Application-Oriented Adaptive MPI_Bcast for Grids *

Rakhi Gupta, and Sathish Vadhiyar

Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore - 560012
India
{rakhi@rishi.serc, vss@serc}.iisc.ernet.in

Abstract

Due to the importance of collective communications in scientific parallel applications, many strategies have been devised for optimizing collective communications for different kinds of parallel environments. Recently, there has been an increasing interest to evolve efficient broadcast algorithms for computational Grids. In this paper, we present application-oriented adaptive techniques that take into account recent resource characteristics as well as the application's usage of broadcasts for deriving efficient broadcast trees. In particular, we consider two broadcast parameters used in the application, namely, the broadcast message sizes and the time interval between the broadcasts. The results indicate that our adaptive strategies can provide 20% average improvement in performance over the popular MPICH-G2's MPI_Bcast implementation for loaded network conditions.

1 Introduction

It is well known that collective communications play important roles in the performance of parallel applications. Hence, various projects [2, 5, 7, 8, 12] have focused on optimizations of collective communications for various kinds of parallel computing environments including homogeneous LAN settings, heterogeneous networks and more recently Grid systems.

Optimization of collective communications essentially involves 3 main components: 1. obtaining information about resource characteristics, 2. building models that use the resource information to estimate performance of collective communication trees or algorithms(The performance metric that is most commonly used is completion time of the collective operation.), and 3. developing heuristics or search space methods that repeatedly invoke the models to determine "optimal" collective communication algorithms for a given set of resource characteristics and collective communication parameters. Almost all optimization strategies for homogeneous LAN networks and heterogeneous networks perform the first phase during installation of the collective communication libraries [1,3,7,12]. Thus they assume that the network performance characteristics remain stable most of the times. Some efforts also perform the other phases during installation and continue to utilize the same communication algorithms for all applications until the physical network topologies and connections are changed [7, 12].

Clearly, the above mentioned strategies are not suitable for long running parallel applications executing on dynamic Grid environments where optimal collective communication trees vary frequently due to frequent variation of network performance characteristics. Recently, various techniques have been developed for deriving efficient collective communication trees for Grid networks. Some strategies consider only the network topology information for optimal trees and are not adaptive to changing network loads [2]. Heuristic techniques have also been developed to determine efficient broadcast trees for Grid systems [8,9,13]. These heuristics take into account dynamic network conditions with the help of network monitoring tools like NWS [14]. But the communication trees(schedule) for collective communications generated by these heuristics are only theoretically near-optimal and can potentially lead to high network contentions due to simultaneous communications on practical Grid networks. Hence the current efforts do not provide adequate adaptive solutions

*This work is supported by Indian Institute of Science's 10^{th} Plan Grant SERC Part(2A) Special Grant (45/SERC)

1-4244-0054-6/06/$20.00

for intensive parallel applications on Grid systems.

In this paper, we present a comprehensive set of adaptive techniques for determining efficient broadcast trees for long running MPI parallel applications executing on computational Grids. In our system, we periodically gather network performance information. The network information is utilized by simple communication models and simulated annealing based search space techniques to derive efficient broadcast trees. Our simulated annealing based algorithm can be dynamically tuned based on the parameters of broadcasts that were used so far in the application. The broadcast parameters that are of particular interest for our adaptive strategies are message sizes and the time interval between broadcasts. Based on these dynamic parameters, our system "tries its maximum best" to derive an efficient broadcast tree by the time the application invokes the next broadcast call. Thus our strategies are dynamic and adaptive for all the 3 phases of collective communication optimizations. Based on the experiments conducted on Microgrid emulation network [6, 10], we find that our adaptive strategies provide 20% average performance improvement over some of the well known strategies in loaded network conditions.

In Section 2, we describe in detail the adaptive techniques used in our system for determining efficient broadcast trees. Section 3 explains our emulation settings and present results comparing our strategies with the MPICH-G2's implementation of broadcast. In Section 4, relevant work is presented. Section 5 gives conclusions and Section 6 details future work.

2 Methodology

In this section, we discuss the impact of network load on the efficiency of broadcast algorithms, describe our simulated annealing procedure for generating efficient broadcast trees and explain implementation issues in our adaptive strategies.

2.1 Motivation for the problem

The efficient broadcast tree for a given message size depends on the network loading conditions and hence the available capacities of the links in the Grid and the message size used for broadcast and network loading conditions. The network loads on shared links frequently vary in Grid environments. Hence, use of a single broadcast tree for a given message size throughout the entire duration of a long running parallel application, as is practiced in MPICH-G2 [2], can potentially result in poor performance of the application as shown in the previous efforts [8, 9, 13].

