Using a Hypercube Algorithm for Broadcasting in Internet-Based Clusters

Silvia M. Figueira

Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053-0566

Abstract

Internet-based clusters of workstations have been extensively used to execute parallel applications. Although these internet-based clusters seem to be an easy and inexpensive way of obtaining great performance, it may not always be so. When using such a cluster 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 an internet-based cluster 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 hypercube-based broadcasting algorithms that are used in regular-topology platforms, so that they can execute efficiently in internet-based clusters of workstations.

Keywords: broadcasting, hypercube algorithms, network of workstations, internet-based clusters of workstations.

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 built with commodity hardware and software components are playing a major role in supercomputing [12], 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 LAN (Local Area Network) to different kinds of machines connected by the Internet. In fact, there are several groups working on systems that enable users to "steal" cycles from idle (or seldom used) machines located in other departments or even other institutions. Some examples are the NOW Project [1] from the University of California, Berkeley and the CONDOR System [6] from the University of Wisconsin.

Although these internet-based clusters seem to be an easy and inexpensive way of obtaining great performance, it may not always be so. When using such a cluster for executing a parallel application, performance may not be as good as expected due to delays in communication. Internet-based clusters are connected by heterogeneous links, which may or may not include long-distance 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.

The heterogeneity in communication makes it hard to take advantage, or reuse, communication strategies that are useful in regular-topology platforms, such as parallel machines or LAN-based clusters of workstations. For instance, broadcasting in an internet-based cluster 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 [8]. In fact, algorithms for broadcasting in different kinds of environments have been discussed extensively. Broadcasting algorithms for hypercube machines have been discussed in both [5] and [9]. In [4], the authors presents a study on broadcasting algorithms for homogeneous parallel environments. Also, several

groups have been working on projects related to broadcasting in heterogeneous internet-based clusters of workstations. In [11], the authors present ECO, a packet containing efficient collective operations for these clusters of workstations. ECO groups hosts according to the network topology and implements the broadcasting operation using specific algorithms for different LANs. In [10], the authors also present a solution for more efficient broadcasting in internet-based clusters 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 broadcasting trees according to each machine's capacity. In [3], the authors present a communication model of heterogeneous clusters of workstations for performance characterization of collective operations.

In this paper, we present a strategy to improve hypercube-based broadcasting algorithms, which are used in regular-topology homogeneous platforms, so that they can execute more efficiently in internet-based clusters of workstations. Our strategy is based on organizing the hypercube according to the distance (or latency) between each pair of nodes. This is accomplished by placing nodes that are close together in communicating positions of the hypercube.

This paper is organized as follows. Section 2 discusses the basic hypercube-based broadcasting algorithm used in regular-topology platforms. Section 3 presents a way of improving this algorithm to enhance its performance in internet-based clusters. Section 4 shows experiments executed and results obtained with the improvement proposed. Section 5 concludes and discusses future work.

2 Hypercube-Based Broadcasting Algorithm

For broadcasting, we use the algorithm proposed by Foster [7]. It broadcasts using a hypercube communication template. The algorithm is executed by each task in a hypercube communication structure (obtained by the processes' identifiers). This algorithm allows an operation that requires all-to-all communication among *P* processes to be performed in *logP* steps. The algorithm is presented below:

```
procedure hypercube (myid, input, logp, output)
begin
  state = input
  for i = 0 to logp - 1
    dest = myid XOR 2^i
    send state to dest
    receive message from dest
    state = OP (state, message)
  endfor
  output = state
end
```

Note that *logp* denotes the size of the hypercube, and *myid* represents the node's identifier. XOR denotes an exclusive OR operation, and OP is the user-supplied operator, used to combine local data with data arriving from the *i*th neighbor in the hypercube. In each step of the algorithm, each process exchanges its local *state* (which embeds its local *input* with the information received so far from its neighbors) with one of its neighbors in the hypercube and, then, combines the *message* received from that neighbor with *state* to generate a new *state*.

As shown in [7], this algorithm can be used efficiently, in regular-topology platforms, for vector reductions, matrix transpositions, merge sorts, and so on. However, in internet-based clusters, the algorithm performance will depend on the organization of the hypercube.

3 Broadcasting in Internet-Based Clusters

To use the algorithm presented in Section 2 in non-hypercube platforms, the processes need to be organized in a hypercube. This is done according to the nodes' identifiers, which are usually assigned randomly (independently of any performance measure) to the nodes. For example, in Figure 1, there is a cluster formed by 4 nodes located in the same LAN. These nodes are assigned identifiers 0, 1, 2, and 3. According to these identifiers, a 2D hypercube can be created, as shown in Figure 2, where node 0 is connected to nodes 1 and 2, but not to node 3, which is also connected to nodes 1 and 2.

