Approximation Schemes for Broadcasting in Heterogenous Networks *

Samir Khuller[1], Yoo-Ah Kim[2], and Gerhard Woeginger[3]

[1] Department of Computer Science, University of Maryland, College Park, MD 20742, samir@cs.umd.edu
[2] Department of Computer Science, University of Maryland, College Park, MD 20742, ykim@cs.umd.edu
[3] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, gwoegi@igi.tu-graz.ac.at

Abstract. We study the problem of minimizing the broadcast time for a set of processors in a cluster, where processor p_i has transmission time t_i, which is the time taken to send a message to any other processor in the cluster. Previously, it was shown that the Fastest Node First method (FNF) gives a 1.5 approximate solution. In this paper we show that there is a polynomial time approximation scheme for the problems of broadcasting and multicasting in such a heterogenous cluster.

1 Introduction

Networks of Workstations (NOWs) are an extremely popular alternative to massively parallel machines and are widely used (for example the Condor project at Wisconsin [17]) and the Berkeley NOW project [16]. By simply using off-the-shelf PC's, a very powerful workstation cluster can be created, and this can provide a high amount of parallelism at relatively low cost. Since NOWs are put together over time, the machines tend to have different capabilities and this leads to a *heterogenous* collection of machines, rather than a *homogenous* collection, in which all the machines have identical capabilities.

One fundamental operation that is used in such clusters, is that of *broadcast* (this is a primitive in many message passing systems such as MPI [1, 6, 8]). In addition it is used as a primitive in many parallel algorithms. The main objective of a broadcast operation is to quickly distribute the input data to the entire network for processing. Another situation is when the system is performing a parallel search, then the successful processor needs to inform all other processors that the search has concluded successfully. Various models for heterogenous environments have been proposed in the literature. One general model is the one proposed by Bar-Noy et al [3] where the communication costs between links are not uniform. In addition, the sender may engage in another communication before the current one is complete. An approximation factor with a guarantee

* Research supported by NSF ITR Award CCR-0113192.

of $O(\log k)$ is given for the operation of performing a multicast. Other popular models in the theory literature generally assume an underlying communication graph, with the property that only adjacent nodes in this graph may communicate.

Broadcasting efficiently is an essential operation and many works are devoted to this (see [18, 9, 10, 4, 5] and references therein). In addition, for emergency notification an understanding of how to perform broadcast quickly is essential.

A simple model and algorithm was proposed by Banikazemi et al [2]. In this model, heterogeneity among processors is modeled by a non-uniform speed of the sending processor. A heterogenous cluster is defined as a collection of processors $p_1, p_2, \ldots, p_n$ in which each processor is capable of communicating with any other processor. Each processor has a transmission time which is the time required to send a message to any other processor in the cluster. Thus the time required for the communication is a function of only the sender. Each processor may send messages to other processors in order, and each processor may be receiving only one message at a time.

Thus a broadcast operation is implemented as a broadcast tree. Each node in the tree represents a processor of the cluster. The root of the tree is the source of the original message. The children of a node p_i are the processors that receive the message from p_i. The completion time of a node is the time at which it completes receiving the message from its parent. The completion time of the children of p_i is $c_i + j \cdot t_i$, where c_i is the completion time of p_i, t_i is the transmission time of p_i and j is the child number. In other words, the first child of p_i has a completion time of $c_i + t_i$ ($j = 1$), the second child has a completion time of $c_i + 2t_i$ ($j = 2$) etc. See Figure 1 for an example.

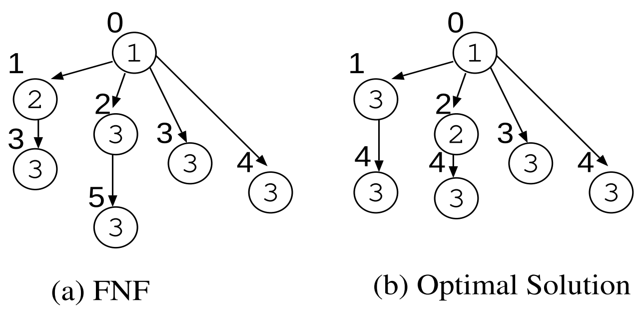

Fig. 1. An example that FNF does not produce an optimal solution. Transmission times of processors are inside the circles. Times at which nodes receive a message are also shown.

