Improving Binomial Trees for Broadcasting in Local Networks of Workstations[1]

Silvia M. Figueira

Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053-0566, USA
sfigueira@scu.edu

Abstract. NOWs (Networks of workstations) have been extensively used to execute parallel applications. Although NOWs seem to be an easy and inexpensive way of obtaining great performance, it may not always be so. When using such an environment for executing a parallel application, performance may not be as good as expected due to delays in communication. Also, the heterogeneity in communication makes it hard to take advantage, or reuse, communication strategies that were useful in regular-topology platforms, e.g., parallel machines or LAN-based clusters of workstations. For instance, broadcasting in a NOW may be more challenging due to the variety of communication links and, consequently, of point-to-point latencies. In this paper, we present a strategy to improve broadcast algorithms that are based on binomial trees, which are used in regular-topology platforms, so that they can execute efficiently in Local NOWs.

1 Introduction

Recent advances in high-speed networks and improved microprocessor performance are making clusters of workstations an appealing vehicle for cost-effective parallel computing. Clusters of workstations are playing a major role in supercomputing [15], being used effectively as parallel machines for large, parallel, scientific applications. With the advent of the Internet, these clusters have been evolving from homogeneous machines connected by a Local Area Network (LAN) to different kinds of machines connected by the Internet. These heterogeneous clusters have been called NOW (Network of Workstations) in the literature. In fact, several groups have developed systems to support the usage of these networks. Some examples are the NOW Project [1] from the University of California, Berkeley and the CONDOR System [6] from the University of Wisconsin.

Although the high-performance computing community has been trying to sell the idea that using machines everywhere is the solution, using local machines may be a

[1]. This research was supported in part by NSF cooperative agreement ACI-9619020 through computing resources provided by the National Partnership for Advanced Computational Infrastructure at the San Diego Supercomputer Center.

more interesting approach in several situations. Usually, researchers have access to machines inside their own institution only. Besides, even when they do have access to machines in other institutions, they will probably have lower priority and higher costs. Also, the performance obtained with local machines only may be higher than the performance obtained when using machines which are separated by WAN (Wide Area Network) links. Another important issue is security. Some researchers may not be willing to have their applications execute outside their own secure network, which sometimes is protected by a firewall. From a different perspective, many institutions may not want foreign applications (applications that belong to other institutions) to execute in their machines, because this will allow potential intruders into their secure environments.

For these reasons, it seems that LNOWs (Local Networks of Workstations), which are formed basically by heterogeneous machines connected by heterogeneous LAN links only, are the perfect environment for distributed parallel applications. In LNOWs, the machines may be directly connected or separated by anything from one to several hops. Administratively, these machines may belong to the same or to cooperating institutions.

Although these LNOWs seem to be an easy and inexpensive way of obtaining great performance, it may not always be so. When using an LNOW for executing a parallel application, performance may not be as good as expected due to delays in communication. LNOWs are connected by heterogeneous links, which may or may not include high-latency low-bandwidth paths. Depending on the application's communication pattern and computation/communication ratio, the delays imposed by these slow paths may affect or not the performance of the application. Also, the machines forming LNOWs may be time-shared, and contention for CPU may need to be factored into performance predictions and scheduling strategies.

The heterogeneity in communication makes it hard to take advantage, or reuse, communication strategies that were useful in regular-topology platforms, e.g., parallel machines or LAN-based clusters of workstations. For instance, broadcasting in an LNOW may be more challenging due to the variety of communication links and, consequently, of point-to-point latencies.

