Decision Trees and MPI Collective Algorithm Selection Problem

Jelena Pješivac–Grbović Graham E. Fagg Thara Angskun George Bosilca
Jack J. Dongarra
Innovative Computing Laboratory,
The University of Tennessee Computer Science Department
1122 Volunteer Blvd., Knoxville, TN 37996-3450, USA
{*pjesa, fagg, angskun, bosilca, dongarra*}*@cs.utk.edu*

December 12, 2006

Abstract

Selecting the close-to-optimal collective algorithm based on the parameters of the collective call at run time is an important step in achieving good performance of MPI applications. In this paper, we explore the applicability of C4.5 decision trees to the MPI collective algorithm selection problem. We construct C4.5 decision trees from the measured algorithm performance data and analyze the decision tree properties and expected run time performance penalty.

In cases we considered, results show that the C4.5 decision trees can be used to generate a reasonably small and very accurate decision function. For example, the Broadcast decision tree with only 21 leaves was able to achieve a mean performance penalty of 2.08%. Similarly, combining experimental data for Reduce and Broadcast and generating a decision function from the combined decision trees resulted in less than 2.5% relative performance penalty. The results indicate that C4.5 decision trees are applicable to this problem and should be more widely used in this domain.

1 Introduction

The performance of MPI collective operations is crucial for good performance of MPI applications that use them [1]. For this reason, significant efforts have gone into design and implementation of efficient collective algorithms both for homogeneous and heterogeneous cluster environments [2, 3, 4, 5, 6, 7, 8]. Performance of these algorithms varies with the total number of nodes involved in communication, system and network characteristics, size of data being transferred, current load and, if applicable, the operation that is being performed, as well as the segment size which is used for operation pipelining. Thus, selecting the best possible algorithm and segment size combination (*method*) for every instance of collective operation is important.

To ensure good performance of MPI applications, collective operations can be tuned for the particular system. The tuning process often involves detailed profiling of the system, possibly combined with communication modeling, analyzing the collected data, and generating a *decision function*. During run-time, the decision function selects the close-to-optimal method for a particular collective instance. This approach relies on the ability of the decision function to accurately predict

the algorithm and segment size to be used for the particular collective instance. Alternatively, one could construct an in-memory decision system that could be queried/searched at run-time to provide the optimal method information. In order for either of these approaches to be feasible, the memory footprint and the time it takes to make decisions need to be minimal.

This paper studies the applicability of C4.5 decision trees [9] to MPI collective algorithm/method selection problem. We assume that the system of interest has been benchmarked and that detailed performance information exists for each of the available collective communication methods[1]. With this information, we focus our efforts on investigating whether the C4.5 algorithm is a feasible way to generate static decision functions.

The paper proceeds as follows: Section 2 discusses existing approaches to the decision making/algorithm selection problem; Section 3 provides background information on the C4.5 algorithm; Section 4 discusses the mapping of performance measurement data to C4.5 input, Section 5 presents experimental results; Section 6 concludes the paper with discussion of the results and future work.

2 Related work

The MPI collective algorithm selection problem has been addressed in many MPI implementations. In FT-MPI [10], the decision function is generated manually using a visual inspection method augmented with Matlab scripts used for analysis of the experimentally collected performance data. This approach results in a precise, albeit complex, decision function. In the MPICH-2 MPI implementation, the algorithm selection is based on bandwidth and latency requirements of an algorithm, and the switching points are predetermined by the implementers [5]. In the tuned collective module of Open MPI [11], the algorithm selection can be done in either of the following three ways: via a compiled decision function; via user-specified command line flags; or using a rule-based run-length encoding scheme that can be tuned for a particular system.

Another possibility is to view this problem as a data mining task in which the algorithm selection problem is replaced by an equivalent classification problem. The new problem is to classify collective parameters, *(collective operation, communicator size, message size)*, into a correct category, a method in our case, to be used at run time. The major benefit of this approach is that the decision making process is a well-studied topic in engineering and machine learning fields so the literature is readily available. The decision trees are extensively used in pattern recognitions, CAD design, signal processing, medicine, and biology [12].

