An Approximation Algorithm for Broadcast Scheduling in Heterogeneous Clusters

Pangfeng Liu[1], Da-Wei Wang[2], and Yi-Heng Guo[3]

[1] Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
[2] Institute of Information Science, Academia Sinica
[3] Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan.

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 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) [2] is very effective in reducing broadcast time for heterogeneous cluster systems. Despite the fact that FNF heuristic does not guarantee an optimal broadcast time for general heterogeneous network of workstation, we prove that FNF always gives near optimal broadcast time in a special case of cluster, and this finding helps us show that FNF delivers guaranteed performance for general clusters. In a previous paper we showed a similar bound on the competitive ratio in a send-only communication model. This paper extends the result to a more realistic sender-receiver model. We show that FNF gives a total broadcast of $2T + \beta$, where T is the optimum time and β is a constant. This improves over the previous bound on $2\alpha T + \beta$ [17], where α is a theoretically unbounded ratio of the processor performance in the cluster.

1 Introduction

Network of workstation (NOW) is a cost-effective alternative to massively parallel supercomputers [1]. 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 Mega bits per second of bandwidth with inexpensive Fast Ethernet adaptors and hubs.

Parallel to 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 [9, 8, 13]. 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 network of workstation) that has MPI implementation.

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.

Every workstation cluster, be it homogeneous or heterogeneous, requires efficient collective communication [2]. 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 any can go to the next phase. 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 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. [2], is very effective in designing broadcast protocols for heterogeneous cluster systems. The fastest-node-first technique schedules the processors to receive the broadcast in the order of their communication speed,

that is, the faster node should be scheduled earlier. Despite the fact that the FNF heuristic does *not* guarantee optimal broadcast time for every heterogeneous network of workstations, we show that FNF does give near optimal broadcast time when the communication time of any slower processor in the cluster is a multiple of any faster processor. Based on this result, we show that FNF is actually an approximation algorithm that guarantees a broadcast time within $2T + \beta$, where T is the optimal broadcast time and β is the maximum difference between two processors. This improves over the previous bound $2\alpha T + \beta$ [17] where α is the maximum ratio between receiving and sending costs, and can be arbitrarily large theoretically. In a previous paper [19] we show a similar result for a communication model where the communication cost is determined by the sender only. This paper shows that FNF can still achieve guaranteed performance when the model determines the communication costs based on both the sender and the receiver.

We also conduct experiments on the performance of the fastest-node-first technique. The cluster we construct in our simulation consists of three types of processors, and the number of nodes is 100. We construct the schedules from a random selection and FNF, and apply them on the heterogeneous cluster model. Experimental results indicate that FNF gives superior performance over random selection, for up to 2 times of throughput.

The rest of the paper is organized as follows: Section 2 describes the communication model in our treatment of broadcast problem in HNOW. Section 3 describes the fastest-node-first heuristic for broadcast in HNOW. Section 4 gives the theoretical results for broadcast. Section 5 describe the experimental results that we compare the completion time of our heuristics(FNF) with the random-select algorithms, 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 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 [15] 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, 15]. 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 an optimal broadcast schedule remains NP-hard [10]. Efficient algorithms and network topologies for other similar problems related to broadcast, including multiple broadcast, gossiping and reduction, can be found in [7, 11, 12, 14, 18, 21–23].

Various models for heterogeneous environments have also been proposed in the literature. Bar-Nod et al. introduced a heterogeneous 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. [2] proposed a simple model in which the heterogeneity among processors is characterized by the speed of sending processors, and show that a broadcast technique called *fastest-node-first* works well in practice. We will refer to this model as the *sender-only model*. Based on the sender-only model, an approximation algorithm for reduction with competitive ratio 2 is reported in [20], and the fastest- node-first technique is shown to be also 2-competitive [19]. Despite the fact that the sender-only model is simple and has a high level abstraction of network topology, the speed of the receiving processor is not accounted for. In a refined model proposed by Banikazemi et al. [3], communication overheads consists of both sending and receiving time, which we will refer to as the *sender-receiver* model. For the sender-receiver model the same fastest- node-first is proven (Libeskind-Hadas and Hartline [17]) to have a total time of no more than $2\alpha T + \beta$, where α is the maximum ratio between receiving and sending time, β is the maximum difference between two receiving time, and T is the optimal time. We adopt the sender- receiver model in this paper and improve this bound to $2T + \beta$. Other models for heterogeneous clusters include [6, 16].