Some heuristic techniques that have been developed recently [8, 9, 13] try to derive near-optimal broadcast tress given various message and network parameters using communication models to predict broadcast completion times. For example, the extended single source shortest path algorithm and vertex labeling by Mateescu [8] follows incremental construction of broadcast trees given various parameters including message size, time needed for message initializations, and latencies and bandwidths of the links. The algorithm also uses a communication model based on these parameters to predict broadcast completion times. Although the generated broadcast trees are theoretically attractive in terms of broadcast completion times predicted by the communication model, the communication schedules implied by the broadcast trees can potentially cause high network contentions on practical Grid networks thereby leading to very large actual broadcast completion times. This is because two distinct communication edges of a broadcast tree derived by Mateescu's algorithm can involve the same communication resources in the actual network.

Even though MPICH-G2's broadcast uses information regarding multiple levels of network topology and does not allow more than one communication between clusters in two different sites, the network topology, clusters and sites are statically defined by the user before the application execution. Heuristic algorithms, for e.g. Mateescu's algorithm, takes into account dynamic network characteristics including the latencies and bandwidths of the links for generating broadcast trees but the trees can result in multiple communications between two clusters even if the two clusters are separated by WAN links thus resulting in very high actual broadcast completion times. The adaptive set of procedures discussed in this paper tries to address these deficiencies. Our method generates broadcast trees based on dynamic network characteristics and involves only one communication between the clusters. In our work, a cluster/pool consisting of a set of nodes is not statically defined but dynamically determined based on Grid dynamics. The nodes in a pool formed by our adaptive method are connected to each other by links whose bandwidths are within a certain range. The number of pools, and the sizes and structures of the pools change as the bandwidths of the links change. Hence our method is adaptive to Grid load and network dynamics.

2.2 Simulated Annealing for Efficient Broadcast Trees

At the core of our adaptive method is a simulated annealing algorithm, *basicSA*, that takes as inputs, message size for the broadcast, root of the broadcast, a set of nodes, initialization costs of the nodes, latencies and bandwidths of the links between the nodes, an initial tree and a parameter called *good_list*. In our simulated annealing formulation, the initial temperature is 100 and the minimum temperature is 0. The temperature is decreased by a factor of 0.8 at each step. At each temperature, *basicSA* performs a maximum of 80 tree transformations, evaluating the cost of each generated tree. The tree with the lowest estimated cost at the end of the annealing algorithm is chosen and output as the final broadcast tree. For transforming a tree, a random node on the path of the tree leading to the leaf node with the longest estimated broadcast completion time is chosen and made the child of another random node in the tree. In order to estimate the cost of broadcast for a given tree, we use the broadcast communication model by Mateescu [8] taking into account latencies and bandwidths of the links and initiation costs in each node for a given message size. We use parametrized-LogP benchmark [4] to measure initiation costs and our own communication benchmark program to measure latencies and bandwidths [1]. The *good_list* is an input and output parameter consisting of a list of those broadcast trees whose estimated broadcast completion times are within 10% of the minimum broadcast completion time corresponding to the best broadcast tree. The list is updated by *basicSA* after each tree transformation. The algorithm of *basicSA* is given in Figure 1.

2.3 Multiple Invocations of Simulated Annealing

The *basicSA* is in turn invoked repeatedly by a driver routine, *driverSA*. The algorithm of *driverSA* is given in Figure 2. *driverSA* takes as inputs a time duration, broadcast message size, root of the broadcast, a set of nodes, initialization costs of the nodes and the latencies and bandwidths of the links. The time duration, *time_duration*, denotes the amount of time *driverSA* is allowed to execute. The main purpose of *driverSA* is to produce the "best possible" broadcast tree within the time duration. There are 2 phases in which *driverSA* invokes *basicSA*. In the first phase, *randomization phase*, *driverSA* repeatedly invokes *basicSA*

[1] Currently, there are difficulties in executing NWS on MicroGrid.

```
1  Algorithm:basicSA()
   input  : msg_size, root, initial_tree, init_costs, lat,
            band, good_list
   output: best_tree, best_cost, good_list
2  initialTemp = 100; minTemp = 1;
   TEMP_FACTOR = 0.8; BoltzmanConst = 1.0;
   total_iter_for_temp = 80 ;
3  curTemp = initialTemp ; cur_tree = initial_tree ;
4  cur_cost = broadcast_cost(cur_tree, msg_size,
   init_costs, lat, band) ;
5  best_tree = cur_tree ; best_cost = cur_cost;
6  while curTemp > minTemp do
7      num_accept = 0 ; best_found = 0 ;
8      for iter_at_temp = 1 to total_iter_for_temp do
9          new_tree = transform_tree(cur_tree, root) ;
10         new_cost = broadcast_cost(new_tree,
           msg_size, init_costs, lat, band) ;
11         update_good_list(good_list, new_tree,
           new_cost) ;
12         if new_cost < best_cost then
13             best_tree = new_tree ; best_cost =
               new_cost ; cur_tree = new_tree ;
               cur_cost = new_cost ;
14             num_accept += 1 ; best_found = 1 ;
15         else
16             diff = new_cost - cur_cost ;
17             generate random number, randN ∈
               [0,1];
18             tempN = -diff / (BoltzmanConst×curTemp) ;
19             if randN < exp^tempN then
20                 cur_tree = new_tree ; cur_cost =
                   new_cost ;
21                 num_accept += 1 ;
22             end
23         end
24         if num_accept == 20 then
25             break out of the for loop;
26         end
27     end
28     if num_accept ≤ 3% of total_iter_for_temp &&
       best_found == 0 then
29         break out of while loop ;
30     end
31     curTemp = curTemp × TEMP_FACTOR ;
32 end
33 return (best_tree, best_cost, good_list) ;
```

