and D. Panda, Multicast on irregular switch-based networks with wormhole routing, *in* "Proceedings of the International Symposium on High Performance Computer Architecture," 1997, pp. 48–57.
16. A. L. Liestman and J. G. Peters, Broadcast networks of bounded degree, *SIAM J. Discrete Math.* **1** (1988), 531–540.
17. P. Liu and D. Wang, Reduction optimization in heterogeneous cluster environments, *in* "Proceedings of the International Parallel and Distributed Processing Symposium," 2000, pp. 477–482.
18. D. Richards and A. L. Liestman, Generalization of broadcasting and gossiping, *Networks* **18** (1988), 125–138.
19. J. A. Ventura and X. Weng, A new method for constructing minimal broadcast networks, *Networks* **23** (1993), 481–497.
20. D. B. West, A class of solutions to the gossip problem, *Discrete Math.* **39** (1992), 285–310.