Vuduc et al. construct statistical learning models to build different decision functions for matrix-matrix multiplication algorithm selection [13]. In their work, they consider three methods for decision function construction: parametric modeling; parametric geometry modeling; and non-parametric geometry modeling. The non-parametric geometry modeling uses statistical learning methods to construct implicit models of the boundaries/switching points between the algorithms based on the actual experimental data. To achieve this, Vuduc et al. use support vector method [14].

Conceptually, the work presented in this paper is close to the non-parametric geometry modeling work done by Vuduc et al. However, our problem domain is different: MPI collective operations instead of matrix-matrix multiplication, and we use the C4.5 algorithm instead of support vector

[1]Detailed benchmarking of all possible methods takes significant amount of time. If this is not an option, performance profiles can be generated using limited set of performance measurements coupled with performance modeling [8].

methods. To the best of our knowledge, we are the only group that has approached the MPI collective tuning process in this way.

3 C4.5 algorithm

C4.5 is a supervised learning classification algorithm used to construct decision trees from the data [9]. C4.5 can be applied to the data that fulfills the following requirements:

- *Attribute-value description*: information about a single entry in the data must be described in terms of attributes. The attribute values can be discrete or continuous, and in some cases, attribute value may be missing or can be ignored;
- *Predefined classes*: the training data has to be divided in predefined classes or categories. This is a standard requirement for supervised learning algorithms;
- *Discrete classes*: the classes must be clearly separated and a single training case either belongs to a class or it does not. C4.5 cannot be used to predict continuous class values such as the cost of a transaction;
- *Sufficient data*: the C4.5 algorithm utilizes an inductive generalization process by searching for patterns in data. For this approach to work, the patterns must be distinguishable from random occurrences. What constitutes the "sufficient" amount of data depends on a particular data set and its attribute and class values, but in general, statistical methods used in C4.5 to generate tests require reasonably large amount of data;
- *"Logical" classification models*: generated classification models must be represented as either decision trees or a set of production rules [9].

In the C4.5 algorithm, the initial decision tree is constructed using a variation of Hunt's method for decision tree construction (Figure 1). The main difference between C4.5 and other similar decision tree building algorithms is in the test selection and evaluation process (last case in Figure 1). The C4.5 utilizes information *gain ratio* criterion, which maximizes normalized information gain by partitioning T in accordance with particular tests [9].

To define the *gain ratio* we have to look at the information conveyed by classified cases. Consider a set T of k training cases. If we select a single case $t \in T$ and decide that it belongs to class C_j, then the probability of this message is $\frac{freq(C_j,T)}{|T|}$ and it conveys $-log_2(\frac{freq(C_j,T)}{|T|})$ bits of information. Then the average amount of information needed to identify the class of a case in set T can be computed as a weighted sum of per-case information amounts [9]:

$$info(T) = -\sum_{j=1}^{k} \frac{freq(C_j,T)}{|T|} \times log_2(\frac{freq(C_j,T)}{|T|}) \quad (1)$$

If the set T was partitioned into n subsets based on outcomes of test X, we can compute a similar information requirement [9]:

$$info_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times info(T_i) \quad (2)$$

Hunt's method for decision tree construction [9]
Given a set of training cases, T, and set of classes $C = \{C_1, C_2, ..., C_k\}$, the tree is constructed recursively by testing for the following cases:
T contains one or more cases which all belong to the same class C_j:
a leaf node is created for T and is denoted to belong to C_j class;
T contains no cases:
a leaf node is created for T but the class it belongs to must be selected from outside source. C4.5 algorithm selects most frequent class at the parent node;
T contains cases that belong to more than one class:
find a test which will split T set to a single-class collections of cases. This test is based on a single attribute value, and is selected such that it results in one or more mutually exclusive outcomes $\{O_1, O_2, ...O_n\}$. The set T is then split into subsets $\{T_1, T_2, ...T_n\}$ such that the set T_i contains all cases in T with outcome O_i. The algorithm is then called recursively on all subsets of T.

Figure 1: Hunt's method for decision tree construction [9].