2.1 Model Definition

The model is defined as follows: A heterogeneous cluster is defined as a collection of processors $p_0, p_1, ..., p_{n-1}$, each capable of point-to-point communication with any other processor in the cluster. Each processor is characterized by its speed of sending and receiving messages, and the network is characterized by the speed to route a message from the source to the destination. Formally, we define the *sending time* of a processor p, denoted by $s(p)$, to be the time it needs for p to send a unit of message into the network. The network is characterized by its latency L, which is the time for the message to go from its source to its destination. Finally we define the *receiving time* of a processor p, denoted by $r(p)$, to be the time it takes for p to retrieve the message from the network interface. We further assume that the processor speed is *consistent*, that is, if a processor p can send messages faster than another processor q, it can also receive the messages faster. Formally we assume that for two processors p and q, $s(p) \leq s(q)$ if and only if $r(p) \leq r(q)$.

The communication model dictates that the sender and receiver processors cannot engage in multiple message transmissions simultaneously. That is, a sender processor must complete its data transmission to the network before sending the next message, that is, a processor can only inject messages into the network at an interval specified by its sending time. This restriction is due to the fact that processor and communication networks have limited bandwidth,

therefore we would like to exclude from our model the unrealistic algorithm that a processor simply sends the broadcast message to all the other processors simultaneously. Similarly, the model prohibits the simultaneous receiving of multiple messages by any processor.

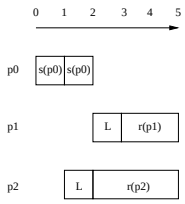

Fig. 1. A broadcast send-receive communication model.

2.2 Broadcast Problem Description

We consider an example with two fast processors p_0, and p_1, and one slow processor p_2. The fast processors have sending time 1 and receiving time 2, the slow processor has sending time 2 and receiving time 3, and the network latency L is 1. We assume that p_0 is the source and that it sends a message to p_2 at time 0. The message enters the network at time 1 since $s(p_0)$ is 1, and leaves the network at time $1 + L = 2$, and is received by p_2 at time $2 + r(p_2) = 5$. After sending a message into the network at time 1, p_0 can immediately send another message to p_1 and inject it into the network at time $1 + s(p_0) = 2$. The message is finally received by p_1 at time $2 + L + r(p_1) = 5$. See Figure 1 for an illustration.

2.3 Simplified Model Description

We can simplify the model as follows: Since a receiving node p always has to wait for $L + r(p)$ time steps before it actually receives the message, we can add the network latency L into the receiving time. The processor p_2 therefore receives its message at time $s(p_0) + r(p_2) = 1 + 4 = 5$, and p_1 receives its message from p_0 at time $2s(p_0) + r(p_1) = 5$. See Figure 2 for an illustration.

Assume that a processor q sends a message to the other processor p at time t, then p becomes *ready to receive* at time $t + s(q)$, since p now can start receiving the message, and we denote the ready to receive time of p by $R(p)$. At time $t + s(q) + r(p)$ p becomes *ready to send* because it can start sending its own message now, and we use $S(p)$ to denote the ready to send time of p. That is, a processor p can finish sending messages into the network at time $S(p) + s(p), S(p) + 2s(p), ..., S(p) + i * s(p)$, where i is a positive integer, until the broadcast is finished.

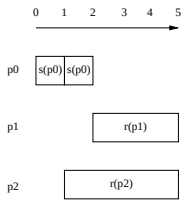

Fig. 2. A simplified send-receive communication model.

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 [2] to find a reasonably good broadcast schedule for the original sender-only heterogeneous model [2].

3.1 Fastest-Node-First Scheduling for Broadcast

The FNF 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 because, as the chosen one, it can inject the message into the network as early as possible. The algorithm then chooses the fastest processor in B as the destination of s. After the assignment, r is moved from B to A and the algorithm iterates to find the next sender/receiver pair. Note that this same technique can be applied to both models – the sender only and the sender-receiver heterogeneous models – since we assume that the sending and receiving times are consistent among processors. 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 fastest-node-first technique is very effective in reducing broadcast time [2, 17, 19]. The FNF has been shown in simulation to have a high probability to find the optimal broadcast time when the transmission time is randomly chosen from a given table [2]. The FNF technique also delivers good communication efficiency in actual experiments. In addition, FNF is simple to implement and easy to compute.