Figure 1. basicSA()

with different random initial trees and obtains broadcast trees output from *basicSA* (lines 2-19). If the root of the broadcast is contained in the set of nodes, then a random initial tree generated by *driverSA* has its root as the root of the broadcast and only the other parts of the tree are randomly generated. The *driverSA* maintains *good_list*, consisting of good broadcast trees, which it passes to various invocations of *basicSA* during randomization phase (line 5). After certain number of invocations of *basicSA* in the randomization phase, the *driverSA* switches to the second phase, *refinement phase* (lines 20-35). In the refinement phase, *driverSA* cycles through the trees in the *good_list* and repeatedly invokes *basicSA* with the trees in the *good_list* as the initial trees (line 22). If the time duration input to *driverSA* is not specified, the *driverSA* invokes *basicSA* fixed number of times (50 in the randomization phase (lines 8-10) and 25 in the refinement phase (lines 26-28)). If the time duration is specified, the number of *basicSA* invocations are determined dynamically based on the time duration so that the entire set of *basicSA* invocations in the randomization and refinement phases are completed within the time duration. In this case, the time at which the *driverSA* switches from the randomization to refinement phase is determined using a sliding window taking into account the number of iterations for which the best tree corresponding to minimum estimated broadcast time has not changed in the randomization phase and the time duration (lines 12-17).

The transition between the randomization and refinement phases is illustrated in Figure 3 where estimated broadcast times corresponding to efficient broadcast trees determined by simulated annealing invocations are plotted against time progression. The message size used for broadcast was 1 MB. 32 nodes were used for broadcast trees with the bandwidth matrix values between the nodes randomly generated with a uniform distribution between 0.1 and 10 Mbps, values typical for WAN links.

2.4 Division of Nodes into Multiple Pools

With the help of some sample experiments, we compared the quality of the trees generated by the simulated annealing based approach explained above with the trees generated by MPICH-G2's and Mateescu's approaches [2]. We found that our simulated annealing based approach produced trees of best quality when the number of nodes is ≤ 16. For larger number of nodes, the random processes in simulated annealing prevent

[2]The actual results of these sample experiments are not presented due to space constraint

```
1  Algorithm:driverSA()
   input  : msg_size, root, node_set, init_costs, lat,
            band, time_duration
   output: best_tree, best_cost
2  phase = RANDOM; rand_count = 0; good_list =
   empty ;
3  while phase == RANDOM do
4      initial_tree = generateRandomTree(node_set) ;
5      (cur_tree, cur_cost, good_list) =
       basicSA(msg_size, root, initial_tree, init_costs,
       lat, band, good_list) ;
6      rand_count ++ ;
7      if time_duration == null then
8          if rand_count == 50 then
9              break out of while loop ;
10         end
11     else
12         calculate time_elapsed, time elapsed since
           start of driverSA ;
13         time_remaining = time_duration -
           time_elapsed ;
14         last_time = time since best tree in
           good_list has not changed ;
15         if last_time ≥ 1.5×time_remaining then
16             break out of while loop ;
17         end
18     end
19 end
20 phase = REFINE; refine_count = 0; dummy_list
   = empty;
21 while phase == REFINE do
22     initial_tree = next tree in good_list with
       wrap-around ;
23     invoke basicSA on initial_tree ;
24     update best_tree, best_cost; refine_count ++;
25     if time_duration == null then
26         if refine_count == 25 then
27             break out of while loop ;
28         end
29     else
30         calculate time_elapsed, time_remaining ;
31         if time_remaining ≤ 0 then
32             break out of while loop ;
33         end
34     end
35 end
36 return (best_tree, best_cost) ;
```

Figure 2. driverSA()

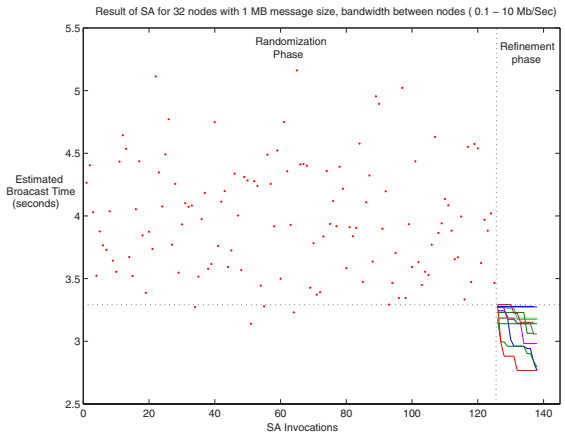

Figure 3. Illustration of randomization and refinement phases

Table 1. Sample bandwidth numbers (Mbps) of a 8-node network

	h1	h2	h3	h4	h5	h6	h7,h8
h1	100	80	80	60	60	20	10
h2	80	100	80	60	60	20	10
h3	80	80	100	60	60	20	10
h4	60	60	60	100	80	40	50
h5	60	60	60	80	100	40	50
h6	20	20	20	40	40	100	30
h7	10	10	10	50	50	30	100
h8	10	10	10	50	50	30	80