Then, the information gained by partitioning T in accordance with the test X can be computed as:

$$gain(X) = info(T) - info_X(T) \tag{3}$$

The predecessor to the C4.5 method, the ID3 algorithm, used *gain* criterion in Equation 3 to select the test for partition. However, the *gain* criterion is biased towards the high frequency data. To ameliorate this problem, C4.5 normalizes the information *gain* by the amount of the potential information generated by dividing T into n subsets:

$$split\ info(X) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times log_2(\frac{|T_i|}{|T|}) \tag{4}$$

The condition on which C4.5 selects the test to partition the set of available cases is defined as:

$$gain\ ratio(X) = \frac{gain(X)}{split\ info(X)} \tag{5}$$

C4.5 selects the test that maximizes *gain ratio* value.

Once the initial decision tree is constructed, a pruning procedure is initiated to decrease the overall tree size and decrease the estimated error rate of the tree[9].

Additional parameters that affect the resulting decision tree are:

- *weight*, which specifies the minimum number of cases of at least two outcomes of a test. By default this value is two, meaning that for the test to be accepted, at least two outcomes of a test must have two or more cases. This prevents near-trivial splits that would result in almost flat and really wide trees;

- *confidence level*, which is used for prediction of tree error rates and affects the pruning process. The lower the confidence level, the higher the amount of pruning that will take place;

- *attribute grouping*, which can be used to create attribute value groups for discrete attributes and possibly infer patterns occurring in sets of cases with different values of an attribute, but do not occur for other values of that attribute;

Decision Tree:
message_size <= 512 :
— communicator_size <= 4 :
— — message_size <= 32 : ring (12.0/1.3)
— — message_size > 32 : linear (8.0/2.4)
— communicator_size > 4 :
— — communicator_size > 8 : bruck (100.0/1.4)
— — communicator_size <= 8 :
— — — message_size <= 128 : bruck (8.0/1.3)
— — — message_size > 128 : linear (2.0/1.0)
message_size > 512 :
— message_size > 1024 : linear (78.0/1.4)
— message_size <= 1024 :
— — communicator_size > 56 : linear (5.0/1.2)
— — communicator_size <= 56 :
— — — communicator_size <= 8 : linear (3.0/1.1)
— — — communicator_size > 8 : bruck (5.0/1.2)

Table 1: C4.5 decision tree for Alltoall on Nano cluster. The numbers in parentheses following the leaves represent number of the training cases covered by each leaf and the number of cases misclassified by that leaf.

- *windowing*, which is used to enable construction of multiple trees based on a portion of the test data, and then select the best performing tree [9].

4 MPI collectives performance data and C4.5

We use the collective algorithm performance information on a particular system to extract the information about the *optimal methods*. The optimal method on the particular system is the method that achieves the lowest duration for a particular set of input parameters.

The collected performance data is described using the collective name, communicator and message size attributes. The collective name attribute has discrete values such as Broadcast, Reduce, etc. Communicator and message size attributes have continuous values.

The predefined set of classes in our case contains methods that were optimal for some of the data points. The class names consist of algorithm name and segment size used, for example, Linear_0KB, or SplittedBinary_16KB. The classes are well defined, and by construction, the data with the same input parameters can belong to a single class only.

As far as the "sufficient" data requirement is concerned, the performance measurement data contains a considerable number of data points in the communicator - message size range. We do not cover every single possible communicator or message size, but our training data set usually contains around 1000 data points, so we feel that for this type of problem, collected data is sufficient to give reasonable results.

Table 1 shows a simple decision tree constructed by C4.5 from the data for an Alltoall collective on a Nano cluster.

The goal of this work is construction of decision functions, so we provide the functionality to generate the decision function source code in C from the constructed decision trees: the internal nodes are replaced by a corresponding *if* statement, and leaf nodes return the decision method

index/name. The `c4.5rules` program is supplied with the software release, but we did not use it for this purpose.

5 Experimental results and analysis

In this work, we used release 8 of C4.5 implementation by J.R. Quinlan [15] to construct decision trees based on existing performance data for Broadcast, Reduce, and Alltoall collectives collected at the Grig cluster at The University of Tennessee at Knoxville and the Nano cluster located at the Lawrence Berkeley National Laboratory.

