Approximation Algorithms for Broadcast and Reduction Scheduling in Synchronous Heterogeneous Clusters

Ching-Lun Lin  Pangfeng Liu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

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. 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.

The communication among the processors within a cluster is one of the key factors of the overall parallel efficiency. This paper focuses on the optimization of the two most important collective communication protocols – broadcast and reduction. We study the optimization problem from a theoretical point of view, propose new algorithms for broadcast and reduction, and prove mathematically that our methods do have performance guarantee.

We first propose a synchronous communication model where the communication cost is determined by both sender and receiver. In this synchronous model both sender and receiver of a message must wait until the current communication finishes. We propose new techniques derived from slowest-node-first (SNF) scheduling, and prove that these approximation algorithms achieve competitive ratio 4. Furthermore, we conduct experiments to demonstrate that our broadcast and reduction schedules are very close to the optimal solutions without extensive computation time.

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) with 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 to 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, and 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 message. In a heterogeneous environment it is no longer clear how we should proceed to complete the same task.

A simple heuristic called fastest-node-first (FNF), introduced by Banikazemi et al. [2], is a very efficient broadcast protocol for heterogeneous cluster systems. Liu [19] showed that FNF is an approximation algorithm with a competitive ratio 2 for a communication model where the communication cost is determined by the sender only. Liu and Wang [20] showed that another simple heuristic called slowest-node-first (SNF) is an approximation algorithm of competitive ratio 2 for reduction in the same sender-only model. Liu, Wang, and Guo [21] showed that FNF broadcast protocol can also apply to a communication model where the communication cost is determined by both sender and receiver. However, the communication model in this case does not synchronize the sender and the receiver processor, that is, the sender does not wait for the current communication to complete before it could start sending the next message.

This paper introduces a synchronous communication model where the communication cost is determined by both sender and receiver. In this synchronous sender-receiver model, we synchronize the sending and receiving processors. Both sender and receiver must wait until the current communication finishes before engaging in any other communication. It is a very hard optimization problem to find an optimal broadcast or reduction schedule in this sender-receiver model. Nevertheless we show that a technique called SNF tree scheduling that we derived from SNF for reduction, is a very effective protocol in our synchronous sender-receiver model for both broadcast and reduction. In addition, we show that SNF tree scheduling is an approximation algorithm with competitive ratio 4 for both communication protocols. We also conduct experiments to demonstrate that SNF tree scheduling produces schedules that are very close to the optimal solutions, without excessive computation time.

The rest of the paper is organized as follows: Section 2 describes the communication model in our treatment of broadcast problem in heterogeneous clusters of workstations. Section 3 gives the SNF tree algorithm. Section 4 gives the theoretical results for the problems. Section 5 presents the experimental results and Section 6 concludes with our experience in this theoretical study and other interesting open problems.

2 Communication Model

There have been two classes of models for collective communication in heterogeneous 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 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 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, 22, 23, 24].

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 in 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 \( \log k \) 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 showed 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 was reported in [20], and the fastest-node-first technique was 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 overhead 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 was proven (Libeskind-Hadas and Hartline [17]) to have a total time of no more than \( 2\alpha T^{+} + \beta \), where \( \alpha \) is the maximum ratio between receiving and sending time, \( \beta \) is the maximum difference between two receiving time, and \( T^{+} \) is the optimal time. This bound is later improved by Liu, Wang and Guo to \( 2T^{+} / \beta \) [21]. Other models for heterogeneous clusters include [6, 16].

We adopt the sender-receiver model in [17, 21] and make the following modification: We synchronize sender and receiver for every communication so that both of them must wait until the current communication to finish before engaging in other communications. We adopt this restriction since the message-passing could be mission critical or even transaction-based, and without the confirmation from the receiver any further computation on the sender side will be meaningless. This synchronous behavior is very different from the asynchronous model in [17, 21] and requires different scheduling techniques. The main contribution of this dissertation is to establish an approximation algorithm to perform broadcast in this new model.