the generation of best broadcast trees within short time durations. Hence we have developed an algorithm that partitions a given number of nodes into disjoint clusters/pools of small size and invoke the *driverSA* for each pool or node subsets. This algorithm, *clusteredSA*, partitions a given set of nodes such that the average of communication bandwidths between nodes within a pool is at least equal to a specified percentage of the maximum of all bandwidths between nodes in the original set. In our implementations, we used 4 percentages: 10, 25, 50 and 75 resulting in 4 pool sets. In a Grid network consisting of LAN and WAN links, lower percentages lead to larger pool sizes or smaller number of pools. Table 1 gives bandwidths between 8 nodes in a sample network. For these sample bandwidths, bandwidth percentage of 20 results in the formation of 2 pools ({h1,h2,h3,h4,h5,h6}, {h7,h8}) while bandwidth percentage of 70 results in the formation of 4 pools({h1,h2,h3},{h4,h5},{h6},{h7,h8}).

The *clusteredSA* algorithm is shown in Figure 4. The algorithm receives the same inputs as *driverSA* and returns the best broadcast tree as determined by the algorithm and the corresponding estimated broadcast cost as outputs. The time duration parameter, *time_duration*, denotes the amount of time *clusteredSA* is allowed to execute. The algorithm first tries to discard pool sets which contain pools of sizes greater than 16 nodes (line 11). But if all the 4 pool sets contain pools of larger sizes, then the algorithm proceeds with the 4 pool sets. For each pool set, *clusteresSA* invokes *driverSA* for each pool in the set to form broadcast trees with nodes in the pool (line 17). The roots of the broadcast trees in the pools of the pool set form the nodes for the next level of the algorithm (lines 19,27,28), i.e. the algorithm works with a reduced problem/node size. New pool sets are formed considering only these root nodes and the communication bandwidths between these nodes. *driverSA* is then invoked for the pools in the new pool sets. The steps of formation of pool sets, invoking *driverSA* for the pools, reducing the problem space to consist of only the root nodes of the broadcast trees in the pools and joining these root nodes are repeated (line 28) until all nodes are contained in a single pool in a pool set (lines 4-6). Thus *clusteredSA* generates highly efficient broadcast trees within a tunable time duration for any number of nodes.

2.5 Adaptive Broadcast Architecture: Communication Benchmark, Application, Broadcast Advisor

Our communication benchmark program periodically collects resource information and forms $N \times N$ latency and bandwidth matrices where N represents the number of clusters. The off-diagonal elements in the matrices represent information on inter-cluster links and are obtained by choosing a representative node in each cluster and conducting benchmarking experiments between the representative nodes. The diagonal elements represent intra-cluster communication performance and are obtained by choosing 2 nodes in a cluster and conducting benchmarking experiments between them. Although our benchmarking procedure assumes the existence of clusters, the assumption is reasonable since information about clusters is mostly available in Grid information repositories. The latency and bandwidth matrices are refreshed, i.e. new resource information is collected, every 5 minutes.

```
1  Algorithm:clusteredSA()
   input  : msg_size, root, node_set, init_costs, lat,
            band, time_duration
   output: best_tree, best_cost
   /* Initialization - step 4.  Done only
      during the 1st level of recursion    */
2  Run the algorithm without invoking
   driverSA.Determine the number of pools,
   total_pools, on which driverSA will be invoked. ;
3  new_td = time_duration / total_pools ;
4  if number of nodes in node_set ≤ 16 then
5  |  return output of driverSA on input ;
6  end
7  pSets = empty ;
8  for per ∈ (10,25,50,75) do
9  |  pSets += formPoolSets(node_set, band, per) ;
10 end
11 remove sets with sizes > 16 from pSets ;
12 best_cost = Large; best_tree = null ;
13 for each ps ∈ pSets do
14 |  new_node_set,pTreeList,pCostList = empty;
   |  count = 0;
15 |  for each pool ∈ ps do
16 |  |  (sub_ic, sub_lat, sub_band) = sub elements
   |  |  of (init_costs, lat, band) for nodes in pool ;
17 |  |  (tree, cost) = driverSA(msg_size, root,
   |  |  pool, sub_ic, sub_lat, sub_band, new_td) ;
18 |  |  add (tree, cost) to (pTreeList, pCostList) ;
19 |  |  new_node_set += root of tree ; count ++ ;
20 |  end
21 |  if count == 1 then
22 |  |  (pTree, pCost) = (tree, cost) ;
23 |  |  if pCost < minCost then
24 |  |  |  best_cost = pCost ; best_tree = pTree ;
25 |  |  end
26 |  else
27 |  |  (new_ic, new_lat, new_band) = sub
   |  |  elements of (init_costs, lat, band) for
   |  |  nodes in new_node_set ;
28 |  |  (nTree, nCost) = clusteredSA(msg_size,
   |  |  root, new_node_set, new_lat, new_band,
   |  |  time_duration) ;
29 |  |  combine nTree with pTreeList to form
   |  |  pTree;
30 |  |  use nCost and pCostList to obtain pCost ;
31 |  |  update (best_tree,best_cost) ;
32 |  end
33 end
34 return (best_tree, best_cost) ;
```