A commonly used method to find a broadcast tree is referred to as the "Fastest Node First" (FNF) technique [2]. This works as follows: In each iteration, the algorithm chooses a sender from the set of processors that have received the message (set S) and a receiver from the set of processors that have not yet received the message (set R). The algorithm then picks the sender from $s \in S$ so that s can finish the transmission as early as possible, and chooses the receiver $r \in R$ as the processor with the minimum transmission time in R. Then r is moved from R to S and the algorithm continues. The intuition is that sending the message to fast processors first is a more effective way to propagate the message quickly. This technique is very effective and easy to implement. In practice it works extremely well (using simulations) and in fact frequently finds optimal solutions as well [2]. However, there are situations when this method also fails to find an optimal solution. A simple example is shown in Figure 1.

Despite several non-trivial advances in an understanding of the *fastest node first* method by Liu [13] (see also work by Liu and Sheng [15] in SPAA 2000) it was not well understood as to how this algorithm performs in the worst case. For example, can we show that in all instances the FNF heuristic will find solutions close to optimal?

Liu [13] (see also [15]) shows that if there are only two classes of processors, then FNF produces an optimal solution. In addition, if the transmission time of every slower processor is a multiple of the transmission time of every faster processor, then again the FNF heuristic produces an optimal solution. So for example, if the transmission time of the fastest processor is 1 and the transmission time of all other processors are powers of 2, then the algorithm produces an optimal solution. It immediately follows that by rounding all transmission times to powers of 2 we can obtain a solution using FNF whose cost is at most twice the cost of an optimal solution[4]. However, this still did not explain the fact that this heuristic does much much better in practice. Recently, Khuller and Kim [12] showed that the FNF heuristic actually produces an optimal solution for the problem of minimizing the sum of completion times. This property is used to show that the FNF method has a performance ratio of at most 1.5 when compared to the optimal solution for minimizing broadcast time. In addition the performance ratio of FNF is at least $\frac{25}{22}$. As a corollary of the above approximation result, it is shown that if the transmission times of the fastest $\frac{n}{2}$ processors are in the range $[1 \dots C]$ then FNF produces a solution with makespan at most $T_{OPT} + C$. It is also shown that the problem is NP-hard, so unless $P = NP$ there is no polynomial time algorithm for finding an optimal solution.

It was conjectured in [12] that there is a polynomial time approximation scheme (PTAS) for this problem. In this paper we prove this conjecture. However, this algorithm is not practical due to its high running time, albeit polynomial.

2 Problem Definition

We are given a set of processors $(p_1, p_2, \dots p_n)$ and there is one message to be broadcast to all the processors. Each processor p_i can send a message to another processor with transmission time t_i once it has received the message. Each processor can be either sending a message or receiving a message at any point of time. Without loss of generality, we assume that $t_1 \leq t_2 \leq \dots \leq t_n$ and $t_1 = 1$. Also we assume that p_1 has the message at time zero.

We define the completion time of processor p_i to be the time when p_i has received the message. Our objective is to find a schedule that minimizes $C_{\max} = \max_i c_i$ where c_i is the completion time of processor p_i. In other words, we want to find a schedule that minimizes the time required to send the message to all the processors.

We recall the following definition from [12].

[4] One approach to obtain a PTAS might be rounding to powers of $(1+\epsilon)$. However, this does not work since their proof works only when the completion times of all processors are integers.

Definition 1. *We define the number of* fractional blocks *as the number of (fractional) messages a processor can send by the given time. In other words, given a time T, the number of fractional blocks of processor p_i is $(T - c_i)/t_i$.*

Our proof makes use of the following results from [12].

