Decision Trees and MPI Collective Algorithm Selection Problem

Jelena Pješivac–Grbović, George Bosilca, Graham E. Fagg, Thara Angskun, and Jack J. Dongarra

Innovative Computing Laboratory,
The University of Tennessee Computer Science Department
1122 Volunteer Blvd., Knoxville, TN 37996-3450, USA
{pjesa, bosilca, fagg, angskun, dongarra}@cs.utk.edu

Abstract. Selecting the close-to-optimal collective algorithm based on the parameters of the collective call at run time is an important step for 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 both the decision tree properties and the 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 the design and implementation of efficient collective algorithms for both homogeneous and heterogeneous cluster environments [2–7]. 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 that 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. As the amount of the system performance information can be significant, the decision function building mechanism must be efficient both in terms of storage and the time-to-decision performance. We are interested in different approaches to storing and accessing the large amount of performance data.

This paper studies the applicability of C4.5 decision trees [8] to the 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; and 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 [6]. 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.

Our previous work [12] used quadtree encoding to store the information about the optimal collective algorithm performance on a particular system. This structure was used either to generate a decision function or as an in-memory decision system for selecting a close-to-optimal method at run-time.

Alternatively, data mining techniques can be applied to the algorithm selection problem with replacing the original problem 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.

Vuduc et al. construct statistical learning models to build different decision functions for the matrix-matrix multiplication algorithm selection [13]. In their

[1] Detailed benchmarking of all possible methods takes a significant amount of time. If this is not an option, performance profiles can be generated using a limited set of performance measurements coupled with performance modeling [9].

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 the *support vector machines* 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 are used instead of matrix-matrix multiplication, and we use the C4.5 algorithm instead of *support vector machines* 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 [8]. 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, the attribute value may be missing or can be ignored.
- *Predefined classes*: the training data has to be divided into 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 [8].

The C4.5 algorithm constructs the initial decision tree using a variation of the 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 a particular test [8].

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[8].

```
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:
1) $T$ 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;
2) $T$ contains no cases:
    A leaf node is created for $T$ and is assigned the most most frequent class at the parent node;
3) $T$ contains cases that belong to more than one class:
    Find a test that splits $T$ 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 $T$.
```

Fig. 1. Hunt's method for decision tree construction [8].

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. 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 greater the amount of pruning that takes 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.
– *windowing*, which enables construction of multiple trees based on a portion of the test data and then selects the best performing tree [8].

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 can be 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. Additionally, constructive induction can be used to create composite attributes that can capture additional system information. For example, a *total data per node* attribute can be used to distinguish between a single-process-per-node and two-processes-per-node run. Moreover, such attributes can potentially indirectly capture information about the system bottlenecks. In this paper, however, we focus on performance data that is fully described by the collective name, communicator and message size attributes.

The predefined set of classes in our case contains methods that were optimal for some of the data points. The class names consist of the algorithm name and segment size used, for example, Linear_0KB or SplitBinary_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.

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. We did not utilize the `c4.5rules` program for this purpose.

5 Experimental results and analysis

In this work, we used release 8 of the C4.5 implementation by J.R. Quinlan [15] to construct decision trees based on existing performance data for broadcast and reduce collectives collected on the Grig cluster at the University of Tennessee, Knoxville.

The Grig cluster has 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 the Fast Ethernet interconnect.

The performance data in this paper was collected using the MPICH-2 [16] version 1.0.3 and OCC library [17]. The OCC library implements a number of collective algorithms on top of MPI point-to-point operations and can be used with any MPI implementation. The OCC benchmark measures collective operation performance by repeating the operation a number of times. To avoid pipelining effects, a balanced barrier (such as Bruck) is inserted between every collective call, and the time to execute the barrier is subtracted from the total running time. More advanced benchmarks, such as SKaMPI [18], can be used as well. The only requirement is to convert the benchmark results to C4.5 input file format.