Broadcasting is an important communication strategy that is used in a variety of linear algebra algorithms [9]. In fact, algorithms for broadcasting in different kinds of environments have been discussed extensively. Broadcast algorithms for hypercube machines have been discussed in both [5] and [10]. In [4], the authors presents a study on broadcast algorithms for homogeneous parallel environments. Also, several groups have been working on projects related to broadcasting in heterogeneous NOWs. In [13], the authors present ECO, a packet containing efficient collective operations for these networks of workstations. ECO groups hosts according to the network topology, i.e., each group contains nodes that belong to the same LAN. Based on these groups, ECO implements the broadcast operation using a specific algorithm for each LAN. In [12], the authors also present a solution for more efficient broadcasting in networks of workstations. They also group the hosts according to the network topology, but they use a binomial tree for each LAN. In [2], Banikazemi et al deal with the heterogeneity in the cluster by forming broadcast trees according to the capacity of each machine. In [3], the authors present a communication model of heterogeneous clusters of workstations for performance characterization of collective operations. In [7], the author shows how to improve hypercube broadcast algorithms so that they can execute more efficiently in NOWs.

In this paper, we present a strategy to improve broadcast (i.e., one-to-all) algorithms that are based on binomial trees, which are generally used in regular-topology platforms, so that they can execute efficiently in LNOWs. The strategies discussed do not assume any kind of topology, i.e., the machines may be organized in any topology. It takes into account the distance (or latency) between the nodes to come up with a performance-efficient binomial tree to be used by a broadcast algorithm. We evaluate the efficiency of the strategy proposed by comparing the time to execute broadcast operations using different strategies.

The paper is organized as follows. Section 2 explains how binomial trees can be used for broadcasting in LNOWs. Section 3 shows how to improve binomial trees to make them more efficient for broadcasting in LNOWs. Section 4 presents a representative set of the experiments performed. Section 5 concludes and discusses future work.

2 Using a Binomial Tree for Broadcasting in LNOWs

According to the results presented in [4] and [11], one-to-all operations require different structures depending on the topology of the network. For instance, if the nodes are connected by high-latency links (e.g., WAN links), a flat tree will be the best choice. This happens because by the time a message comes to a distant node, and this node is ready to start forwarding the message, the source node will have had time to start forwarding the message to all the nodes. On the other hand, if the nodes are part of a LAN connected by high-speed links, a binomial tree is the best choice, since a node can start forwarding a message received before the source has time to send the message to all the nodes. In fact, binomial trees have been extensively used for one-to-all operations in homogeneous LAN-based clusters of workstations, in which their wide augmenting structure provides the broadcast operation with parallelism.

These are the extremes, and other options may be necessary when the topology is more complex. For example, in [11], the authors show that, when the network is hierarchical and formed by LAN-based clusters interconnected by WAN links, using binomial trees intra cluster and a flat structure inter cluster is the best solution.

In this paper, we will focus on the case in which the network is formed by nodes connected by LAN links. These nodes may be one or more hops away, and some of them may be connected directly. Since the nodes are connected by LAN links, and there is no specific structure connecting these nodes, the best approach would be to use a binomial tree.

To use a binomial tree for broadcasting in an LNOW, the nodes need to be organized as a binomial tree. In MPICH [14], this is done according to their identification number. For example, in Figure 1, there is an LNOW formed by 8 nodes. These nodes are assigned identification numbers 0 to 7. According to these numbers, a binomial tree can be created, as shown in Figure 2.

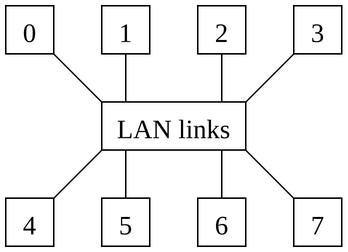

Fig. 1. LNOW formed by 8 nodes connected by LAN links.

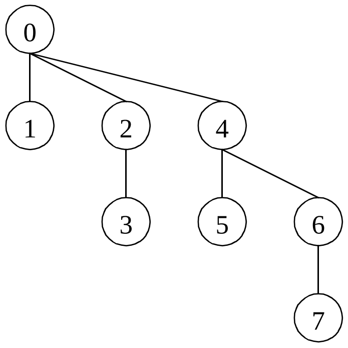

Fig. 2. Binomial tree formed by 8 nodes according to their identification number.