3.2 FNF not Guarantee Optimal Broadcast Time

Despite its efficiency in scheduling broadcast in heterogeneous systems, the FNF heuristic does not guarantee optimal broadcast time [2, 6] in sender-only model. Since the sender-only model is a special case of the sender-receiver model, FNF is not optimal in the sender-receiver model either. For example, in the situation of Figure 1 FNF will not achieve optimal time, as Figure 3 indicates.

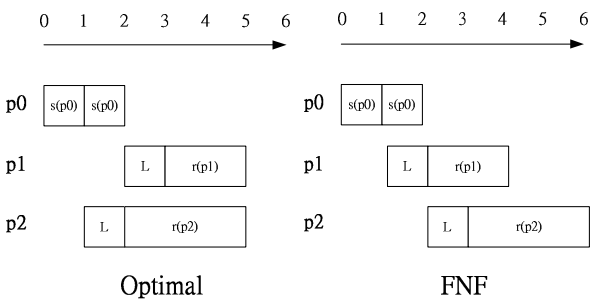

Fig. 3. A counterexample that FNF always produces the optimal broadcast time since the fast processor p_0 sends message to the faster p_1 first, instead of the slower p_2.

4 Theoretical Results

Despite the fact that FNF cannot guarantee optimal broadcast time, we show that FNF is optimal in some special cases of heterogeneous clusters. Based on the results of these special cases, we show that the fastest-node-first algorithm produces a schedule with guaranteed performance.

Theorem 1. *[2]*

There exists an optimal schedule in which all processors sends messages without delay. That is, for all processor p in T, starting from its ready to send time, p repeatedly sends a message with a period of its sending time until the broadcast ends.

With Theorem 1, we can simply discard those schedules that will delay messages, and still find the optimal one. Since there is no delay, we can characterize a schedule as a sequence of processors sorted in their *ready to receive time*. Since no delay is allowed, any scheduling method must schedule s, the processor in A that could have completed the sending at the earliest time, to send a message immediately. Formally we define $P = (p_0, ..., p_{n-1})$ to be a sequence of n processors sorted in their ready to receive time and the processors appear in P in non-decreasing sending speed, except for the source s_0. The total broadcast time of P (denoted by $T(P)$) is by definition $\max_{i=1}^{n-1} S(p_i)$, the latest *ready to send* time among all the processors[4]. A broadcast sequence P is *optimal* if and only if for any other permutation of P (denoted by P'), $T(P) \leq T(P')$.

Let p be a processor and $NS_P(p,t)$ be the *number* of messages successfully *sent* at and before time t by p in the sequence P. Formally, $NS_P(p,t) = \lfloor \frac{t-S(p)}{s(p)} \rfloor$, for $t \geq S(p)$. We can define ready to receive time $R(p_i)$ and ready to send time $S(p_i)$ recursively (Eqn. 1). that is, the ready to receive time of the i-th processor in P is the earliest time when the total number of messages sent by the first $i-1$ processors reaches i.

[4] Note that the processor that has the latest ready to receive time may not have the latest ready to send time.

$$
\begin{aligned}
&R(p_0) = 0 \quad \text{and} \quad S(p_0) = 0 \\
&R(p_i) = \min\{t | \sum_{j=0}^{i-1} NS_P(p_j, t) \geq i\}, \quad 1 \leq i \leq n-1 \\
&S(p_i) = R(p_i) + r(p_i), \quad 1 \leq i \leq n-1
\end{aligned} \tag{1}
$$

4.1 Power 2 clusters

In this section we consider a special case of heterogeneous clusters in which all the sending and receiving costs are power of 2, and we refer to such clusters as power 2 clusters [19]. Similar notation is also used in [17]. We show that FNF technique does guarantee minimum ready to receive time for the last processor receiving the broadcast message in a power 2 cluster, and this is the foundation of our competitive ratio analysis.

Henceforth we will focus on minimizing the ready to receive time of the last processor in a sequence $P = (p_0, ..., p_{n-1})$, which is denoted as $TR(P) = R(p_{n-1})$. We will later relate our finding with the latest ready to send time among all the processors, denoted by $TS(P) = \max_{i=0}^{n-1} S(p_i)$, which is the time the broadcast actually takes. We choose this approach since $TR(P)$ is much easier to handle in our mathematical analysis than $TS(P)$.