The Grig cluster had 64 dual Intel(R) Xeon(TM) processor nodes at 3.2 GHz and Fast Ethernet and MX interconnects. The experimental data from the Grig cluster in this paper was gathered using Fast Ethernet interconnect. The Nano cluster had 128 dual Intel(R) Xeon(TM) processor nodes at 3.6 GHz and a Topspin PCI Express Infiniband interconnect.

In our experiments, we tested decision trees constructed using different weight and confidence level constraints. We did not use windowing because our data was relatively sparse in comparison to the complete communicator - message size domain size, so we did not expect that there would be a benefit to not utilizing all available data points. Also, since communicator and message sizes were described as continuous attributes, we were not able to use the grouping functionality of C4.5.

We constructed decision trees both per-collective (e.g. just for Broadcast or Alltoall) and for the set of collectives that have similar or the same set of available implementations (eg. both have Linear, Binary, and Pipeline algorithms) and for which we expected to have similar decision functions (e.g. Broadcast and Reduce).

5.1 Analysis of Broadcast decision trees

Figure 2 shows six different decision maps[2] for a broadcast collective on the Grig cluster. We considered five different broadcast algorithms (Linear, Binomial, Binary, Splitted Binary, and Pipeline)[3] and four different segment sizes (no segmentation, 1KB, 8KB, and 16KB). The measurements covered all communicator sizes between two and 28 processes and message sizes in 1B to 384KB range with total of 1248 data points. The original performance data set had 1248×20 data points.

Command Line	Before Pruning			After Pruning			
	Size	Errors		Size	Errors		Predicted Error
-m 2 -c 25	133	102	(7.9%)	127	103	(7.9%)	14.6%
-m 4 -c 25	115	114	(8.8%)	95	122	(9.4%)	15.0%
-m 6 -c 15	99	135	(10.4%)	65	149	(11.5%)	17.6%
-m 8 -c 5	73	155	(12.0%)	47	166	(12.8%)	21.0%
-m 40 -c 5	21	231	(17.8%)	21	231	(17.8%)	21.9%

Table 2: Broadcast decision tree statistics corresponding to the data presented in Figure 2. Size refers to the number of leaf nodes in the tree. Errors are in terms of misclassified training cases. The data set had 1248 training cases.

[2]Decision map is 2D representation of decision tree output for particular communicator and message sizes ranges.
[3]For more details on these algorithms, refer to [8].