In homogeneous LAN-based clusters of workstations, the nodes are homogeneous and connected by a fast LAN, and the distance (or latency) between each pair of nodes is the same. Therefore, the position of each node in the tree does not affect the performance of the broadcast operation, and forming the tree *blindly*, i.e., independently of the network topology, leads to good performance. However, in LNOWs, due to the heterogeneity in communication, we need a performance-efficient binomial tree, in which the placement of the nodes on the tree reflects the topology of the network. To illustrate, Figure 3 shows the tree obtained according to the identification numbers assigned blindly to the nodes. The number on each edge represents the distance between the respective pair of nodes, which was obtained from Table 1. Distances are in number of hops, and a distance of zero between two nodes means that these two nodes are directly connected.

It is clear that the tree in Figure 3 is not very efficient. For instance, nodes 0 and 3 are close, but they communicate through node 2, which is not as close to either of them. The same happens to nodes 0 and 6, which are also close, but communicate through node 4, which is not close to either of them. This example motivates the need to rearrange the nodes in order to decrease communication costs.

Table 1: Distances between the nodes (in number of hops).

	0	1	2	3	4	5	6	7
0	0	2	2	0	3	3	0	0
1	2	0	0	2	5	5	2	2
2	2	0	0	2	5	5	2	2
3	0	2	2	0	3	3	0	0
4	3	5	5	3	0	0	3	3
5	3	5	5	3	0	0	3	3
6	0	2	2	0	3	3	0	0
7	0	2	2	0	3	3	0	0

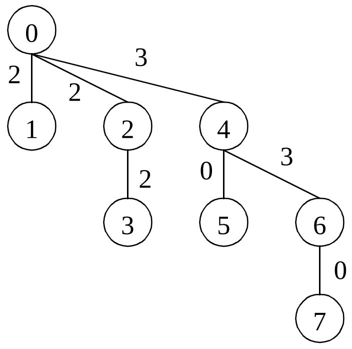

Fig. 3. Binomial tree formed blindly according the nodes' identification number.

3 Improving the Binomial Tree for Broadcasting in LNOWs

Intuitively, a broadcast algorithm would execute more efficiently if we could have communication take place only between nodes that are close together. We define that node *A* is *closer* to node *B* than to node *C*, if the latency between nodes *A* and *B* is lower than the latency between nodes *A* and *C*. Our strategy to reorganize the nodes to form a more performance-efficient low-cost binomial tree is based on the distance (or latency) between the nodes. We try to place nodes that are close together in communicating positions, so that communication in each step of the broadcast operation happens within a short distance. This will decrease communication costs and lower the total cost of the broadcast operation.

In order to compare different strategies, we will use a *cost* measure. The cost of a tree is defined as the cost of the path with the highest cost, and the cost of each path is calculated as the sum of all its edges (i.e., distances) from the root to the respective leaf. For example, the cost for the tree in Figure 3 is 3+3+0=6. This measure assumes that all the paths are covered roughly in parallel, which is true in LNOWs, and it uses the longest path for comparison since this is the path that determines the time to execute a broadcast operation.

Taking the distance into account, we can build an improved binomial tree using different algorithms. The first algorithm, which we call Depth-First, sweeps the binomial tree recursively from right to left, depth first, choosing each child to be the node (from the ones left) that is closest to the parent. The algorithm is shown in Figure 4.

This algorithm tries to place nodes that are closer in the same subtree and gives priority to the right (deepest and fullest) side of the tree. Figure 5 shows the tree obtained using the Depth-First algorithm when applied to the same LNOW described in Table 1.

```
depth-first (p, n)
    >> node n is in position p
    c = number of children in position p
    for i = c - 1 down to 0
        x = new child chosen for node n
        update x's position in the tree
        depth-first (x's position, x)
```

Fig. 4. Depth-First algorithm.