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 by 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 (e.g., 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 three different decision maps[2] for a broadcast collective on the Grig cluster. We considered five different broadcast algorithms (Linear, Binomial, Binary, Split 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 the 1B to 384KB range with total of 1248 data points. The original performance data set contains $1248 \times 4 \times 5$ data points.

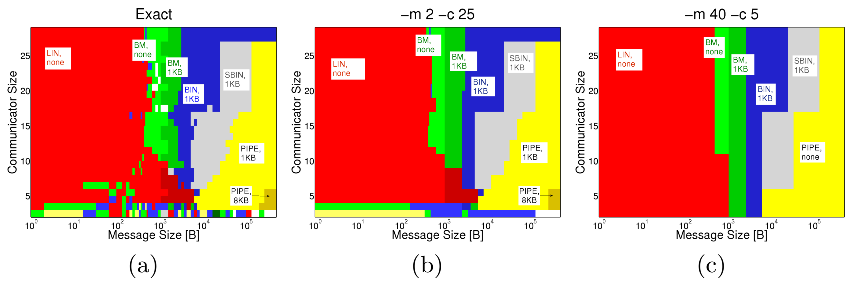

Fig. 2. Broadcast decision maps from the Grig cluster: (a) Measured (b) '-m 2 -c 25' (c) '-m 40 -c 5'. X-axis corresponds to message sizes, Y-axis represents the communicator sizes. Different colors correspond to different method indices. In this Figure, "LIN" stands for Linear, BM for Binomial, "BIN" for Binary, "SBIN" for Split Binary, and "PIPE" for Pipeline algorithm. Also, "none", "1KB", and "8KB" refer to the corresponding segment size.

Figure 2 (a) shows an exact decision map generated from experimental data. The subsequent maps were generated by C4.5 decision trees constructed by specifying different values for weight ("`-m`") and confidence level ("`-c`") parameters (See Section 3). The statistics about these and additional trees can be found in Table 1.

The exact decision map in Figure 2 (a) 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

[2] Decision map is a 2D representation of the decision tree output for a particular communicator and message size ranges.

[3] For more details on these algorithms, refer to [9].

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.

Command line	**Before pruning**		**After pruning**			**Performance penalty**		
	Size	**Errors**	**Size**	**Errors**	**Predicted Error**	**Min**	**Max**	**Mean**
-m 2 -c 25	133	7.9%	127	7.9%	14.6%	0%	75.41%	0.66%
-m 4 -c 25	115	8.8%	95	9.4%	15.0%	0%	316.97%	1.16%
-m 6 -c 15	99	10.4%	65	11.5%	17.6%	0%	316.97%	3.24%
-m 8 -c 5	73	12.0%	47	12.8%	21.0%	0%	316.97%	1.66%
-m 40 -c 5	21	17.8%	21	17.8%	21.9%	0%	316.97%	2.08%

Table 1. 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. The median performance penalty was 0% in all cases.

Looking at the decision tree statistics in Table 1, 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 and the experimentally optimal ones.

The last three columns in Table 1 provide performance penalty statistics for the broadcast decision trees we are considering. 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 a performance penalty of the tree.

In all cases, the mean and median performance penalty values are excellent, but the maximum performance penalty of 316.97% requires explanation. At communicator size 25 and message size 480, the experimentally optimal method is Binary algorithm without segmentation (1.12 ms), but most decision trees select

Binomial algorithm without segmentation (4.69 ms). However, the Binomial algorithm performance in the neighborhood of this data point is around and less than 1 ms, which implies that the 4.69 ms result is probably affected by external factors. Additionally, in the "-m 40 -c 5" tree, only six data points had a performance penalty above 50%.

5.2 Combined decision trees

It is reasonable to expect that similar MPI collective operations have similar decision functions on the same system. To test this hypothesis, we decided to analyze the 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 Split Binary algorithm for which we do not have an equivalent in reduce implementation, but we expect that C4.5 should be able to handle these cases correctly.