We first establish a lemma that it is always possible to switch a processor p with a slower processor q that became ready to receive right ahead of p (with the exception that q is the source) so that p and q will contribute more on the NS function after the switch. We then use an induction to show that this modification will not increase the ready to receive time of the processors thereafter, including the last one in the sequence. This leads to the optimality of FNF for the last ready to receive time in a power 2 cluster.

Lemma 1. *Let p be a first faster processor that became ready to receive right after a slower processor q in a sequence P, that is, $R(p) = t_1 > R(q) = t_0$, and $s(p) < s(q)$. By switching p with q in P we obtain a new sequence P'. Then, in this new sequence P', $R(p)$ is moved forward from t_1 to t_0, and $R(q)$ is delayed from t_0 to no later than t_1, and $NS_{P'}(p,t)+NS_{P'}(q,t) \geq NS_P(p,t)+NS_P(q,t)$, for $t \geq t_0$.*

Proof. Let's consider the time interval from t_0 to t_1. Since p is the first faster processor that becomes ready to receive *right after* a slower processor q, no processor becomes ready to receive between t_0 and t_1. Since, in P', p is moved to q's position in P, p has $R(p) = t_0$. As p is faster in sending *and* receiving, q becomes ready at or before t_1 from Equation 1. For our purpose we will assume that q becomes ready to receive at time t_1 since if the time is earlier, it is more likely that $NS_{P'}(p,t) + NS_{P'}(q,t) \geq NS_P(p,t) + NS_P(q,t)$, for $t \geq t_0$.

Let $d = t_1 - t_0$. Since all the ready to receive time is integer, d is at least 1. It is easy to see that when d is larger, $NS_{P'}(p,t) + NS_{P'}(q,t)$ is more likely

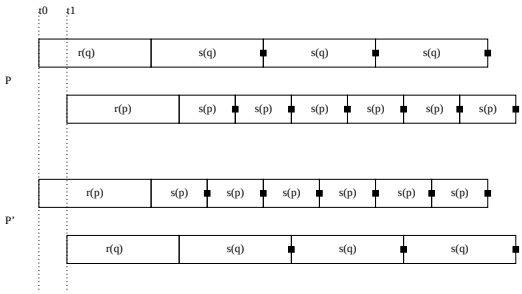

Fig. 4. An illustration that the NS function in P and P'.The black squares indicate where the NS function increases by 1. Note that the NS function in P' is no less than in P for all time later than t_0. In this example $r(p) = r(q) = 4$, $s(p) = 2$, $s(q) = 4$, and $d = 1$.

to be larger than $NS_S(p,t) + NS_S(q,t)$, when $t > t_0$. In fact, from p's point of view, when the sequence changes from P to P', the $NS(p)$ increases between $\lfloor \frac{d}{s(p)} \rfloor$ and $\lceil \frac{d}{s(p)} \rceil$, but the decrease in $NS(q)$ is only between $\lfloor \frac{d}{s(q)} \rfloor$ and $\lceil \frac{d}{s(q)} \rceil$. The increase in $NS(p)$ is larger than the decrease in $NS(q)$ when d is sufficiently large, since $s(q)$ is at least twice as large as $s(p)$. In addition, $r(p)$ is no larger than $r(q)$, and that means $NS(p)$ increases earlier than the decrease of $NS(q)$. Therefore, by moving p further ahead in time, it becomes easier for the increase of the NS function from p to compensate the decrease of the NS function from q, when the sequence changes from P to P'. Therefore it suffices to consider the worst case when $d = 1$.

Let us consider the change of NS function from q's point of view. q is delayed by only one time step, so $NS_S(q)$ is at most greater than $NS_{S'}(q)$ by 1, which only happens at time interval $[t_0 + r(q) + ks(q), t_0 + r(q) + ks(q) + 1)$, where k is a positive integer, $r(q)$ is the receiving time of q, and $s(q)$ is the sending time of q. See Figure 4 for an illustration. However, during this interval $NS_{P'}(p)$ will be larger than $NS_P(p)$ by one since $s(q)$ is a multiple of $s(p)$, and $r(q)$ is a multiple of $r(p)$ due to speed consistency. This increase compensates the decrease due to q and the Lemma follows.