2.1 Model Definition

The synchronous communication model of a heterogeneous environment is defined as follows: The system consists of \( n \) proc-
processors $p_0, p_1, \ldots, p_n$, each is capable of direct point-to-point communication to any other processor in the cluster. Each processor is characterized by its speed of sending and receiving messages. Formally, we define the sending time of a processor $p$, denoted by $s(p)$, to be the time it needs to send a unit of message into the network. We also define the receiving time of a processor $p$, denoted by $r(p)$, to be the time it takes to retrieve the message from the network interface. We further assume that the processor speed is consistent and 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 $q$, $s(p) < s(q)$ if and only if $r(p) < r(q)$.

The communication model dictates that the sender and receiver processors cannot engage in multiple message transmissions simultaneously, and we synchronize the transmission between sender and receiver. That is, a sender processor must complete its data transmission to a receiver before sending the next message. This restriction is due to the fact that processor and communication networks have limited bandwidth, therefore we would like to exclude from our model those unrealistic algorithms 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. The model disallows an unrealistic implementation of a reduction operation by having one processor to receive the messages from all the other processors simultaneously. Although many message passing libraries provide non-blocking send and receive primitives in practice, these simultaneous message transmissions are eventually serialized in the hardware level.

In practice the ratio of receiving time to sending time is between 1.05 and 1.85 [17], that is, it is almost equally time-consuming to retrieve a message from the network than preparing it for sending. Thus we make the following assumption on the sending and receiving costs.

**Assumption 1** We assume that the sending cost of a processor is equal to the receiving cost of the same processor, that is, $s(p) = r(p)$.

### 2.2 Reduction Protocol

The reduction problem is defined as follows: Suppose each processor in the system has a unit of information, and we would like to combine all these information into the final answer. Each processor except the final reduction destination sends out only exactly one message. These messages will be combined and forwarded following a tree pattern, and finally received by the destination of the reduction, which is located at the root of the tree. To simplify the discussion, we also make the assumption that the reduction destination is the fastest processor. See Figure 1 for an illustration.

We give an example of reduction example in a cluster of 7 processors. This cluster has one fastest processor $p_0$, two fast processors $p_1$ and $p_2$, three slower processors $p_3, p_4,$ and $p_5$, and one slowest processor $p_6$. The fastest processor has sending and receiving time 2, the fast processors have sending and receiving time 3, the slower processors have sending and receiving time 4, and the slowest processor has sending and receiving time 5, as in Figure 1.

![Figure 1](image1.png)

**Figure 1.** A reduction example in synchronous sender-receiver communication model.

### 2.1 Broadcast Protocol

The broadcast problem is defined as follow: Suppose a processor in the system has a unit of information and we would like to distribute it to all the other processors in the cluster. It is easy to see that each processor except the broadcast source receives exactly one message. These messages will be forwarded following a tree pattern, and finally received by all processors. To simplify the discussion, we make the assumption that the broadcast source is the fastest processor.

Figure 2 illustrates a broadcast schedule with the same cluster we mentioned in Section 2.2. We have one fastest processor, two fast processors, three slower processors, and one slowest processor.

![Figure 2](image2.png)

**Figure 2.** An example of broadcast in synchronous sender-receiver communication model.

### 3 Reduction and Broadcast Algorithms

In this section we will introduce the SNF reduction scheduling in the sender-only model, and derive reduction and broadcast algorithm based on SNF reduction scheduling.

#### 3.1 SNF Sender-Only Reduction Scheduling

Liu and Wang [20] consider a simple scheduling method called slowest-node-first (SNF) for the reduction problem in a sender-only model. We will derive a SNF tree scheduling based on SNF schedule, and show that SNF tree scheduling does have performance guarantee on both the broadcast and reduction problem in synchronous sender-receiver model.

The sender-only communication model for SNF reduction is defined as follows: The heterogeneous cluster consists of $n$ processors $p_0, p_1, \ldots, p_n$, 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. Note that this is a sender-only model so the communication time is determined by the sender only.

Formally we define the scheduling function $s$ so that for any processor $p$ (expect the reduction destination), $p$ starts and completes sending its message at time $s(p)$ and $s(p) + t(p)$, respectively. We also define $c(s(p)) = s(p) + t(p)$ to be the completion time of $p$ under the schedule $s$. 