Figure 4. clusteredSA()

We use MPI profiling interface for our implementation of MPI_Bcast. When an application calls MPI_Bcast, the time stamp corresponding to the broadcast and the message size used in the broadcast are written to a file, app_file. MPI_Bcast then reads the current best broadcast tree, corresponding to the message size, generated from *clusteredSA* and performs broadcast of the message using the broadcast tree.

The app_file written by the application is continuously monitored by a persistent service, *broadcast advisor*. The broadcast advisor operates in 2 modes. The first mode is when no application is running on the system. In this mode, the advisor cycles through different message sizes from 1 KBytes to 2 MBytes, with increments of power-of-two, in a round-robin fashion and invokes *clusteredSA* for each message size and writes the corresponding broadcast trees to files. In this mode, the time duration parameter needed by *clusteredSA* is passed as empty/null/unspecified value (the reader is referred to decisions based on time duration in *driverSA*).

The second mode is when a parallel application invoking broadcasts is executing on the system. In this mode, the broadcast advisor maintains history of broadcast parameters corresponding to the broadcast calls invoked in the application, and predicts the broadcast parameters corresponding to the next broadcast call in the application. The particular broadcast parameters that are of interest are message sizes used in the broadcasts and the time intervals between broadcast calls. The broadcast advisor reads timestamps and message sizes from app_file and uses simple linear regression models for predictions of next message size and the time gap before next broadcast is called. The advisor then cycles through a set of message sizes close to the predicted message (predicted message size, $\times 2$, $\div 2$) in a round-robin fashion and invokes *clusteredSA* simulated annealing for each message size to determine efficient broadcast trees and writes the trees to files. The predicted time gap before the next broadcast invocation in the application is passed as time duration to *clusteredSA* so that *clusteredSA* can generate the efficient broadcast trees for the predicted message sizes by the time the parallel application invokes its next broadcast call. Thus given the history of broadcast calls in the application, the broadcast advisor "tries its best" to determine the best broadcast tree for the next broadcast invocation. The interactions between the application, broadcast advisor, communication benchmark and *clusteredSA* are shown in Figure 5.

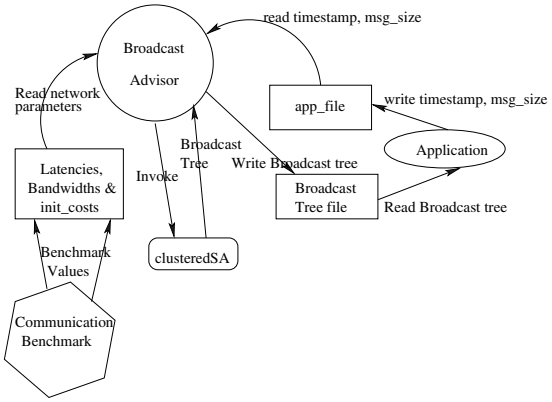

Figure 5. Interactions in broadcast advisor architecture

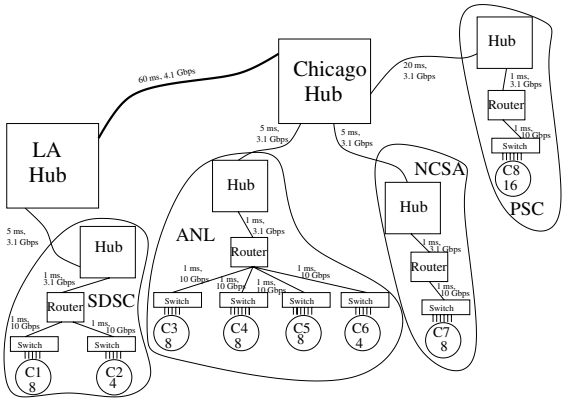

Figure 6. TeraGrid framework

3 Experiments and Results

In this section, we describe our experiment setup and comparisons of our methodologies with MICH-G's and Mateescu's strategies. In order to validate our collective communication optimization strategies, we utilize the Microgrid [6, 10] emulation framework from University of California, San Diego. Microgrid helps to emulate large-scale virtual Grid environments using smaller number of machines. For all our experiments, Microgrid was run on a cluster of 7 Intel Pentium 4 nodes connected by 100 Mbps Ethernet. Each node has a 2.8 GHz processor with 512 MB RAM, 80 GB hard disk and running Fedora Core 2.0 Linux 2.6.5 operating system.