After establishing the effects of exchanging the two processors on the NS function, we argue that the ready to receive time of the processors after p and q will not be delayed from P to P'. We prove this statement by an induction and the following lemma serves as the induction base:

Lemma 2. *Let p and q be the $(j-1)^{th}$ and j^{th} processor in P, then the ready to receive time of p_{j+1} in P' is no later than in P.*

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

$$
\begin{aligned}
R_{P'}(p_{j+1}) &= \min\{t|\sum_{l=0}^{j} NS_{P'}(p_l) \geq j+1\} \\
&= \min\{t|(\sum_{l=0}^{j-2} NS_{P'}(p_l)) + NS_{P'}(p) + NS_{P'}(q) \geq j+1\} \\
&= \min\{t|(\sum_{l=0}^{j-2} NS_{P}(p_l)) + NS_{P'}(p) + NS_{P'}(q) \geq j+1\} \\
&\leq \min\{t|(\sum_{l=0}^{j-2} NS_{P}(p_l)) + NS_{P}(p) + NS_{P}(q) \geq j+1\} \\
&= R_P(p_{j+1})
\end{aligned}
$$

Lemma 3. *The ready to receive time of p_l in P' is no later than in P, for $j+1 \leq l \leq n-1$.*

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

$$
\begin{aligned}
&R_{P'}(p_{j+m+1}) \\
&= \min\{t|\sum_{l=0}^{j+m} NS_{P'}(p_l) \geq j+m+1\} \\
&= \min\{t|((\sum_{l=0}^{j-2} NS_{P'}(p_l)) + NS_{P'}(p) + NS_{P'}(q) + \sum_{l=j+1}^{j+m} NS_{P'}(p_l)) \geq j+m+1\} \\
&\leq \min\{t|((\sum_{l=0}^{j-2} NS_{P}(p_l)) + NS_{P}(p) + NS_{P}(q) + \sum_{l=j+1}^{j+m} NS_{P'}(p_l)) \geq j+m+1\} \\
&\leq \min\{t|((\sum_{l=0}^{j-2} NS_{P}(p_l)) + NS_{P}(p) + NS_{P}(q) + \sum_{l=j+1}^{j+m} NS_{P}(p_l)) \geq j+m+1\} \\
&= R_P(p_{j+m+1})
\end{aligned}
$$

The second-to-the-last inequality follows from Lemma 1, and the last inequality follows from the induction hypothesis that all the processors from p_{j+1} to p_{j+m} have earlier ready to receive time (hence earlier ready to send time) in P' than in P, so they will have larger NS function, and a smaller t to satisfy Equation 1. One immediate result from Lemma 2 and 3 is that for any processor sequence of a power 2 cluster, including the optimal ones, the final ready to receive time will never be increased by making the faster processors ready to receive earlier than slower ones. Now we have the following theorem:

Theorem 2. *The fastest-node-first algorithm gives optimal final ready to receive time for a power 2 cluster.*

4.2 An approximation algorithm

We can use Theorem 2 to show that FNF is actually an approximation algorithm of competitive ratio 2 for the final ready to receive time. By increasing the transmission time of processors, we can transform any heterogeneous cluster into a power 2 cluster. We increase the sending and receiving time of each processor p to be $2^{\lceil \log s(p) \rceil}$ and $2^{\lceil \log r(p) \rceil}$ respectively. We will show that FNF, optimal for the transformed cluster, also gives a schedule at most twice that of the optimal final ready to receive time for the original cluster.

Theorem 3. *The fastest-node-first scheduling has a final ready to receive time no greater than twice that of the optimal final ready to receive time.*

Proof. Let P be a sequence that gives optimal final ready to receive time for a heterogeneous cluster C, and C' be the power 2 cluster transformed from C. We apply the same sequence P on C and C' and let T and T' be the final ready to receive time TR respectively, that is, before and after the power 2 cluster transformation. We argue that this increase in transmission time will at most double the TR, that is, $T' \leq 2T$. This is achieved by an induction on the processor index i. We argue that p_i, which is ready to receive at time $R(p_i)$ for C, becomes ready to receive no later then $2R(p_i)$ for C'. The induction step follows from the fact that all the previous p_j for $j < i$, become ready no later than $2R(p_j)$ for C', and that both the sending time of the previous p_j, $j < i$, and the receiving time of p_i are, at most doubled from C to C'.