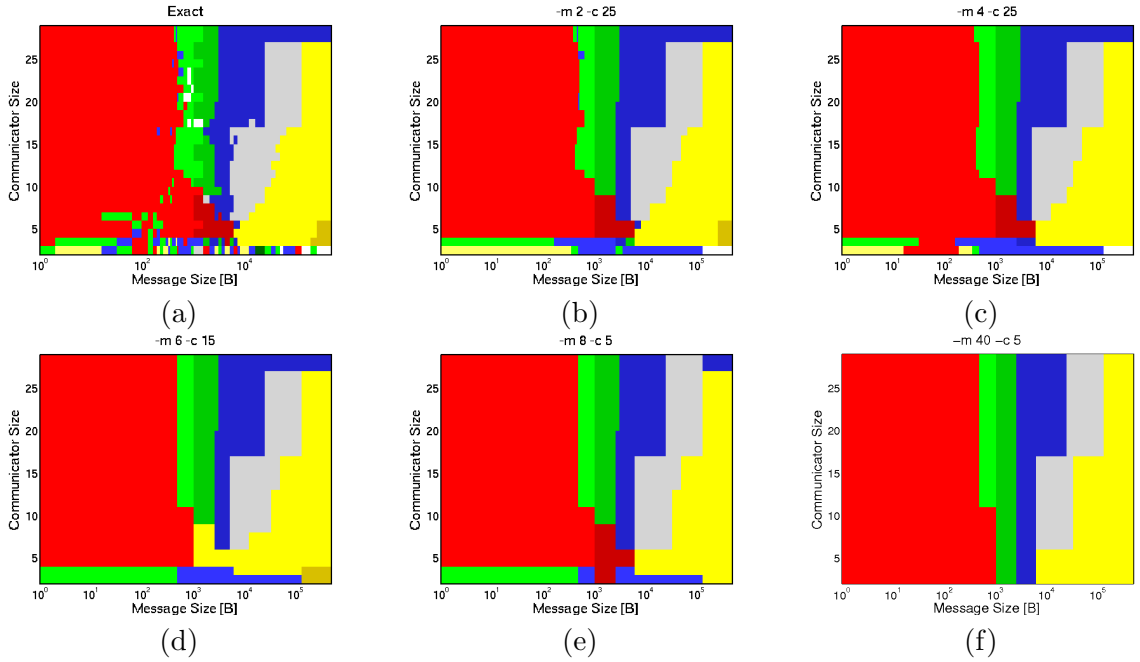

Figure 2: Broadcast decision maps from Grig cluster: (a) Measured (b) '-m 2 -c 25' (c) '-m 4 -c 25' (d) '-m 6 -c 15' (e) '-m 8 -c 5' (f) '-m 40 -c 5'. Different colors correspond to different method indices. The trees were generated using the specified command line parameters. The x-axis scale is logarithmic. The bright red color represents a Linear algorithm with no segmentation, the shades of green represent Binomial without segmentation and with 1KB segments, dark blue is a Binary algorithm with 1KB segments, gray is a Splitted Binary algorithm with 1KB segments, and shades of yellow are Pipeline algorithms with 1KB and 8KB segments.

In the upper left corner of Figure 2 we can find an exact decision map generated from experimental data. The subsequent maps were generated from C4.5 decision trees constructed by specifying different values for weight (`-m`) and confidence level (`-c`) parameters. The statistics about these trees can be found in Table 2.

The exact decision map in Figure 2 exhibits trends, but there is a considerable amount of information for intermediate size messages (between 1KB and 10KB) and small communicator sizes. The decision maps generated from different C4.5 trees capture general trends very well. The amount of captured detail depends on weight, which determines how the initial tree will be built, and confidence level, which affects the tree pruning process. "Heavier" trees require that branches contain more cases, thus limiting the number of fine-grained splits. A lower confidence level allows for more aggressive pruning, which also results in coarser decisions.

Looking at the decision tree statistics in Table 2, we can see that the default C4.5 tree ('-m 2 -c 25') has 127 leaves and a predicted misclassification error of 14.6%. Using a slightly "heavier" tree '-m 4 -c 25' gives us a 25.20% decrease in tree size (95 leaves) and maintains almost the same predicted misclassification error. As we increase tree weight and decrease the confidence level, we produce the tree with only 21 leaves (83.46% reduction in size) with a 50% increase in predicted misclassifications (21.9%).

In this work, the goal is to construct reasonably small decision trees that will provide good run-time performance of an MPI collective of interest. Given this goal, the number of misclassified training examples is not the main figure of merit we need to consider. To determine the "quality" of the resulting tree in terms of collective operation performance, we consider the performance penalty of the tree.

The performance penalty is the relative difference between the performance obtained using methods predicted by the decision tree instead of the experimentally optimal ones:

$$pp_X(comm, msg) = \frac{t_X(comm, msg) - t(comm, msg)}{t(comm, msg)} \tag{6}$$

where $t_X(comm, msg)$ is operation duration using the method predicted by tree X for communicator and message sizes $comm$ and msg, and $t(comm, msg)$ is the operation duration using experimentally optimal method. The mean performance penalty of a decision tree is the mean value of performance penalties for all communicator and message sizes of interest.

Command Line	**Performance Penalty of Decision Tree**			
	Min [%]	**Max [%]**	**Mean [%]**	**Median [%]**
-m 2 -c 25	0.00	75.41	0.66	0.00
-m 4 -c 25	0.00	316.97	1.16	0.00
-m 6 -c 15	0.00	316.97	3.24	0.00
-m 8 -c 5	0.00	316.97	1.66	0.00
-m 40 -c 5	0.00	316.97	2.08	0.00

Table 3: Performance penalty of Broadcast decision trees corresponding to the data presented in Figure 2 and Table 2.