We utilized Microgrid to emulate a simple version of TeraGrid [11] framework. Our simple Teragrid framework consists of 4 sites, namely, San Diego Super Computer Center (SDSC), Argonne National Lab. (ANL), National Center for Supercomputer Applications (NCSA) and Pittsburgh Supercomputer Center (PSC). The framework consists of 8 clusters named **C1** to **C8** distributed across the 4 sites and consisting of 64 nodes as shown in Figure 6. The links in the figure are labeled by latencies and bandwidths of the links. The clusters represented by circles also show the number of nodes in the clusters. In our TeraGrid framework, intra-cluster connections have 100 Mbps bandwidths and the machine speeds are 1.5 GHz. Although our simple version is different from the real TeraGrid network in terms of number of clusters, sites, number of nodes within each cluster and interconnection speeds within clusters, the general framework of our version is similar to the actual TeraGrid in terms of interconnection topology.

Our communication benchmark program runs continuously on 8 representative nodes of the 8 clusters and measures latencies and bandwidths of links between the clusters. The broadcast advisor service is run on cluster **C2** in SDSC. In our experiments, a node in **C2** is chosen as the root of the broadcasts. A network loading program was used to introduce synthetic network loads on the links and to reduce the available bandwidths of those links. The amount and duration of the load can be specified to the loading program. The loading program takes as input a source and destination host. It then continuously sends packets of fixed sizes from the source to the destination thereby reducing the end-to-end bandwidth from the source to the destination host.

In our experiments, parallel application with broadcast calls are executed with MPICH-G2 topology-aware hierarchical broadcast tree, dynamic broadcast trees generated by Mateescu and by our adaptive approach using the same load dynamics and the performance of broadcasts are compared between the three approaches. The MPICH-G2 broadcast tree for the Teragrid framework is shown in Figure 7. The small dark circles denote representative nodes of the clusters. Binomial trees for broadcast communication are used within clusters in MPICH-G2 broadcasts. The open-ended lines from the representative nodes in the figure denote the initiations of binomial broadcasts. For generating Mateescu's broadcast trees, the same architecture shown in Figure 5 is used where the broadcast advisor, instead of invoking *clusteredSA*, invoked Mateescu's algorithm. Both Mateescu's and adaptive approach utilized initiation costs of all 64 nodes, and latencies and bandwidths of all the links (64×64-64) between the nodes to derive broadcast trees containing the 64 nodes.

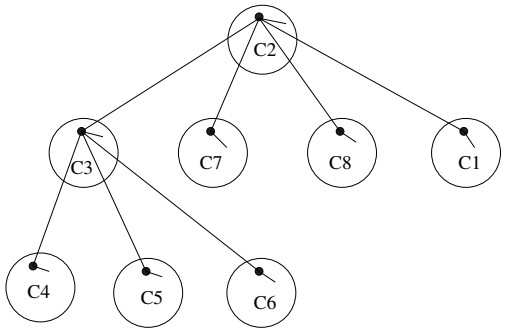

Figure 7. MPICH-G2's broadcast tree for the TeraGrid framework

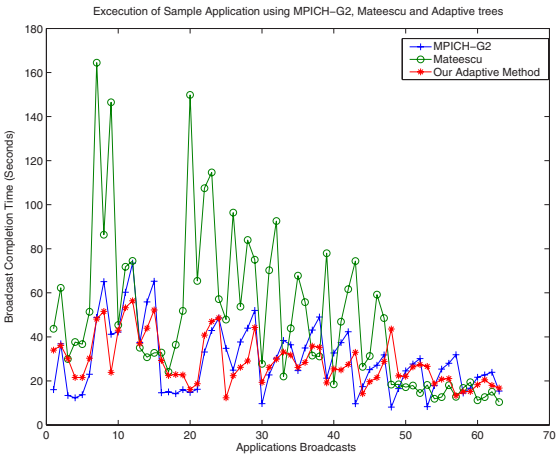

Figure 8. A simulated application scenario

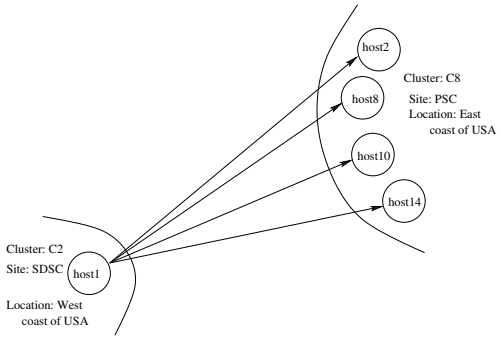

Figure 9. Broadcast tree by Mateescu for the 6^{th} broadcast

In the experiment, we used a synthetic MPI parallel application that invokes about 60 MPI_Bcasts with different message sizes and time intervals between broadcasts. The root initially broadcasts a message of 1 MBytes. The message size is then decreased linearly for subsequent broadcast invocations. The time interval between broadcasts is initially set to 90 seconds and then decremented exponentially for subsequent invocations. This broadcast pattern is usually seen in numerical linear algebra applications, for e.g. right-looking Gaussian elimination, in which sizes of the panels to be factored, the message sizes in broadcasts and the amount of computations between broadcasts decrease with iterations.