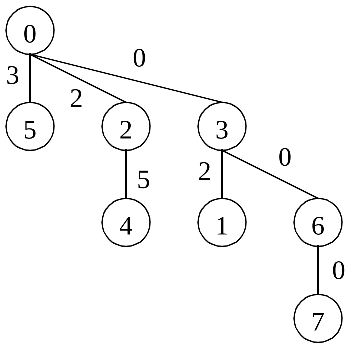

Fig. 5. Binomial tree formed by the Depth-First algorithm, according to the distances shown in Table 1.

This strategy provides a bad result for some topologies, because it may cause some paths to be heavier than others, delaying the entire operation. In Figure 5, for example, the path formed by nodes 0, 2, and 4 is heavier than the others, and the cost of the tree, 2+5=7, is higher than the cost of the tree shown in Figure 3. Intuitively, the best binomial tree for broadcasting in an LNOW should have a balanced total distance on the paths from the root to the leaves. This would lead to a better distribution of communication costs.

The second algorithm, which we call Breadth-First, sweeps the binomial tree from right to left, breadth first, choosing each child to be the node (from the ones left) that is closest to the parent. This strategy tries to balance the paths from the root to the leaves by choosing a child for each path in each level of the tree. The process goes from right to left, giving priority to the longest paths at the right side of the tree. The algorithm is shown in Figure 6. It uses a list, where new children is placed dynamically, to enforce the breadth-first search.

```
breadth-first ()
    >>node 0 is in position 0
    place node 0 in list l
    for each node n in list l
        take node n from list l
        p = position of node n
        c = number of children in position p
        for i = c - 1 down to 0
            x = new child chosen for node n
            update x's position in the tree
            include x in list l
```

Fig. 6. Breadth-First algorithm.

This seems to be a good approach, and it does work in some cases. However, it is also not very robust, in the sense that it fails for many topologies, producing trees that are worse than the one generated blindly. The problem is that this algorithm does not take into account the fact that the paths are not all the same size. Therefore, the total distance in the paths will not be balanced, and there will be heavy paths imposing a long delay on the broadcast process. This solution may work well for *n*-ary trees, where the paths have about the same number of nodes.

Figure 7 shows the tree obtained using the Breadth-First algorithm when applied to the LNOW described in Table 1. The tree generated in this example also contains a heavy path, 0+2+5=7, which compromises the performance of the broadcast operation.

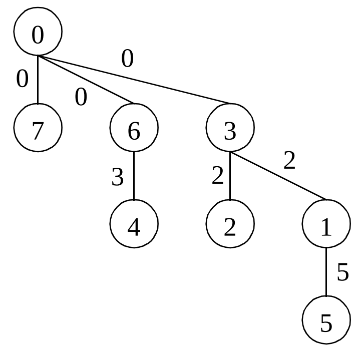

Fig. 7. Binomial tree formed by the Breadth-First algorithm, according to the distances shown in Table 1.

The third algorithm, which we call Balanced-Path, solves the problem with the paths of different sizes. This algorithm sweeps the tree, serving the nodes with the highest number of undefined children first, choosing each child to be the node (from the ones left) that is closest to the parent. The algorithm is shown in Figure 8.

This strategy balances the paths from the root to the leaves by giving priority to the parents that have the largest number of children, since parents with more children belong to longer paths. In a tie, the algorithm goes from right to left, giving priority to the longest paths at the right side of the tree. This algorithm does take into account the fact that the paths are not the same size. Therefore, the total distance in the paths will be balanced, and the communication costs will be distributed roughly evenly. Experiments have shown that this strategy does generate performance-efficient binomial trees.

```
balanced-path ()
    for each node n
        p is the node with the largest number of children
        x = new child chosen for node p
        update x’s position in the tree
```

Fig. 8. Balanced-Path algorithm.