Table 3 provides performance penalty statistics for the Broadcast decision trees we are considering. We can see that the minimum, mean, and median performance penalty values are rather low - less than 4%, even as low as 0.66%, indicating that even the simplest tree we considered should provide good run-time performance. Moreover, the simplest tree, "-m 40 -c 5", had a lower performance penalty than the "-m 6 -c 15," which indicates that the percent of misclassified training cases does not translate directly into performance penalty of the tree.

It is also interesting to consider the case with maximum performance penalty. Most of the trees would incur 316.97% at communicator size 25 and message size 480. For this data point, the exact tree selects the Binary algorithm without segmentation (1.12 ms) while the C4.5 decision trees would select the Binomial algorithm without segmentation (4.69 ms). Additionally, in the "-m 40 -c 5" tree, only six data points had a performance penalty above 50%.

5.2 Combined decision trees

MPI collective operations can be grouped into four categories based on their data exchange pattern: one-to-many, many-to-one, many-to-many, and "other" (such as Scan and Exscan). It is reasonable to expect that similar collectives have similar decision functions on the same system. We decided to analyze decision trees generated from the experimental data collected for Broadcast and Reduce collectives on the Grig system. Our implementations of these collectives are symmetric: each of them has Linear, Binomial, Binary, and Pipeline based implementations. Broadcast supports the

Splitted Binary algorithm for which we do not have an equivalent in Reduce implementation, but we expected that C4.5 will be able to handle these cases correctly.

The training data for this experiment contained three attributes (collective name, communicator size, and message size) and the set of predetermined classes was the same as in the Broadcast-only case.

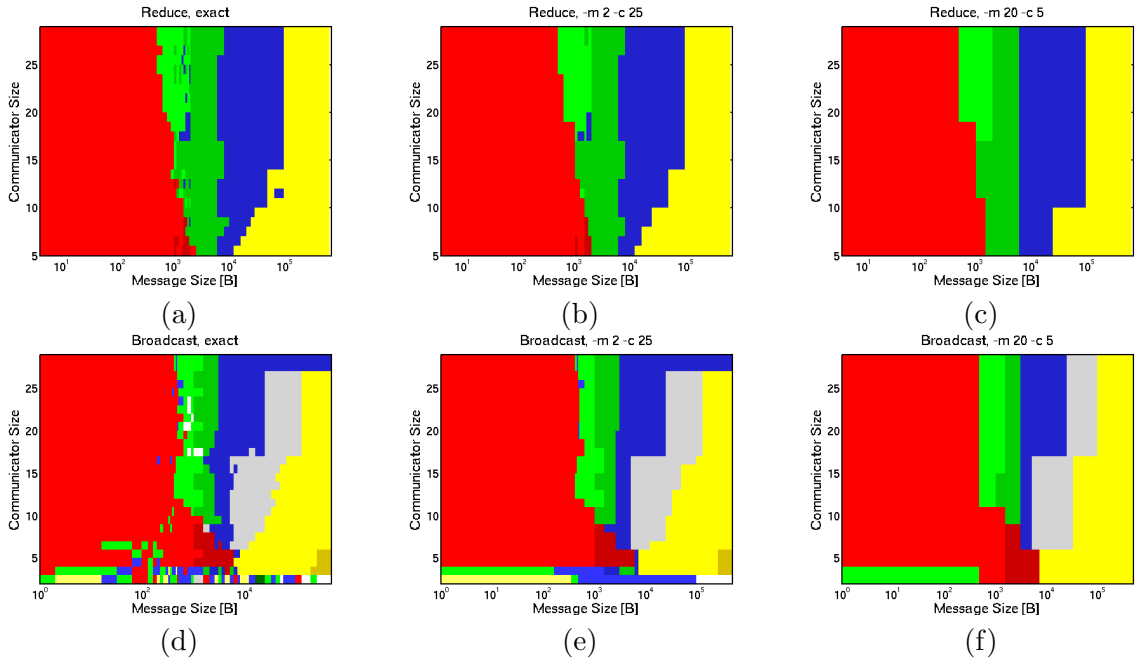

Figure 3: Combined Broadcast and Reduce decision maps from the Grig cluster: (a) Reduce, Exact (b) Reduce, '-m 2 -c 25' (c) Reduce, '-m 20 -c 5' (d) Broadcast, Exact (e) Broadcast, '-m 2 -c25' (f) Broadcast, '-m 20 -c 5'. Color has the same meaning as in Figure 2.