In regular-topology platforms, organizing the

hypercube randomly may lead to good performance. However, in internet-based clusters, we need a more sophisticated strategy to build the hypercube. Intuitively, the algorithm would execute more efficiently if we could form the hypercube by having 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*.

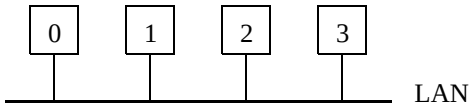

Figure 1: Local Area Network with 4 nodes, identified as 0, 1, 2, and 3.

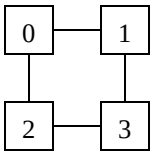

Figure 2: 2D hypercube formed by 4 nodes, according to their identifiers.

Our strategy to reorganize the nodes to form a more performance-efficient hypercube is based on the distance (or latency) between the nodes. The broadcasting algorithm works synchronously and, at each step, each node communicates with a specific node in the same subcube. In this case, placing nodes that are close together in communicating positions, so that communication in each step of the algorithm happens within a short distance, will decrease communication costs and lower the total cost of the broadcasting operation.

To illustrate, consider the cluster presented in Figure 3, where nodes 0 and 3 are part of a LAN, while nodes 1 and 2 are part of another LAN. Suppose the communication link between LAN1 and LAN2 has high latency. If the hypercube is formed according to the nodes' identifiers, it is going to be like the one shown in Figure 2. In this case, all the communication operations will go through the low-latency link between the two LANs. A broadcast operation in a 2D hypercube is done in two steps. In the first step, node 0 exchanges information with node 1, while node 2 exchanges information with node 3. These operations will go through the low-latency link between the 2 LANs. In the second step, node 0 exchanges information with node 2, while node 1 exchanges information with node 3. These operations will also go through the low-latency link between the 2 LANs.

However, if the hypercube is organized according to the nodes' locality, it is going to look like the one shown in Figure 4. In this case, only one step of the algorithm is going to be penalized. In the first step, node 0 exchanges information with node 3, while node 1 exchanges information with node 2. These are intra-LAN communication operations. In the second step, node 0 exchanges information with node 1, while node 2 exchanges information with node 3. These operations will go through the low-latency link between the 2 LANs. It is clear that the hypercube shown in Figure 4 is more performance-efficient than the hypercube shown in Figure 3.

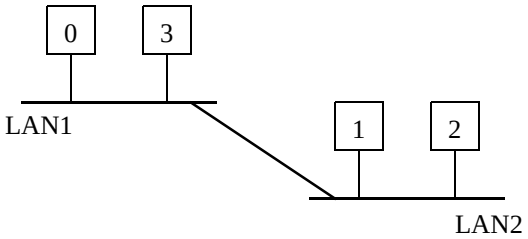

Figure 3: Cluster formed by 4 nodes: 0, 1, 2, and 3. Nodes 0 and 3 are part of LAN1, while nodes 1 and 2 are part of LAN2.

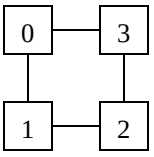

Figure 4: 2D hypercube formed by the nodes in Figure 3, according to the nodes' locations.

To form the low-cost hypercube, we have developed a *form-hypercube* algorithm, which tries to minimize the maximum distance within

each subcube. The maximum distance within a subcube is the maximum distance between all pairs of nodes inside the same subcube. The algorithm starts with a random hypercube, i.e., a hypercube formed according to the nodes' identifiers. Then, it tries to exchange nodes between pairs of subcubes. The exchange takes place if, after exchanging nodes, the maximum distance within both new subcubes is less than the maximum between the maximum distance within each subcube before the exchange. Intuitively, the algorithm tries to equalize the maximum distances within all the subcubes.

The form-hypercube algorithm is presented below:

```
algorithm form-hypercube
begin
  for each subcube i and for each subcube j
    for each node x in i and each node y in j
      if exchanging x and y makes
      max_i < max and max_j < max
        exchange x and y
      endif
    endfor
  endfor
end
```

In the algorithm above, *max* represents the maximum between the maximum distances within subcube *i* and subcube *j* before the exchange. The values max_i and max_j represent the maximum distance within subcube *i* and the maximum distance within subcube *j*, respectively, after the exchange.

The form-hypercube algorithm executes in time $O(nodes^2 * subcubesize^2)$, where *nodes* is the total number of nodes in the hypercube, and *subcubesize* is the size (in number of nodes) of the subcube that the algorithm is trying to minimize. For example, if the algorithm is trying to exchange nodes between 1D subcubes, then *subcubsize* = 2.