Theorem 1. *[12] The Fastest Node First algorithm maximizes the total number of fractional blocks for any value T.*

3 Approximation Scheme

We now describe a polynomial time approximation scheme for the problem of performing broadcast in the minimum possible time. Unfortunately, the algorithm has a very high running time when compared to the *fastest node first* heuristic.

We will assume that we know the broadcast time T of the optimal solution. Since $t_1 = 1$, we know that the minimum broadcast time T is between 1 and n, and we can try all possible values of the form $(1+\epsilon)^j$ for some fixed $\epsilon > 0$ and $j = 1 \ldots \lceil \frac{\log n}{\log(1+\epsilon)} \rceil$. In this guessing process we lose a factor of $(1+\epsilon)$.

Let $\epsilon' > 0$ be a fixed constant. We define a set of fast processors F as all processors whose transmission time is at most $\epsilon' T$. Formally, $F = \{p_j | t_j \leq \epsilon' T\}$. Let S be the set of remaining (slow) processors. We partition S into collections of processors of similar transmissions speeds. For $i = 1 \ldots k$, define $S_i = \{p_j | \epsilon' T(1+\epsilon')^{i-1} < t_j \leq \epsilon' T(1+\epsilon')^i\}$ where k is $\lceil \frac{\log(1/\epsilon')}{\log(1+\epsilon')} \rceil$.

We first send messages to processors in F using FNF. We prove that there is a schedule with broadcast time at most $(1 + O(\epsilon))T$ such that all processors in F receive the message first. We then find a schedule for slow processors based on a dynamic programming approach.

Schedule for F: We use the FNF heuristic for the set F to generate a partial broadcast schedule. Assume that the schedule for F has a broadcast time of T_{FNF}. In this schedule every processor $p_j \in F$ becomes idle at some time between $T_{FNF} - t_j$ and T_{FNF}.

We will prove that there is a schedule with broadcast time at most $(1 + O(\epsilon))T$ such that all processors in F receive the message first, and then send it to the slow processors. The following lemma relates T_{FNF} with the time taken by the optimal schedule to propagate the message to *any* $|F|$ processors.

Lemma 1. *In any schedule, we need at least $T_{FNF} - 2\epsilon' T$ time units to have $|F|$ processors receive (any portion of) the message.*

Proof. We prove this by contradiction. In any schedule let C_t be the number of processors that have completely received the message by time t. In addition, let I_t be the number of processors that have started receiving the message by time t. Suppose that at time $T' < T_{FNF} - 2\epsilon' T$, we have $C_{T'} + I_{T'} \geq |F|$. First note that $I_t \leq C_t$ since each processor in I_t is getting the message from exactly one (distinct) processor in C_t. This means we should have that $C_{T'} \geq |F|/2$.

If a schedule is able to complete sending the message to at least $|F|/2$ processors by time T', then the number of fractional blocks of this schedule is at least $|F|/2$. Since FNF maximizes fractional blocks by Theorem 1, we claim that FNF also has at least $|F|/2$ fractional blocks by time T'. Let t_f be the transmission of slowest processor in F. Notice that in additional time $t_f(\leq \epsilon' T)$ all processors that had started receiving the message must have finished receiving it. In additional time $t_f(\leq \epsilon' T)$, FNF most certainly can double the number of processors that have received the message. Thus before T_{FNF}, more than $|F|$ processors would have received the message; a contradiction since T_{FNF} is the earliest time at which the fastest $|F|$ processors receive the message in FNF.