Figure 9 shows the tree obtained using the Balanced-Path algorithm when applied to the LNOW described in Table 1. It is clear that the improved tree will be able to provide a better performance, since the paths are well balanced. The cost of this tree, which is 3, is lower than the cost of the trees shown in Figure 3, Figure 5, and Figure 7.

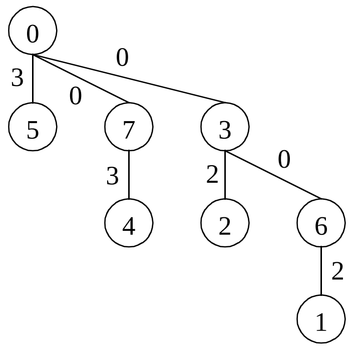

Fig. 9. Binomial tree formed by the Balanced-Path algorithm, according to the distances shown in Table 1.

For an LNOW formed by *p* nodes, the three algorithms presented take $O(p^2)$ steps to generate an improved binomial tree. Since the binomial tree only needs to be updated in case of a significant change in the network load, this cost is not significant in comparison with the gains obtained.

4 Experiments

We have verified the performance obtained with each kind of tree by performing experiments on the IBM SP Blue Horizon at the San Diego Supercomputer Center. The IBM-SP is a LAN-based, regular-topology cluster, in which we have emulated diverse LNOWs by enforcing different latencies between the nodes. Our emulator was implemented using MPI [14] and has the following parameters: the number of nodes in the cluster and the maximum number of groups in the cluster, i.e., the maximum number of groups in which the nodes are directly connected. From these two parameters, the emulator generates random LNOWs, which have various combinations of number of groups, groups' sizes, and latencies among the various groups. The latencies are given in multiples of a basic latency, which is equivalent to the latency between 2 workstations connected by fast Ethernet (100Mbps), i.e., ~35μs. The emulator generates the topologies by generating the distance between each node and node 0. Distances are in number of hops, and the distance between two nodes is calculated as follows. Two nodes that have the same distance to node 0 are in the same group, i.e., they are 0 hops away from each other. For example, if node 1 is 2 hops away from node 0, and node 2 is 2 hops away from node 0, then node 1 is 0 hops away from node 2. The distance between two nodes that belong to different groups is calculated as the sum of the distance between each node and node zero. For example, if node 2 is 2 hops away from node 0, and node 4 is 3 hops away from node 0, then node 2 is 5 hops away from node 4. To characterize the LNOW, the maximum distance between 2 nodes is 20 hops.

Experiments have shown that the time to execute a broadcast operation depends on the cluster topology, and that reorganizing the nodes according to the distance definitely affects the performance obtained. The graphs below present a representative set of our experiments. Each graph presents a set of 4 experiments (i.e., 4 different topologies) and compares the usage of different strategies for each of them. The topology, i.e., the number of groups and the size of the groups, is generated randomly and varies from experiment to experiment. For each experiment, we show the time to execute 10,000 one-to-all operations using, respectively, the regular binomial tree generated blindly (which is actually used by MPICH [14]), a depth-first tree, a breadth-first tree, and a balanced-path tree. We also show the time to execute 10,000 one-to-all operations using a flat structure (in which the source node sends a point-to-point message to each of the other nodes) and a magpie [11] structure (in which the source node sends a message to one node in each group, which forwards the message to the other nodes in the same group using a binomial tree). Note that, even though the magpie structure is not a perfect match for this unstructured environment, since the nodes may or may not be organized in groups, we include its performance for completeness. In our experiments, when the topology presents groups, the magpie algorithm uses a binomial tree for each group, and when the topology does not present groups, the magpie structure is reduced to a flat structure.

Figure 10 shows a set of experiments where the number of nodes is 32 and the maximum number of groups is 8. This means that the 32 nodes are divided into up to 8 groups of nodes. In this case, the flat structure is the best option. Among the algorithms based on binomial trees, the one that uses the balanced-path tree is the best option, corroborating with the study shown in Section 3. Note that, in experiment 2, the depth-first tree achieves a performance that is worse than the performance obtained with the regular binomial tree. In experiment 4, it is the breadth-first tree that achieves a performance that is worse than the performance obtained by the regular binomial tree.