4 Experiments

We have performed experiments on the IBM SP at the University of Michigan. The IBM SP is a LAN-based, regular-topology cluster, in which we emulated diverse internet-based clusters by enforcing different latencies between the nodes. Our emulation has the following parameters: the number of nodes in the cluster and the maximum number of *islands* in the cluster, i.e., the number of groups of nodes which are in the same LAN, or close enough to share a very low latency. From these two parameters, the emulator generates random internet-based clusters, which have various combinations of number of islands, islands' sizes, and latencies among the various islands. The latencies are given in multiples of a basic latency.

In each set of experiments, we ran the algorithm 50 times, each with 2,000 all-to-all broadcast operations on 16 nodes. In each experiment, the number of islands, the islands' size, and the latency between the islands vary randomly. The number of islands varies from 1 to 4, and the latency varies from 1 to 4 times 1500usec, which is the latency encountered in 10Mb Ethernet networks.

The experiments performed have shown that the time to execute a broadcast operation depends heavily on the cluster topology, and that reorganizing the nodes according to the distance definitely affects the performance obtained.

Figure 5, Figure 6, and Figure 7 show the times to execute 10 experiments. In each experiment, the topology of the cluster was generated randomly. For each experiment, the first bar represents the time to execute the algorithm when the hypercube was organized randomly, i.e., the assigning of nodes to subcubes was random. The second bar represents, for each experiment, the time to execute the algorithm when the nodes were reorganized by the form-hypercube algorithm, i.e., the hypercube was formed by reordering the nodes in an attempt to place nodes that are close in communicating positions.

In Figure 5, the second bar represents, for each experiment, the time to execute the algorithm when the hypercube was formed by reordering the nodes in an attempt to place nodes that are close in the same 1D subcube. In this set of experiments, the times to execute with a random hypercube vary from 22 seconds to 194 seconds, according to the topology of the cluster. Note that reorganizing the nodes improves the efficiency of the algorithm for all the clusters shown in this set of experiment. In this case, the gain was between 2 and 59%.

In Figure 6, the second bar represents, for

each experiment, the time to execute the algorithm when the hypercube was formed by reordering the nodes in an attempt to place nodes that are close in the same 2D subcube. In this set of experiments, the times to execute with a random hypercube vary from 22 seconds to 142 seconds, according to the topology of the cluster. Note that reorganizing the nodes improves the efficiency of the algorithm for all the clusters shown in this set of experiments as well. In this case, the gain was between 4 and 51%.

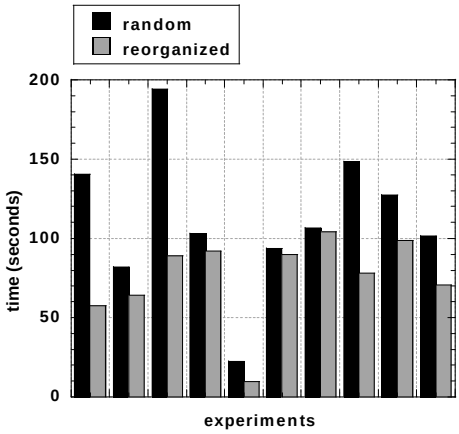

Figure 5: Times to execute the broadcasting algorithm with a hypercube formed randomly and with a hypercube that was reorganized according to 1D subcubes.

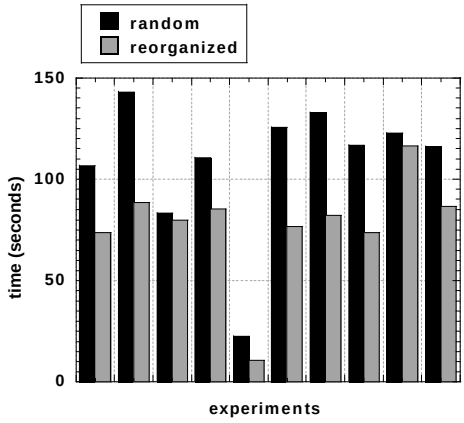

Figure 6: Times to execute the broadcasting algorithm with a hypercube formed randomly and with a hypercube that was reorganized according to 2D subcubes.

In Figure 7, the second bar represents, for each experiment, the time to execute the algorithm when the hypercube was formed by reordering the nodes in an attempt to place nodes that are close in the same 3D subcube. In this set of experiments, the times to execute with a random hypercube vary from 22 seconds to 142 seconds, according to the topology of the cluster. Note that reorganizing the nodes improves the efficiency of the algorithm for all the clusters shown in this set of experiments. In this case, the gain was between 2 and 54%.