Figure 3 shows the decision maps generated from the combined broadcast and reduce decision tree. The left most maps in both rows are the exact decisions for each of the collectives based on experimental data. The remaining maps are generated by querying the combined decision tree. Figures 3 (b) and (e) were generated using a "-m 2 -c 25" decision tree with 221 leaves and a 12.6% predicted misclassification error, while (c) and (f) were generated by a "-m 20 -c 5" decision tree with 55 leaves and a 20.6% predicted misclassification error. Table 4 provides the detailed information about the combined Broadcast and Reduce trees we considered. The mean performance penalty of the combined tree for each of the collectives is less than 2.5% (Figure 4).

The structure of combined Broadcast and Reduce decision trees reveals that the test for the type collective occurs for the first time on the third level of the tree. Even then, the subtrees have somewhat similar structure as we can see in Table 5. Considering the message sizes in the range $1,408$B - $6,144$B, we can see that both Broadcast and Reduce would use a variant of the Binomial algorithm. However, switching points are shifted, and for larger communicator sizes and smaller messages, Reduce would use a non-segmented version of the Binomial algorithm.

Command Line	Before Pruning			After Pruning			
	Size	Errors		Size	Errors		Predicted Error
-m 2 -c 25	239	137	(6.0%)	221	142	(6.2%)	12.6%
-m 6 -c 25	149	205	(9.0%)	115	220	(9.6%)	14.0%
-m 8 -c 25	127	225	(9.8%)	103	235	(10.3%)	14.4%
-m 20 -c 5	63	310	(13.6%)	55	316	(13.8%)	20.6%
-m 40 -c 25	33	392	(17.1%)	33	392	(17.1%)	19.6%

Table 4: Statistics for combined Broadcast and Reduce decision trees corresponding to the data presented in Figure 3. Size refers to the number of leaf nodes in the tree. Errors are in terms of misclassified training cases. The data set had 2286 training cases.

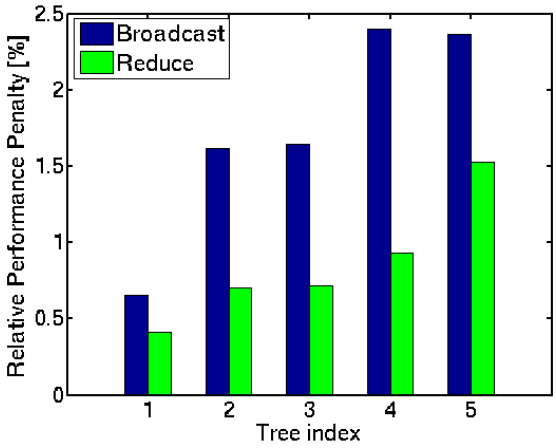

Figure 4: Mean performance penalty of the combined decision tree for each of the collectives. Tree index denotes the decision tree: (1) '-m 2 -c 25' (2) '-m 6 -c 25' (3) '-m 8 -c25' (4) '-m 20 -c 5', (5) '-m 40 -c 25'.

6 Discussion and future work

In this paper we studied the applicability of C4.5 decision trees to the MPI collective algorithm/method selection problem. We assumed that the system of interest has been benchmarked and that detailed performance information exists for each of the available collective communication methods. Using this information, we focused on investigating whether C4.5 decision trees are a feasible way to generate static decision functions.

Using a publicly available C4.5 implementation, we constructed decision trees based on existing performance data for Broadcast, Reduce, and Alltoall collectives. We evaluated decision trees constructed using different weight and confidence level parameters.