Figure 11 shows a set of experiments where the number of nodes is 64 and the maximum number of groups is 32. This means that the 64 nodes are divided into up to 32 groups of nodes. In this case, the flat structure is just a little better than the balanced-path binomial tree. The performance obtained with the flat structure is sensitive to the number of nodes used. Among the binomial trees, the balanced-path one is still the best option.

Figure 12 shows a set of experiments where the number of nodes is 128 and the maximum number of groups is 8. This means that the 128 nodes are divided into up to 8 groups of nodes. In this scenario, the flat structure is the worst option, and the balanced-path binomial tree is the best one. This set of experiments shows that for a large number of nodes, in this type of environment, flat structures are inefficient.

Figure 13, Figure 14, and Figure 15 show sets of experiments where the number of nodes is 128. In Figure 13, the maximum number of groups is 32. In Figure 14, the maximum number of groups is 64. In Figure 15, the maximum number of groups is 128. The results obtained in each set of experiments are similar, showing that the number of groups does not affect the behavior of the strategies.

From the experiments presented, it is clear that using a strategy to improve the binomial tree is key for broadcast operations to execute efficiently in LNOWs. Generally, the gains provided by the improved trees can be quite high. Among the algorithms based on binomial trees, the one based on a balanced-path tree achieves the best performance. In fact, it always achieves a performance that is better than the performance obtained with the regular blind tree. The performance obtained with the depth-first and the breadth-first trees is, in most cases, comparable to the performance obtained with the regular blind tree.

The flat structure is the best option when using a small number of nodes, but it is inefficient when the number of nodes increases. In fact, for a large number of nodes, the flat structure provides a performance that is much worse than the performance obtained with any of the binomial trees.

The magpie approach, which is the best option for LAN-based clusters interconnected by WAN links (according to [11]), is not a good option in this environment, specially when there are no groups, i.e., when the nodes are scattered. In this case, the magpie structure resembles the flat structure, which provides a poor performance when the number of nodes is large.

From the experiments presented, we can also conclude that, in this kind of environment, the topology of the network and the number of groups in the network do not affect significantly the performance of the algorithms. In fact, the important factor is the number of nodes, which affects the performance obtained with the flat structure.

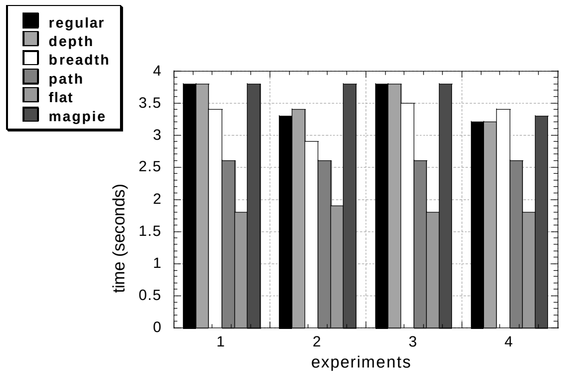

Fig. 10. The graph compares the usage of different strategies, when the number of nodes is 32 and the number of groups in the given topologies is at most 8.

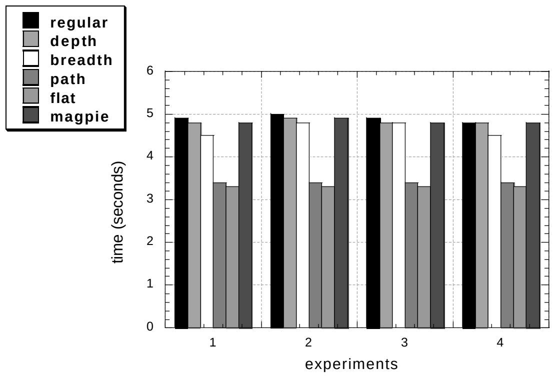

Fig. 11. The graph compares the usage of different strategies, when the number of nodes is 64 and the number of groups in the given topologies is at most 32.