![Figure 3](image3.png)

**Figure 3.** A reduction example in asynchronous sender-receiver communication model.
Due to the constraint that a processor can only participate a single communication at any given time, we distinguish valid schedule from invalid ones. 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 the schedule \(s\).

\[
\begin{align*}
A(s, t) &= \{ p_i : s(p_i) \leq t < c(s, p_i) \} \\
C(s, t) &= \{ p_i : t \geq c(s, p_i) \} \\
2A(s, t) + C(s, t) &\leq n \tag{3.1}
\end{align*}
\]

A schedule function \(s\) is valid if and only if inequality 3.1 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.

We now describe a technique called earliest possible scheduling that can normalize all the possible valid reduction schedules. An earliest possible (EP) schedule is one in which all the communications are initiated as early 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. 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 3. The algorithm EP assigns non-decreasing start time to the processors in the order they appear in the input processor sequence \(P\).

**Algorithm EP(P) {**

\[
i = 1; \text{time} = 0; \text{free} = n; \\
\text{Active} = \text{empty set}; \text{Complete} = \text{empty set};
\]

while \((i <= n-1)\) {

\[
\text{do while (free} \geq 2) \{
\]

\[
\text{set the start time of the ith processor in P to time;}
\]

\[
\text{add the ith processor in P into Active;}
\]

\[
i = i + 1;
\]

\[
\text{free} = \text{free} - 2;
\]

\[
}\text{Find the set of processors p with the smallest completion time in Active;}
\]

\[
\text{time} = \text{the completion time of p;}
\]

\[
\text{Move p from Active to Complete;}
\]

\[
\text{free} = \text{free} + \text{the number of elements in p;}
\]

**}**

**Figure 3.** The earliest possible scheduling algorithm.

A slowest-node-first (SNF) scheduler for reduction problem sorts the processors in non-increasing speed order, and uses the sorted processor sequence to produce an earliest-possible schedule.

We consider a special class of clusters in which the transmission time of every processor is a power of \(2\). We will use this fact in Theorem 1 to construct an approximation algorithm for our synchronous sender-receiver model.

**Theorem 1** [20] The slowest-node-first method gives an optimal sender-only reduction schedule for all clusters in which the transmission time of every processor is a power of \(2\).

### 3.2 SNF Tree Scheduling for Reduction

Since we can have a valid sender-only schedule by SNF method, the problem now is how to assign a receiver for each communication, and how to prove that the new sender-receiver reduction schedule guarantees good performance. We first show that under any EP schedule, we can assign receivers without violating the constraint that a processor cannot participate in more than one communication simultaneously.

First we establish the dependency among the processors. We define the predecessors of each sender to be the two sender processors that must complete before the new transmission can start. This dependency forms a binary tree among all sender processors, as in Figure 4.

Now we can start to fill in the receivers for all senders, as in Figure 5.

![Figure 4](image-url) The dependency among senders forms a binary tree.

![Figure 5](image-url) The algorithm that decides the receivers in dependency binary tree.

**Observation 1** In a SNF tree reduction schedule every sender \(s\) has larger or equal receiving time than its receiver.

We construct a binary tree as follows: For each transmission we construct a tree node, with a weight equal to the length of the transmission. We place an edge from a node \(a\) to another node \(b\) if the transmission window represented by \(b\) can only occur after the transmission represented by node \(a\) completes. See Figure 6(a) for an illustration.

![Figure 6](image-url) (a) An example for SNF tree reduction schedule in sender-receiver model. (b) An example for SNF tree broadcast schedule in the sender-receiver model.
3.3 SNF Tree Scheduling for Broadcast

It is easy to transform a reduction to a broadcast due to the duality between them. If we traverse back in time in SNF tree schedule, we can have a broadcast schedule, in which the destination of the reduction schedule is the source of the new broadcast schedule.

We formally define this duality as follows: Let \( C(p,q) \) be the cost measurement function for a processor to send a message to another processor \( q \). If \( C(p,q) = s(p) \) we say the model is sender-only. If \( C(p,q) = s(q) \), then the model is receiver-only.

Based on the discussion above we have the SNF tree algorithm for broadcast, as in Figure 7.