Lemma 2. *There is a schedule in which all processors in F receive the message no later than any processor in S and the makespan of the schedule is at most $(1 + 3\epsilon')T$.*

Proof. The main idea behind the proof is to show that an optimal schedule can be modified to have a certain form. Consider the set of processors of an optimal schedule that have received any portion of the message by time $T_{FNF} - 2\epsilon' T$. This consists of some fast processors, F' and some slow processors S'. Let $F'' = F \setminus F'$ and $S'' = S \setminus S'$. Note that $|F''| \geq |S'|$ since $|F'| + |S'| \leq |F|$ (Lemma 1). In the FNF schedule, by time T_{FNF} all processors in $F = F' \cup F''$ have the message. We can now have the processors in F'' send messages to all processors in S'. Since $|F''| \geq |S'|$ each processor sends only one message and this will take additional time at most $\epsilon' T$. By time $T_{FNF} + \epsilon' T$, all processors in $F' \cup S'$ certainly have the message in addition to the processors in F''. Notice that the optimal schedule now broadcasts the message to all remaining processors in additional time $T - (T_{FNF} - 2\epsilon' T)$. Thus we can also finish the broadcasting in additional time $T - (T_{FNF} - 2\epsilon' T)$. The broadcast time of this schedule is at most $T_{FNF} + \epsilon' T + T - (T_{FNF} - 2\epsilon' T) = (1 + 3\epsilon')T$.

Create all possible trees of S: For the processors in S, we will produce a set $\mathcal{S}$ of labeled trees $\mathcal{T}$. A tree $\mathcal{T}$ is any possible tree with broadcast time at most T consisting of a subset of processors in S. Then we label a node in the tree as i if the corresponding processor belongs to S_i ($i = 1 \dots k$). We prove that the number of different trees is constant.

Lemma 3. *The size of $\mathcal{S}$ is constant for fixed $\epsilon' > 0$.*

Proof. First consider the size of a tree $\mathcal{T}$ (that is, the number of processors in the tree). Let us denote it as $|\mathcal{T}|$. Since the transmission time of processors in S is greater than $\epsilon' T$, we need at least $\epsilon' T$ time units to double the number of processors that received the message. It means that given a processor as a root of the tree, within time T we can have at most $2^{1/\epsilon'}$ processors receive the message. Therfore, $|\mathcal{T}| \leq 2^{1/\epsilon'}$. Now each node in the tree can have different label $i = 1 \dots k$. To obtain an upperbound of the number of different trees, given a tree $\mathcal{T}$ we transform it to a complete binomial tree of size $2^{1/\epsilon'}$ by adding nodes labeled as 0. Then the number of different trees is at most $(k+1)^{2^{1/\epsilon'}}$.

Attach $\mathcal{T}$ to F: Let the completion time of every processor $p_j \in F$ be c_j. Each processor p_j in F sends a message to a processor in S every t_j time unit. Therefore, a fast processor p_j can send messages to at most $X_j = \lfloor \frac{T-c_j}{t_j} \rfloor$ other processors. Let $X = \sum_{p_j \in F} X_j$. Let us consider the time x_i of each sending point in X. We sort those x_i in nondecreasing order and attach a tree from $\mathcal{S}$ to each point (See Figure 2). Note that we can attach at most $|X|$ trees of slow processors. Clearly $|X| \leq n$.

We check if an attachment is feasible, using dynamic programming. Recall that we partition slow processors into a collection of processors $S_1, S_2, \ldots S_k (k = \frac{\log(1/\epsilon')}{\log(1+\epsilon')})$. Let s_i denote the number of processors in set S_i. We define a state $s[j, n_1, n_2, \ldots n_k]$ $(0 \leq j \leq |X|, 0 \leq n_i \leq s_i)$ to be true if there is a set of j trees in $\mathcal{S}$ that we can attach to first j sending points and the corresponding schedule satifies the following two conditions: i) the schedule completes by time T and ii) exactly n_i processors in S_i appear in j trees in total. Our goal is to find out whether $s[j, s_1, s_2, \ldots s_k]$ is true for some j, which means that there is a feasible schedule with makespan at most T. The number of states is at most $O(n^{k+1})$ since we need at most n trees ($|X| \leq n$) and $s_i \leq n$.