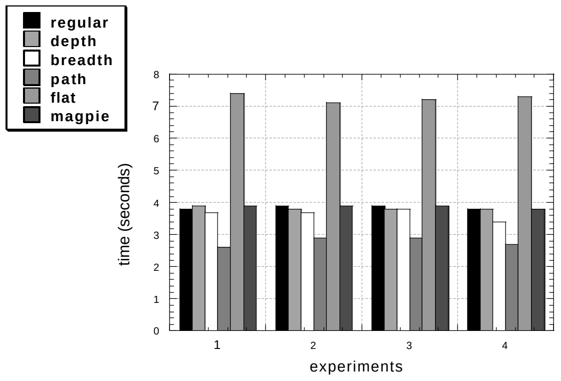

Fig. 12. The graph compares the usage of different strategies, when the number of nodes is 128 and the number of groups in the given topologies is at most 8.

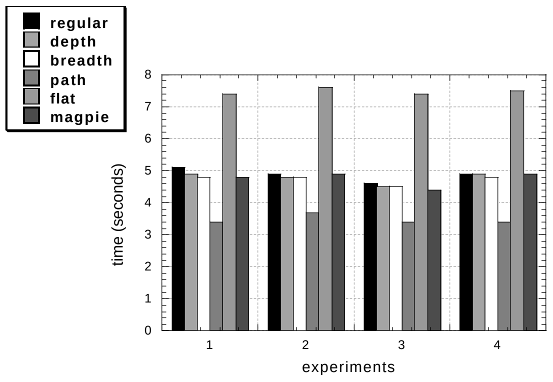

Fig. 13. The graph compares the usage of different strategies, when the number of nodes is 128 and the number of groups in the given topologies is at most 32.

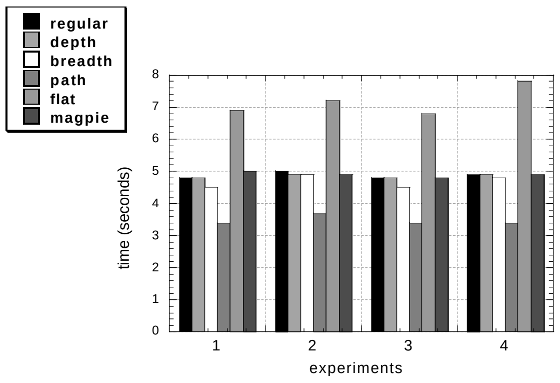

Fig. 14. The graph compares the usage of different strategies, when the number of nodes is 128 and the number of groups in the given topologies is at most 64.

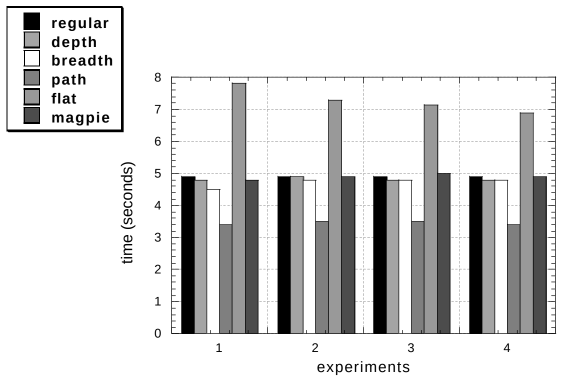

Fig. 15. The graph compares the usage of different strategies, when the number of nodes is 128 and the number of groups in the given topologies is at most 128.

5 Conclusion

Broadcast algorithms are heavily used by scientific applications, and adapting these algorithms to execute efficiently in LNOWs is crucial to improve these applications' performance in these networks.