Now we apply FNF scheduling on C' and let T'' be the resulting final ready to receive time. Since C' is a power 2 cluster, it follows from Theorem 2 that T'' is no more than T'. Finally, we apply the same FNF scheduling on C and let T^* be the resulting final ready to receive time. T^* should be no more than T'' since the sending and receiving times of each corresponding processor are higher in C' than in C. As a result T^* is no greater than T'', which in turn is no greater than T', which in turn is no more than $2T$.

Theorem 4. *The total broadcast time from fast-node-first technique is at most* $2T + \beta$*, where* T *is the optimal total broadcast time, and* β *is* $\max\{r(p_i)\} - 2\min\{r(p_i)\}$.

Proof. Let P be an optimal schedule in total broadcast time. Let p be the last processor that became ready to receive in P. As a result the optimal total broadcast time T is at least $R_P(p)+r(p)$. Let p' be the last processor that became ready to receive according to FNF. From Theorem 3 we have $R_{P'}(p') < 2R_P(p)$. Note that this inequality holds when P is any schedule, and not necessarily the optimal schedule for the final ready to receive time. The total broadcast time using FNF is $R_{P'}(p')+r(p')$, which is at most $2R_P(p)+r(p') = 2R_P(p)+2r(p)+r(p')-2r(p) \leq 2T + \beta$.

5 Experimental Results

This section describes the experimental results and compare the completion times of our heuristics (FNF) with those of a random-selection algorithm and a trivial lower bound. The experimental results indicate that FNF outperforms the random-selection algorithm by a factor of 2 in average, and is not very far away from the lower bound.

5.1 Experimental Environment

The input cluster configurations for our experiments are generated as follow: We assume that the number of classes in a cluster is 3. We vary the cluster size from 6 to 100, and set one third of the nodes to be fast processors, one third to be normal processors, and the others to be slow processors. For each processor in the same class, we assign the same sending time and receiving cost to it, that is, each node in the fast processor group has sending time 1 and receiving time 2, the sending and receiving time for normal processors are 5 and 6 respectively, finally the time for slow processors are 10 and 11.

We compare the results from FNF and random selection. We repeat the experiments for random-selection algorithm for 200 times and compute the average broadcast time. On the other hand since FNF is a deterministic algorithm, for each cluster size we test the FNF algorithm for only once.

5.2 FNF Heuristics and Random-Select Algorithm

We describe our implementation of FNF as follows: The program uses an array to represent the set of processors that have not yet received broadcast message (denoted by R-set), and a priority queue for the set of processors that have received the broadcast message (denoted by S-set). The elements in the R-set array are sorted according to their communication speed, and the elements in the S-set are ordered so that the processor that could send out the next message fastest has the highest priority. In other words, the processors in the S-set are sorted according to their availability in time. Initially the S-set has the broadcast source and the R-set is empty, and the simulation time is set to zero. The priority queue design simplifies and speeds up the simulation, since the simulator can be driven by events, not by time.

In each iteration we check if all nodes have received the broadcast message. If this is not the case then we will schedule the next message. We pick the next sender (with the highest priority) from the S-set priority queue, and the receiver that has the minimum receiving time from the R-set. After choosing the sender and the receiver, we calculate the updated available time for the sender and new available time for the receiver, and place them into the S-Set (the chosen receiver is therefore removed from the R-set). At the end the R-set will be empty and the ready-to-send time of the last receiver is the total broadcast time. Figure 5 gives an example of a broadcast scheduling among 6 node.

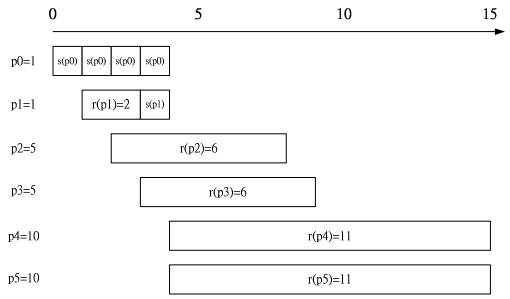

Fig. 5. The example of FNF algorithm under 6 node case.

We now describe the random-selection algorithm. Due to the random nature of this algorithm, we will not need to maintain any priority queue or sorted array. We randomly choose a sender from the S-set and a receiver from the R-set for the next message. We repeatedly schedule the transmission until all processors receive the message. The average time for the last receiver to receive its messages is the time that we are interested in.