While the application was executing, artificial network loads were introduced on different links to/from **C3** cluster at different points of application execution. The links chosen for loading, the amount of the network loads (50%-90% bandwidth reductions), the time durations of the loads (6-8 minutes), time durations between two successive loads (4-7 minutes) were randomly varied with uniform distributions. When the application was executed with Mateescu's and our adaptive broadcasts, the broadcast advisor was run on the same machine as the root of the broadcast. The advisor monitored the broadcast message sizes and the time durations between the broadcasts in the application and invoked either Mateescu's algorithm or *clusteredSA* to determine efficient broadcast trees. These efficient broadcast trees were in turn used by the application for the subsequent broadcasts. Figure 8 shows the broadcast completion times for the different broadcasts used in the application with MPICH-G2's, Mateescu's and our adaptive broadcasts.

The Figure shows that Mateescu's tree consistently performs worse than the other 2 trees. As mentioned earlier, the theoretically near-optimal trees generated by Mateescu's approach lead to network contentions due to simultaneous communications in practical networks and hence result in sub-optimal broadcast performance. Figure 9 shows a part of a broadcast tree derived by the Mateescu's algorithm for the 6^{th} broadcast call. The tree shows the nodes and communication edges involving **C2** cluster in SDSC and **C8** cluster in PSC. In this tree, a number of communication edges exist between the node in **C2** cluster and the nodes in **C8** cluster. Even though these communication edges appear to be distinct, these edges involve large number of common network components including routers, switches and links. The communication edges connecting a cluster in a site with another site involves WAN resources which are typically scarce and shared. This results in contention of resources and hence large broadcast completion times.

As Figure 8 shows, our adaptive techniques performed better than MPICH-G2 broadcasts under

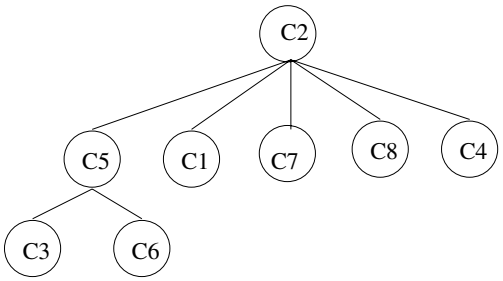

Figure 10. Broadcast tree generated by our adaptive approach for 14^{th} broadcast

loaded conditions (as represented by peaks) and performed worse than MPICH-G2 broadcasts under unloaded conditions (as represented by valleys). Under loaded conditions, the percentage improvement of our adaptive approach over MPICH-G2's approach ranges between 1.47%-64.57% with average performance improvement of 19.58%. During conditions when links to/from **C3** are loaded, use of **C3** as an intermediate node of the hierarchical MPICH-G2 tree shown in Figure 7 presents a major bottleneck in the propagation of broadcast messages to other clusters at the last level and hence significantly delays the completion of the broadcast. Our adaptive techniques automatically detect the load on the links to/from **C3** and generates efficient trees where **C3** is "pushed" to the lowermost level. This leads to performance improvements over the static MPICH-G2 tree. Figure 10 shows the broadcast tree generated by our adaptive approach for the 14^{th} broadcast. In this case, load was applied between a host in C1 and one in C3. As can be seen from the figure, our adaptive method produces an efficient broadcast tree where neither C3 nor C1 is used to propagate messages to other clusters.

In unloaded conditions, the trees generated by our adaptive approach were equivalent to the MPICH-G2's broadcast tree. But the additional overheads in our advisor architecture result in performance degradation of our broadcast. One source of overhead is that in addition to the propagation of the actual message during broadcast, information regarding the dynamic tree should also be propagated to the nodes in our adaptive approach while this information is statically known in MPICH-G2's broadcast. Our current work is on improving the advisor architecture to reduce extra overheads and improve the performance of our adaptive broadcast for unloaded network conditions.

In Figure 8, for the 48^{th} broadcast, we find that the completion time of the adaptive broadcast increases while the completion time of the MPICH-G2's broadcast decreases to a small value. This is due to the classical "stale information" problem where the change in network loads from high to low values is not immediately updated by network monitor for use by our adaptive approach. Hence the tree that was generated in our adaptive approach for high load conditions gives poor performance for unloaded conditions.

4 Related Work

Various research efforts have focused on determining efficient broadcast trees for Grid-scale systems. MagPIe [3] uses parametrized-logP model to determine the completion time of a given broadcast tree. Their model utilizes different parameters that are dependent on network characteristics and message sizes. Although they derive broadcast trees during the run time of the application using hill climbing techniques, the broadcast tree for a particular message size and a particular network remains the same irrespective of the load dynamics.