1. Find the SNF sender-only reduction schedule in \( C' \), where \( C' = \{(p', s(p')) = s(p), r(p') = r(p) \}, \) for all \( p \) in \( C \).
2. Assign the receiver for each communication according to the dependency in Figure 4.
3. Turn the SNF tree upside down to transform the reduction schedule into a broadcast schedule. (Figure 6(b))

**Observation 2** In a SNF tree broadcast schedule every sender has smaller or equal receiving time than its receiver.

**4 Theoretical Results**

4.1 Competitive Ratio Analysis

We now prove that the reduction time derived from SNF tree scheduling is at most \( 4T \), where \( T \) is the optimal reduction. Before going into the proof, we first analyze the relation between the senders and receivers in a SNF tree scheduling. Observation 1 indicates that the cost of a sender is always no less than the cost of the receiver, as in Figure 8. Based on this observation we can derive a relation between the reduction time in sender-only model and in sender-receiver model respectively.

**Lemma 1** The total time of SNF tree sender-receiver reduction schedule is at most \( 2T \), where \( T \) is the total time of SNF sender-only reduction schedule.

**Proof:**

From Observation 1 we can replace the receiver’s cost with its sender’s cost in each communication without decreasing the total time. We now have a new schedule whose total reduction time is no less than the original SNF tree reduction schedule. The total reduction time of the new schedule is exactly twice the total time of old SNF sender-only reduction schedule since we double the communication time for each transmission.

Figure 8 illustrates an SNF tree reduction schedule. We assume that the total broadcast time is \( T_0 \). Each transmission has two numbers – the upper one is the sending time of the sender and the lower one is the receiving time of the receiver. The number next to a tree node is the time the message-passing starts.

**Theorem 2** The total reduction time of SNF tree scheduling is at most \( 4T \), where \( T \) is the optimal total reduction time.

**Proof:**

Consider a cluster \( C \). We first increase the sending and receiving time to \( 2^{\text{ceiling}(\log p)} \) and \( 2^{\text{ceiling}(\log q)} \) for processor \( p \) in \( C \) and denoted the resulting processor as \( C' \) (Figure 9). We then replace the receiver’s cost with the sender’s cost in each communication. The resulting schedule \( S_I \) will be twice the length of a sender-only schedule \( S_2 \) for a cluster \( C'' \) in which the transmission time of every processor has been increased to a power of 2. The schedule \( S_I \) and cluster \( C'' \) are shown in Figure 10.

Liu and Wang [20] showed that the SNF method gives an optimal reduction schedule for all clusters in which the transmission time of every processor is a power of 2, therefore from Theorem 1 the sender-only reduction schedule \( S_2 \) in \( C'' \) is optimal. Now we change the sender cost of this sender-only schedule back to their original transmission time. (Figure 11) We now have \( T_3 \leq T_2 \), where \( T_3 \) is the optimal reduction time in \( C'' \), and \( T_3 \) is the reduction time in the original cluster \( C \) for the sender-only model.

The power-2 transformation does not increase the total broadcast time by more than a factor of 2, thus the total reduction time of a power-2 cluster is no greater than twice that of the original cluster, and we have \( T_3 \leq 2T^* \), where \( T^* \) is the optimal total time of reduction schedule in sender-receiver model in the original cluster \( C \).

Finally from Lemma 2, the total time of SNF tree sender-receiver reduction schedule is at most \( 2T \), where \( T \) is the total time of SNF sender-only reduction schedule. To sum up, we have \( T_0 \leq 2T_2 \leq 4T^* \) and the theorem follows.
would in a sender-only model. The reduction schedule is then

\[ \text{SNFT} \] by applying SNF, considering only the sending cost as we

5.2 Algorithms

number of classes from 3 to 8 and study the impacts.

market at a very fast pace, the number of processor classes may

number of processor classes. Since faster processors may enter the

are generated as follow: We assume that the number of classes of

capability among different processors. The cluster configurations

focus on the extents of heterogeneity from the communication

scheduling. We also compare the broadcast time obtained from

the reduction times from our SNF tree scheduling and a greedy

5.3 Algorithms

We now describe the algorithms for synchronous broadcast

in the sender-receiver model. We construct SNF tree scheduling

(SNFT) by applying SNF, considering only the sending cost as we

would in a sender-only model. The reduction schedule is then

transformed back into a broadcast schedule with all the original

sending and receiving times.