Now we prove that each state can be computed in constant time. Given $s[j-1, \ldots]$, we compute $s[j, n_1, n_2, \ldots n_k]$ as follows. We try to attach all possible trees in $\mathcal{S}$ to x_j. Then $s[j, n_1, n_2, \ldots n_k]$ is true if there exists a tree $\mathcal{T}'$ such that the makespan of $\mathcal{T}'$ is at most $T - x_j$ and $s[j-1, n_1 - m_1, n_2 - m_1, \ldots n_k - m_k]$ is true where $\mathcal{T}'$ has m_i slow processors belonging to set S_i. It can be checked in constant time since the size of $\mathcal{S}$ is constant (Lemma 3).

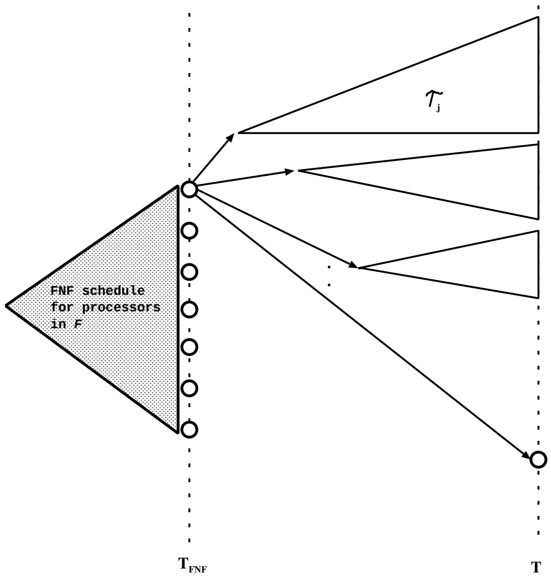

Fig. 2. Attach trees for slow processors to fast processors

Theorem 2. *Given a value T, if a broadcast tree with broadcast time T exists, then the above algorithm will find a broadcast tree with broadcast time at most $(1+\epsilon')(1+3\epsilon')T$.*

Proof. Consider the best schedule among all schedules in which processors in F receive the message first. By Lemma 2, the broadcast time of this schedule is at most $(1+3\epsilon')T$. We round up the transmission time of p_j in S_i to $\epsilon' T(1+\epsilon')^i$ where i is the smallest integer such that $t_j \leq \epsilon'(1+\epsilon')^i T$. By this rounding, we increase the broadcast time by factor of at most $1+\epsilon'$. Therefore, the broadcast time of our schedule is at most $(1+\epsilon')(1+3\epsilon')T$.

Theorem 3. *The algorithm takes as input the transmission times of the n processors, and constants $\epsilon, \epsilon' > 0$. The algorithm finds a broadcast tree with broadcast time at most $(1+\epsilon)(1+\epsilon')(1+3\epsilon')T$ in polynomial time.*

Proof. We try the above algorithm for all possible value of the form $T = (1+\epsilon)^j$ for $j = 1 \ldots \lceil \frac{\log n}{\log(1+\epsilon)} \rceil$. This will increase the broadcast time by factor of at most $1+\epsilon$. Therefore the broadcast time of our schedule is at most $(1+\epsilon)(1+\epsilon')(1+3\epsilon')T$.

For each given value $(1+\epsilon)^j$, we find FNF schedule for processors in F (it takes at most $O(n \log n)$) and attach trees of slow processors to processors in F, using dynamic programming. As we discussed earlier, the number of states is $O(n^{k+1})$ and each state can be checked if it is feasible in $O((k+1)^{2^{1/\epsilon'}})$ time, which is constant. Thus the running time of our algorithm is $O(\lceil \frac{\log n}{\log(1+\epsilon)} \rceil (n \log n + (k+1)^{2^{1/\epsilon'}+1} \cdot n^{k+1})$ where k is $\lceil \frac{\log(1/\epsilon')}{\log(1+\epsilon')} \rceil$.