The comparison of our work with the static MPICH-G2's broadcast strategy is adequately covered in the paper. Heuristic techniques have been developed to build broadcast trees based on dynamic network load conditions on Grids. The work by Vorakosit and Uthayopas [13] uses genetic algorithms to derive optimal broadcast trees. The work by Park et. al. [9] has the same motivations as our work in that it takes into account the dynamic network properties to determine broadcast trees. They use Hierarchical Latency Optimal Tree (HLOT) algorithm that uses dynamic network latency information from NWS to minimize the sum of latencies in the longest path from the root. Recently, the work by Mateescu [8] uses NWS network information, simple communication models, generalized Dijikstra's single-source shortest-path algorithm and node labeling based on post-order traversals to determine efficient broadcast trees. As discussed in the previous sections, Mateescu's algorithm can result in large broadcast completion times in practice. Also, the heuristic techniques are not tunable in terms of their execution times and do not consider application characteristics. Our method considers application's behavior and is suitable for long running parallel applications on dynamic Grid environments.

5 Conclusions

In this paper, we proposed adaptive strategies where the broadcast tree for a message size is varied depend-

ing on resource conditions. Our strategies use lightweight simulated annealing based approach to generate efficient broadcast trees based on the parallel application's usage of broadcasts. Our strategies are compared with the popular MPICH-G2's hierarchical tree approach. Our adaptive strategies exhibited 20% average improvement in performance over MPICH-G2's hierarchical tree approach.

6 Future Work

Although the communication model based on latencies, bandwidths and initiation costs, used in our simulated annealing algorithm can give reasonable predictions, it cannot give accurate estimates due to various factors including Grid topology, various MPI implementation complexities, buffering strategies etc. We plan to include non-intrusive real broadcasts to augment the predictive models in order to improve the efficiency of broadcast trees.

Acknowledgment

The authors would like to thank the MicroGrid team of University of California, San Diego for their immense help with MicroGrid installation and configuration. The authors would also like to thank Prof. S. Keshav and Suihong Liang of University of Waterloo, Canada for providing their code for introducing synthetic network workloads. Also, many tanks to the reviewers whose comments helped in significantly improving the quality of the paper.

References

[1] M. Banikazemi, J. Sampathkumar, S. Prabhu, D. K. Panda, and P. Sadayappan. Communication Modeling of Heterogeneous Networks of Workstations for Performance Characterization of Collective Operations. In *Proceedings of the 8th Heterogeneous Computing Workshop*, pages 125–136, April 1999.

[2] N. T. Karonis, B. R. de Supinski, I. Foster, and W. Gropp. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, pages 377–388, Cancun, Mexico, May 2000.

[3] T. Kielmann, H. E. Bal, and S. Gorlatch. Bandwidth-Efficient Collective Communication for Clustered Wide Area Systems. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, pages 492–499, Cancun, Mexico, May 2000.

[4] T. Kielmann, H. E. Bal, and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *Lecture Notes in Computer Science, Vol. 1800*, pages 1176–1183, 4th Workshop on Runtime Systems for Parallel Programming (RTSPP) held in conjunction with IPDPS 2000, Cancun, Mexico, May 2000.

[5] T. Kielmann, R. F. Hofman, H. E. Bal, A. Plaat, and H. N. R. A.F. Bhoedjang title = MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99)*, pages 131–140, Atlanta, GA, USA, May 1999.

[6] X. Liu and A. Chien. Realistic Large-Scale Online Network Simulation. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Conference on High Performance Networking and Computing*, pages 31–, Pittsburgh, Pennsylvania, USA, November 2004.

[7] B. Lowekamp and A. Beguelin. ECO: Efficient Collective Operations for Communication on Heterogeneous Networks. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*, pages 399–405, Honolulu, HI, USA, April 1996.

[8] G. Mateescu. A Method for MPI Broadcast in Computational Grids. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, page 251b, Denver, Colorado, USA, April 2005.

[9] K. Park, H. Lee, Y. Lee, O. Kwon, S. Park, and S. K. H.W. Park. An Efficient Collective Communication Method for Grid Scale Networks. In *Proceedings of the International Conference on Computational Science*, pages 819–828, Melbourne, Australia and St. Petersburg, Russia, June 2003.

[10] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The Microgrid: a Scientific Tool for Modeling Computational Grids. In *Proceedings of the IEEE/ACM SC2000 Conference*, pages 53–, Dallas, Texas, USA, November 2000.

[11] TeraGrid. `http://www.teragrid.org`.

[12] S. Vadhiyar, G. Fagg, and J. Dongarra. Toward an Accurate Model for Collective Communications. Technical Report ut-cs-05-550, Department of Computer Science, University of Tennessee, U.S.A., 2005.

[13] T. Vorakosit and P. Uthayopas. Generating an Efficient Dynamics Multicast Tree under Grid Environment. In *Proceedings of the 10th European PVM/MPI User's Group Meeting, Lecture Notes in Computer Science*, pages 636–643, Venice, Italy, September 2003. Springer Verlag.

[14] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, October 1999.