The greedy algorithm (GR) works as follow: The algorithm

places the set of processors that have not yet received the broadcast

message into an R-set, and the set of processors that have received

the broadcast message into an S-set. We repeatedly match the ear-

liest available sender from the S-set with the minimum receiving

cost receiver from the R-set. We then compute the sum of the

sending time of the sender and the receiving time of the receiver,

and update the available times for both the sender and the receiver.

We place both of them into S-set and repeat this process until R-set

becomes empty.

The enhanced greedy method (EGR) is very similar to the

greedy broadcast algorithm: We first divide the total processors

into two groups – the faster two thirds of the processors and the rest.

We schedule the faster two-thirds of the processors using the

original greedy method. After scheduling those faster processors,

we schedule the rest of the processors, but in the reverse order as

we would in the original greedy method. That is, we pick an earli-

est available sender from the S-set, and match it with the receiver

that has the largest receiving time from the R-set. The intuition is

that by doing so the slowest receiver will start first, so that it could

finish at the same time as other faster receivers, which are sched-

uled to receive messages later than it does.

5.3.1 Fixed Class-size Timing Comparison

We compare the results from SNFT, GR, and EGR with the

optimal broadcast schedule. From Figure 12 we observe that SNFT,

GR, and EGR do not always find the minimum broadcast time.

However, the completion time of all these algorithms is not greater

than twice of the optimal broadcast time. This is consistent with

previous theoretical results. In addition, all these algorithms are

very easy to implement and does not require significant scheduling

time. On the other hand an optimal broadcast schedule is very dif-

cult to find and expensive to compute. We also note that the op-

timal completion time grows very slowly when the size of the

cluster increases, even when the cluster has up to 24 processors.

Figure 12 indicates that among the three algorithms, the en-

hanced greedy method perform the best, then the original greedy

method, then the SNFT method. We believe that both GRs are

actually approximation algorithms with better competitive ratios

than SNFT, and the enhanced method could have a better competi-

tive ratio than the original greedy method.

We observe that the gaps between SNFT, the GR, the EGR,

and the optimal schedule increase when the difference between

transmission costs of the processors within a cluster increases. For

example, the gap between SNFT and the optimal solution and the

gap between GR and the optimal solution in Figure 13 are larger

than those in Figure 12. In a greedy schedule the slowest proces-
sors receives the messages last, therefore the broadcast may wait

for those slowest processors to complete. However, the SNF tree

algorithm could suffer from the additional delays due to the in-

creasing cost of the other processors. As a result, the total broadcast

times of GR and SNFT increase faster than that of the optimal

broadcast schedule when the difference between the communica-

tion overheads increases.

We note that in some cases the completion time fluctuates

when the number of the processors in the clusters increases, espe-
cially when the processors speeds are very different (as in Figure 13). The reason could be that a larger cluster may have a “balanced” schedule so that the completion time is actually smaller than an unbalanced schedule in a smaller cluster. We conclude that the total broadcast time of SNFT, GR, and EGR are all affected by the difference of processor speed.

The completion time of the GR scheduling is always strictly greater than the optimal broadcast time. The reason is that in GR the slowest processors are always the last ones to receive messages, and the OPT never does that from our observation. On the other hand, the completion time of the enhanced greedy scheduling of the processors in the clusters sometimes may be the same as the optimal broadcast time when the speed difference between processors is small. In addition, the difference of transmission time among processors is small, the enhanced greedy scheduling and the OPT all converge into a schedule similar to a binomial tree schedule, which is the OPT for a homogeneous cluster.

Although we assume that sending cost is equal to receiving cost in our theoretical analysis, we still observe SNFT, GR, and EGR all perform well, even when we assume that the sending time is smaller than the receiving time (Figure 14). We make this assumption since in the real world the receiving operation usually incurs a slightly higher overhead than the sending operation. Even under this more practical assumption, we do not see significant derivation in our experiments from the mathematical analysis we had.