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

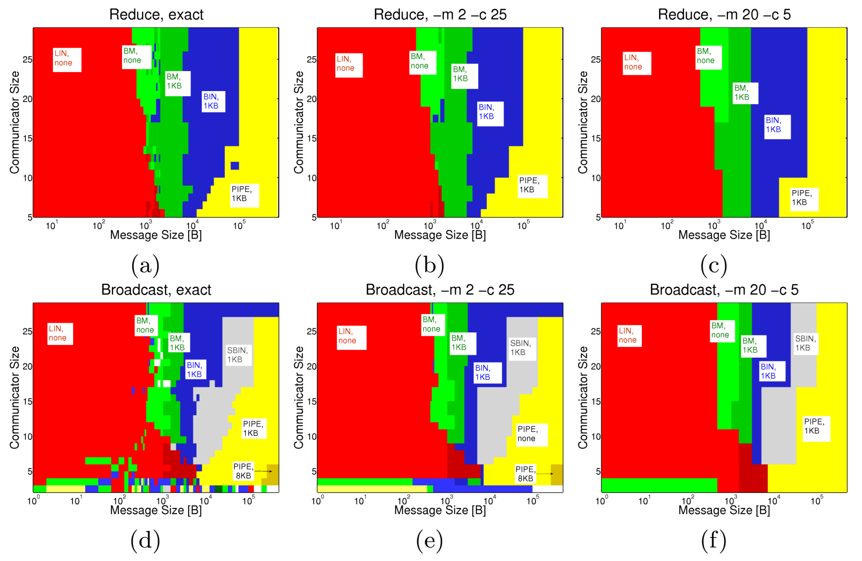

Fig. 3. Combined broadcast and reduce decision maps from the Grig cluster: (a) reduce, Exact (b) reduce, '-m 2 -c 25' (c) reduce, '-m 20 -c 5' (d) broadcast, Exact (e) broadcast, '-m 2 -c25' (f) broadcast, '-m 20 -c 5'. Color has the same meaning as in Figure 2.

Figure 3 shows the decision maps generated from the combined broadcast and reduce decision tree. The leftmost 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, while (c) and (f) were generated by a “-m 20 -c 5” decision tree. Table 2 provides the detailed information about the combined decision trees of interest including the mean performance penalty of the trees.

Command line	**Before Pruning**		**After Pruning**			**Mean performance penalty**	
	Size	**Errors**	**Size**	**Errors**	**Predicted error**	**Broadcast**	**Reduce**
-m 2 -c 25	239	137	221	142	12.6%	0.66%	0.41%
-m 6 -c 25	149	205	115	220	14.0%	1.62%	0.71%
-m 8 -c 25	127	225	103	235	14.4%	1.64%	0.72%
-m 20 -c 5	63	310	55	316	20.6%	2.40%	0.93%
-m 40 -c 25	33	392	33	392	19.6%	2.37%	1.53%

Table 2. 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.

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. This implies that the combined decision tree is able to capture the common structure of the optimal implementation for these collectives, as one would expect based on decision maps in Figure 3.

5.3 C4.5 decision trees vs. quadtree encoding

Quadtree encoding is an alternative method for storing performance information and generating decision functions based on the performance data. We explored this approach in [12].

The quadtree results for broadcast collective on the Grig cluster showed that the 6-level quadtree can represent experimental data fully. The 5-level quadtree for this data set, incurred around 5.41% mean performance penalty, while the 3-level quadtree introduced 8.83% mean performance penalty. In comparison, the C4.5 decision trees we considered incurred less than 3.5% mean performance penalty.

The main benefit of the quadtree encoding is the fact that the size of the generated quadtree can be easily manipulated. This allows us to limit the maximum number of expressions that need to be evaluated to reach the decision. The depth of the C4.5 decision tree is hard to estimate, making it impossible to set an a priori limit on the maximum number of expressions to be evaluated in the final decision function.