In this paper, we have presented three algorithms to organize the nodes that form an LNOW into a performance-efficient binomial tree to be used in broadcast operations. These algorithms - Depth-First, Breadth-First, and Balanced-Path - are based on the distance between the nodes in the cluster.

Intuitively, the best binomial tree for broadcasting in an LNOW should have a balanced total distance on the paths from the root to the leaves, since this leads to a better distribution of communication costs. In fact, according to the experiments, the broadcast algorithm based on the balanced-path binomial tree achieves the best performance among the algorithms based on binomial trees. The experiments performed have also shown that using the performance-efficient balanced-path binomial tree is even more important as more nodes are used, since in this case the flat structure provides a poor performance. In summary, the balanced-path binomial tree is an important general option for broadcasting in LNOWs.

Using a performance-enhanced binomial tree is important not only for a broadcast operation's performance, but it also contributes to decrease the traffic on the network, improving the performance of the network as a whole.

In the future, we plan to develop better strategies for other broadcast operations, such as gather, scatter, and barrier, so that they can execute efficiently in LNOWs.

References

1. T. Anderson, D. Culler, D. Patterson, and the NOW team, "A Case for Networks of Workstations: NOW," IEEE Micro, vol. 15, no. 1, pp. 54-64, February 1995.
2. M. Banikazemi, V. Moorthy, and D. K. Panda, "Efficient Collective Communication on Heterogeneous Networks of Workstations," in Proceedings of the International Conference on Parallel Processing, August 1998.
3. M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda, and P. Sadayappan, "Communication Modeling of Heterogeneous Networks of Workstations for Performance Characterization of Collective Operations," in Proceedings of the Heterogeneous Computing Workshop, April 1999.
4. M Bernaschi and G. Iannello, "Collective Communication Operations: Experimental Results vs. Theory," Concurrency: Practice and Experience, vol. 10, no. 5, pp. 359-386, 1998.
5. D. P. Bertsekas, C. Özveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis, "Optimal Communication Algorithms for Hypercubes," in Journal of Parallel and Distributed Computing, vol. 11, pp. 263-275, 1991.
6. A. Bricker, M. Litzkow, and M. Livny, "Condor Technical Summary", Technical Report #1069, University of Wisconsin, Computer Science Department, May 1992.
7. S. M. Figueira, "Using a Hypercube Algorithm for Broadcasting in Internet-Based Clusters," in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), June 2000.

8. I. Foster, “Designing and Building Parallel Programs - Concepts and Tools for Parallel Software Engineering,” Addison Wesley Publishing Company, 1995.
9. D. Gannon and J. Van Rosendale, “On the Impact of Communication Complexity in the Design of Parallel Numerical Algorithms,” IEEE Transactions on Computers, vol. C-33, pp. 1180-1194, December 1984.
10. S. L. Johnsson and C. Ho, “Optimum Broadcasting and Personalized Communication in Hypercubes,” IEEE Transactions on Computers, vol. 38, no. 9, pp. 1249-1268, September 1989.
11. T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang, “MagPIe: MPI’s Collective Communication Operations for Clustered Wide Area Systems,” in Proceedings of the Symposium on Principles and Practice of Parallel Programming, May 1999.
12. T. Kielmann, H. E. Bal, and S. Gorlatch, “Bandwidth-Efficient Collective Communication for Clustered Wide Area Systems,” in Proceedings of the International Parallel and Distributed Processing Symposium, May 2000.
13. B. B. Lowekamp and A. Beguelin, “ECO: Efficient Collective Operations for Communication on Heterogeneous Networks,” in Proceedings of the 10th International Parallel Processing Symposium, April 1996.
14. Message-Passing Interface Forum, “MPI: A Message-Passing Interface Standard,” International Journal of Supercomputing Applications, 8(3/4), 1994.
15. Task Force on Cluster Computing, http://www.dgs.monash.edu.au/~rajkumar/tfcc/.