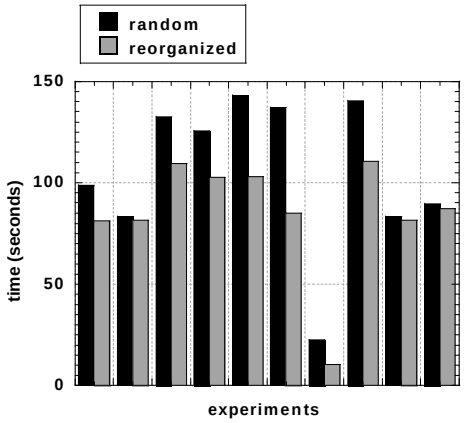

Figure 7: Times to execute the broadcasting algorithm with a hypercube formed randomly and with a hypercube that was reorganized according to 3D subcubes.

One important result obtained from these experiments is the average gain obtained with each hypercube. In the graphs shown above, the average gain for hypercubes formed using 1D and 2D subcubes was 30 and 29%, respectively. The average gain obtained with hypercubes formed using 3D subcubes was 20%. This shows that, for 16-node clusters, it is more effective, on the average, to use smaller subcubes.

Note that, in some cases, the gain obtained by reorganizing the hypercube is very small or none at all. This happens when the random hypercube already has a performance-efficient topology, and no improvement is possible.

Figure 8 shows the times to execute 7 experiments. In each experiment, the topology of the cluster was generated randomly. For each experiment, the graph compares the times to execute the broadcasting algorithm when the hypercube

was reorganized according to subcubes of size 1D, 2D, and 3D. These experiments are representative of a large set of experiments, which have shown that, for 16 nodes, reorganizing the hypercube according to 1D subcubes is the best approach in most of the cases. In some cases, using a 2D subcube works better. However, 3D subcubes were not a good option at all in this setup.

This is an important result, since, as shown before, the form-hypercube algorithm executes in time $O(nodes^2 * subcubesize^2)$. Therefore, for this setup, since 1D subcubes should be used, the time to execute the form-hypercube algorithm is going to be $O(4 * nodes^2)$.

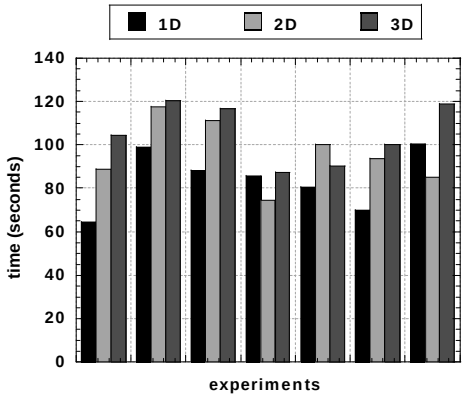

Figure 8: Times to execute the broadcasting algorithm with a hypercube that was reorganized according to 1D, 2D, and 3D subcubes.

For larger clusters, it may be useful to use a double version of the form-hypercube algorithm. In this case, the hypercube would be reorganized according to 1D subcubes. Then, the already improved hypercube would be reorganized again according to 2D hypercubes. Note that this option was not useful with 16 nodes, and we were not able to execute in larger clusters.

5 Conclusion

Broadcasting algorithms are extensively used by scientific applications, and adapting these algorithms to execute efficiently in internet-based clusters is crucial to improve these applications' performance in this kind of platform.

In this paper, we presented a strategy to organize the nodes in an internet-based cluster into a hypercube. The strategy is based on the distance between the nodes in the cluster and is implemented by the form-hypercube algorithm, which exchanges nodes between subcubes trying to decrease the distance between communicating nodes in the hypercube. The experiments performed have shown that decreasing these distances leads to lower communication costs and, consequently, to better performance of the broadcasting algorithm.

We plan to extend other broadcasting algorithms, such as the binomial tree, to improve their efficiency when executing on internet-based clusters. We also plan to study the effects that different clusters, with different topologies, have on the performance of specific broadcasting algorithms.

Acknowledgements

The experiments presented in this paper were performed at the IBM SP at the University of Michigan. Time at the IBM SP has been provided by a grant from NPACI.

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] I. Foster, “Designing and Building Parallel Programs - Concepts and Tools for Parallel Software Engineering,” Addison Wesley Publishing Company, 1995.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Task Force on Cluster Computing, http://www.dgs.monash.edu.au/~rajkumar/tfcc/.