The main benefit of C4.5 decision trees is the ability to handle multi-dimensional data automatically. In this paper, we added collective name as a third dimension in Section 5.2. The composite attributes or ordinal attributes that describe system information can be automatically handled by C4.5. The quadtree encoding is restricted to two-dimensional data (communicator and message sizes), and cannot be easily extended to include additional attributes. Moreover, one-dimensional decisions (such as "for this communicator size and all message sizes use method A, but do not use this method for neighboring communicator sizes") cannot be captured with size-restricted quadtrees, while C4.5 does not have this problem.

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 and reduce 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% 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 good decision function for another collective, under the assumption that the two are similar.

In the direct comparison to the decision functions generated by the quadtrees from [12], C4.5 trees produced decision functions with lower mean performance penalties. However, the size and structure of a C4.5 decision tree is less predictable than the one of the corresponding quadtree. More detailed comparison of both methods is planned for future work.

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 the tuned collective module of Open MPI. We also plan to integrate C4.5 decision trees with our MPI collective testing and performance measurement framework, OCC.

References

1. Rabenseifner, R.: 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. (1999) 77–85
2. Worringen, J.: Pipelining and overlapping for MPI collective operations. In: 28th Annyal IEEE Conference on Local Computer Network, Bonn/Königswinter, Germany, IEEE Computer Society (2003) 548–557
3. Rabenseifner, R., Träff, J.L.: 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. Chan, E.W., Heimlich, M.F., Purkayastha, A., van de Geijn, R.M.: On optimizing of collective communication. In: Proceedings of IEEE International Conference on Cluster Computing. (2004) 145–155
5. Bernaschi, M., Iannello, G., Lauria, M.: Efficient implementation of reduce-scatter in MPI. Journal of Systems Architure **49**(3) (2003) 89–108
6. Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of Collective Communication Operations in MPICH. International Journal of High Performance Computing Applications **19**(1) (2005) 49–66
7. Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A., Bhoedjang, R.A.F.: MagPIe: MPI's collective communication operations for clustered wide area systems. In: Proceedings of the ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, ACM Press (1999) 131–140
8. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, California (1993)
9. Pješivac-Grbović, J., Angskun, T., Bosilca, G., Fagg, G.E., Gabriel, E., Dongarra, J.J.: Performance analysis of mpi collective operations. In: Proceedings of IPDPS'05 - PMEO-PDS Workshop, IEEE Computer Society (2005) 272.1
10. Fagg, G.E., Gabriel, E., Bosilca, G., Angskun, T., Chen, Z., Pješivac-Grbović, J., London, K., Dongarra, J.: 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. Fagg, G.E., Bosilca, G., Pješivac-Grbović, J., Angskun, T., Dongarra, J.: Tuned: A flexible high performance collective communication component developed for open mpi. In: Proccedings of DAPSYS'06, Innsbruck, Austria, Springer-Verlag (2006) 65–72
12. Pješivac-Grbović, J., Fagg, G.E., Angskun, T., Bosilca, G., Dongarra, J.J.: Mpi collective algorithm selection and quadtree encoding. In: Proceedings of 13th European PVM/MPI User's Group Meeting. LNCS 4192, Bonn, Germany, Springer-Verlag (2006) 40–48
13. Vuduc, R., Demmel, J.W., Bilmes, J.A.: Statistical Models for Empirical Search-Based Performance Tuning. International Journal of High Performance Computing Applications **18**(1) (2004) 65–94
14. Vapnik, V.N.: Statistical Learning Theory. Wiley, New York, NY (1998)
15. Quinlan, J.R.: C4.5 source code. `http://www.rulequest.com/Personal` (2006)
16. MPICH-2: Implementation of MPI 2 standard. `http://www-unix.mcs.anl.gov/mpi/mpich2` (2005)
17. OCC: Optimized Collective Communication Library. `http://www.cs.utk.edu/~pjesa/projects/occ` (2005)
18. SKaMPI: Special Karlsruher MPI Benchmark. `http://liinwww.ira.uka.de/~skampi` (2005)