Our results show that C4.5 decision trees can be used to generate a reasonably small and very accurate decision function: the mean performance penalty on existing performance data was within the measurement error for all trees we considered. For example, the Broadcast decision tree with only 21 leaves was able to achieve a mean performance penalty of 2.08%. Moreover, using this tree, only six points in the communicator, message size ranges we tested would incur more than 50%

```
...
message_size > 1408 :
— message_size <= 6144 :
— — collective = Broadcast:
— — — message_size <= 2560 : Binomial_1K (81.0/35.0)
— — — message_size > 2560 : Binary_1K (135.0/36.0)
— — collective = Reduce:
— — — message_size > 1920 : Binomial_1K (132.0/16.0)
— — — message_size <= 1920 :
— — — — communicator_size <= 15 : Binomial_1K (40.0/16.0)
— — — — communicator_size > 15 : Binomial_0K (48.0/24.0)
...
```

Table 5: Segment of combined Broadcast and Reduce decision tree '-m 40 -c 25'.

performance penalty. Similar results were obtained for Reduce and Alltoall.

Additionally, we combined the experimental data for Reduce and Broadcast to generate the combined decision trees. These trees were also able to produce decision functions with less than a 2.5% relative performance penalty for both collectives. This indicates that it is possible to use information about one MPI collective operation to generate a reasonably well decision function for another collective.

Our findings demonstrate that the C4.5 algorithm and decision trees are applicable to this problem and should be more widely used in this domain. In the future, we plan to use C4.5 decision trees to reevaluate decision functions in FT-MPI and tuned collective module of Open MPI, as well as integrate C4.5 decision trees with our MPI collective testing and performance measurement framework, OCC.

Acknowledgments

This work was supported by Los Alamos Computer Science Institute (LACSI), funded by Rice University Subcontract #R7B127 under Regents of the University Subcontract #12783-001-05 49.

References

[1] R. Rabenseifner, "Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512," in *Proceedings of the Message Passing Interface Developer's and User's Conference*, pp. 77–85, 1999.

[2] J. Worringen, "Pipelining and overlapping for MPI collective operations," in *28th Annyal IEEE Conference on Local Computer Network*, (Bonn/Königswinter, Germany), pp. 548–557, IEEE Computer Society, October 2003.

[3] R. Rabenseifner and J. L. Träff, "More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems," in *Proceedings of EuroPVM/MPI*, Lecture Notes in Computer Science, Springer-Verlag, 2004.

[4] E. W. Chan, M. F. Heimlich, A. Purkayastha, and R. M. van de Geijn, "On optimizing of collective communication," in *Proceedings of IEEE International Conference on Cluster Computing*, pp. 145–155, 2004.

[5] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[6] 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 seventh ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pp. 131–140, ACM Press, 1999.

[7] M. Bernaschi, G. Iannello, and M. Lauria, "Efficient implementation of reduce-scatter in MPI," *Journal of Systems Architure*, vol. 49, no. 3, pp. 89–108, 2003.

[8] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance analysis of mpi collective operations," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - PMEO-PDS Workshop*, (Washington, DC, USA), p. 272.1, IEEE Computer Society, 2005.

[9] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, California: Morgan Kaufmann Publishers, 1993.

[10] G. E. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pješivac-Grbović, K. London, and J. Dongarra, "Extending the mpi specification for process fault tolerance on high performance computing systems," in *Proceedings of the International Supercomputer Conference (ISC) 2004*, Primeur, 2004.

[11] G. E. Fagg, G. Bosilca, J. Pješivac-Grbović, T. Angskun, and J. Dongarra, "Tuned: A flexible high performance collective communication component developed for open mpi," in *Proccedings of 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems*, (Innsbruck, Austria), pp. 65–72, Springer-Verlag, September 2006.

[12] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 345–389, 1998.

[13] R. Vuduc, J. W. Demmel, and J. A. Bilmes, "Statistical Models for Empirical Search-Based Performance Tuning," *International Journal of High Performance Computing Applications*, vol. 18, no. 1, pp. 65–94, 2004.

[14] V. N. Vapnik, *Statistical Learning Theory*. New York, NY: Wiley, 1998.

[15] J. R. Quinlan, "C4.5 source code." http://www.rulequest.com/Personal/, Accesses on September 2006.