![Figure 12](image1.png)

**Figure 12.** The comparison of SNFT, the GR, the EGR, and the OPT. The sending and receiving time of classes of processors are (3,3), (4,4), (5,5), (6,6) on (a), and (2,2), (4,4), (8,8), (16,16) in (b).

![Figure 13](image2.png)

**Figure 13.** The sending time and receiving time of classes of processors are (3,3), (6,6), (9,9), (12,12) on (a), and (3,3), (9,9), (27,27), (81,81) on (b).

![Figure 14](image3.png)

**Figure 14.** The sending receiving time of classes of processors are (2,3), (3,4), (4,5), (5,6) on (a), and (1,3), (2,4), (3,5), (4,6) on (b).

5.3.2 Number of Processor Classes

We compare the results from SNFT, greedy scheduling, the enhanced greedy scheduling, and the optimal broadcast schedule when the cluster has different number of processor classes. From Figure 15 we observe that the completion times of GR and EGR are almost the same, since we only schedule one-third of processors in a different way.

Figure 16 and in Figure 17 give the results when the number of processor classes is large. We observe that the gap between the greedy and the enhanced greedy scheduling increases. That means the EGR does find a much better schedule than greedy, because the “slowest-node-first” approach for the one third of the last processors does improve performance. For example, the slowest processor in greedy scheduling is definitely the last one to receive the message, but the EGR will choose a processor with a smaller transmission cost to be the last one to receive the message. If we have more classes of processors, the EGR may have better candidate processors to choose from.

As we mentioned earlier, the more classes of processors we have, the more gaps of processors are generated. We note that SNFT, EGR, and GR may produce nearly the optimal broadcast time when there are only a few classes of processors.

![Figure 15](image4.png)

**Figure 15.** The sending time and receiving time of classes of processors are (1,1), (2,2), (3,3) in (a), and (1,1), (2,2), (3,3), (4,4) in (b).

![Figure 16](image5.png)

**Figure 16.** The sending time and receiving time of classes of processors are (1,1), (2,2), (3,3), (4,4), (5,5) in (a), and (1,1), (2,2), (3,3), (4,4), (5,5), (6,6) in (b).

![Figure 17](image6.png)

**Figure 17.** The sending time and receiving time of classes of processors are (1,1), (2,2), (3,3), (4,4), (5,5), (6,6), (7,7) in (a), and (1,1), (2,2), (3,3), (4,4), (5,5), (6,6), (7,7) in (b).
6 Conclusion

This paper introduces a realistic synchronous sender-receiver model, in which the characteristics of both sender and receiver are taken into account. In addition, we consider the synchronous behavior between sender and receiver, which is important for transaction-based and mission critical applications.

We show that in synchronous sender-receiver model, a simple technique called SNF tree scheduling produces a broadcast and reduction schedule with a total time at most four times of the optimal schedule. We also describe the experimental results in which we compare the completion time of our SNF tree scheduling, two greedy methods, and the optimal broadcast schedule.

There are many research issues open for investigation. First we would like to improve the competitive ratio. For example, we showed that for broadcast [19] FNF scheduling guarantees 2-competitiveness in sender-only model. Even in a sender-receiver asynchronous model we also show that FNF gives a broadcast schedule at most twice that of the optimal time plus a constant [21]. It would be interesting to extend these results to the sender-receiver synchronization model and improve the competitive ratio.

From our experiments we observed that the both greedy algorithms deliver better performance than our SNF tree scheduling, even though we do not have any guarantee on its performance. It seems very promising that a better competitive ratio than 4 can be derived for the greedy algorithms, since the experimental results indicates that it is a better algorithm in the cases we tried.

It will be interesting to extend our scheduling technique to other communication protocols. For example, it will be worthwhile to investigate the possibility to extend the analysis to similar protocols like parallel prefix, all-to-all broadcasting, or all-to-all reduction. These communication patterns are more complicated and may require totally different treatments. These communication problems are very fundamental in designing collective communication protocols in heterogeneous clusters, and will certainly be the focus of further investigations in this area.

References