5.3 Timing Comparison

Figure 6 shows the experimental results. The completion time of FNF is about half of the average time of random-selection algorithm.

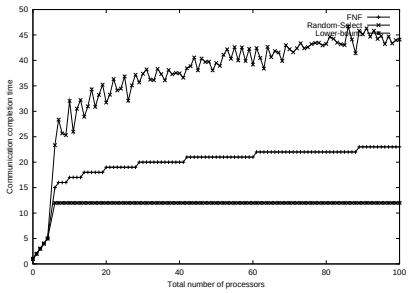

Fig. 6. The comparison of two scheduling algorithms.

We also give a lower bound on the optimal communication time for our experimental cluster. No matter how the processors are scheduled, the broadcast source must spend at least one unit of time to send the message, and a slow destination processor must at least spend eleven units of time to receive the message. As a result, the lower bound is at least 12 Figure 6 shows that the total time of FNF is no more than twice that of the lower bound in our experiments.

From our experiments, we observed that it is almost impossible to find a single case from 200 times of random-selection that gives a better broadcast time than

the FNF algorithm. In addition, the broadcast time of the FNF algorithm might be very close to optimal since our lower bound estimate is very rough. These timing results also indicate that the completion time grows very slowly when the size of the cluster increases, even when the cluster has up to 100 processors. Our experimental results are consistent with those obtained by previous theoretical sections. In addition, the FNF schedule is very easy to compute and efficient to use.

6 Conclusion

FNF is a very useful technique in reducing broadcast time. In a previous paper we show that FNF gives a broadcast schedule at most twice that of the optimal time for the sender-only communication model[19]. For a more realistic sender-receiver model adapted by this paper, we show that FNF gives a broadcast schedule at most twice that of the optimal time plus a constant. This improves over the previous bound by a performance ratio factor. In practice this factor is bounded by 1.85 [17], but could be unbounded theoretically.

We also describe the experimental results in which we compare the completion time of our heuristics (FNF) with a random-selection algorithm. The experimental results indicate that FNF outperforms the random-selection algorithm by a factor of 2 in average. In addition, we also compare the timing results of FNF with a very roughly estimated lower bound, and FNF always gives a total broadcast time within twice of the lower bound.

There are many research issues open for investigation. For example, it will be interesting to extend this technique to other communication protocols, including reduction and all-to-all communication. For example, we showed that for reduction there is a technique called "slowest-node-first" [20] that also guarantees 2-competitiveness in sender-only model. It would be interesting to extend the result to the sender-receiver model, as we did for broadcasting in this paper. 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 broadcasting. 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.

References

1. T. Anderson, D. Culler, and D. Patterson. A case for networks of workstations (now). In *IEEE Micro*, Feb 1995.
2. M. Banikazemi, V. Moorthy, and D.K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of International Parallel Processing Conference*, 1998.
3. M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda, and P. Sadayappan. Communication modeling of heterogenous networks of workstations for performance characterization of collective operations. In *Proceedings of International Workshop on Heterogeneous Computing*, 1999.

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. Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.
10. M. R. Garey and D. S. Johnson. *Computer and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman, 1979.
11. L. Gargang and U. Vaccaro. On the construction of minimal broadcast networks. *Network*, 19, 1989.
12. M. Grigni and D. Peleg. Tight bounds on minimum broadcast networks. *SIAM J. Discrete Math.*, 4, 1991.
13. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.
14. S. M. Hedetniemi, S. T. Hedetniem, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks.*, 18, 1991.
15. 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.
16. 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.
17. R. Libeskind-Hadas and J. Hartline. Efficient multicast in heterogeneous networks of wrokstations. In *Proceedings of 2000 International Workshop on Parallel Processing*, 2000.
18. A. L. Liestman and J. G. Peters. Broadcast networks of bounded degree. *SIAM J. Discrete Math.*, 1, 1988.
19. P. Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42, 2002.
20. P. Liu and D. Wang. Reduction optimization in heterogeneous cluster environments. In *Proceedings of the International Parallel and Distributed Processing Symposium*, 2000.
21. D. Richards and A. L. Liestman. Generalization of broadcast and gossiping. *Networks*, 18, 1988.
22. J.A. Ventura and X. Weng. A new method for constructing minimal broadcast networks. *Networks*, 23, 1993.
23. D. B. West. A class of solutions to the gossip problem. *Discrete Math.*, 39, 1992.