4 Multicast

A multicast operation involves only a subset of processors. By utilizing fast processors which are not in the multicast group, we can reduce the multicasting time significantly. For example, suppose that we have m processors with transmission time t_1 and m more processors with transmission time t_2 where $t_1 < t_2$. Let we want to multicast a message to all processors with transmission time t_2. If we only use processors in the multicast group, it will take $t_2 \cdot \log m$ time. But if we utilize processors with transmission time t_1, we can finish the multicast in $t_1 \cdot (\log m + 1)$. Therefore, when $t_1 \ll t_2$, the speed-up is significant.

Theorem 4. *We have a polynomial time approximation scheme for multicasting.*

Proof. Note that if an optimal solution utilizes k processors not in the multicast group, then those processors are the k fastest ones. Therefore, if we know how many processors participate in multicasting, we can use our PTAS for broadcasting. By trying all possible k and taking the best one, we have PTAS for multicasting.

5 Acknowledgement:

We thank Maxim Sviridenko for useful discussions.

References

1. *Message Passing Interface Forum*. March 1994.
2. M. Banikazemi, V. Moorthy and D. K. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations. *International Conference on Parallel Processing*, 1998.
3. A. Bar-Noy, S. Guha, J. Naor and B. Schieber. Multicasting in Heterogeneous Networks. *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 448-453, 1998.
4. A. Bar-Noy and S. Kipnis. Designing broadcast algorithms in the Postal Model for Message-passing Systems. *Mathematical Systems Theory*, 27(5), 1994.
5. P. Bhat, C. Raghavendra and V. Prasanna. Efficient Collective Communication in Distributed Heterogeneous Systems. *Proceedings of the International Conference on Distributed Computing Systems*, 1999.
6. 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 Computing*, 40:19–34, 1997.
7. M.R.Garey and D.S.Johnson Computer and Intractability: A Guide to the Theory of NP-completeness, Freeman, New York, 1979.
8. W. Gropp, E. Lusk, N. Doss and A. Skjellum. A High-performance, portable Implementation of the MPI: a Message Passing Interface Standard. *Parallel Computing* 22:789–828, 1996.
9. S.M.Hedetniemi, S.T. Hedetniemi and A.L.Liestman. A Survey of Gossiping and Broadcasting in Communication Networks, *Networks* 18:129–134, 1991.
10. R. Karp, A. Sahay, E. Santos and K.E.Schauser. Optimal Broadcast and Summation in the Logp Model. *Proceedings of 5th Annual Symposium on Parallel Algorithms and Architectures*, pp. 142-153, 1993.
11. R. Kesavan, K. Bondalapati and D. Panda. Multicast on Irregular Switch-based Networks with Wormhole Routing. *Proceedings of the International Symposium on High Performance Computer Architectures*, pp. 48-57, 1997.
12. S. Khuller and Y. Kim. On Broadcasting in Heterogeneous Networks. *Proc. of* 15^{th} *ACM/SIAM Symp. on Discrete Algorithms*, pp. 1004-1013, 2004.
13. P. Liu. Broadcasting Scheduling Optimization for Heterogeneous Cluster Systems. *Journal of Algorithms* 42:135-152, 2002.
14. P. Liu and D. Wang. Reduction Optimization in Heterogeneous Cluster Environments. *Proceedings of the International Parallel and Distributed Processing Symposium* pp. 477-482, 2000.
15. P. Liu and T-H. Sheng. Broadcasting Scheduling Optimization for Heterogeneous Cluster Systems. *ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pp 129–136, 2000.
16. D. A. Patterson, D. E. Culler and T. E. Anderson. A case for NOWs (Networks of Workstations). *IEEE Micro*, 15(1):54–64, 1995.
17. J. Pruyne and M. Livny. Interfacing Condor and PVM to Harness the Cycles of Workstation Clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.
18. D. Richards and A. L. Liestman. Generalization of broadcasting and Gossiping. *Networks* 18:125–138, 1988.