Dynamic Load Balancing of Unstructured Computations in Decision Tree Classifiers *

A. Srivastava
Information Technology Lab
Hitachi America, Ltd.
anurags@hitachi.com

E. Han V. Kumar
Dept. of Computer Science
Army HPC Research Center
University of Minnesota
{han,kumar}@cs.umn.edu

V. Singh
Information Technology Lab
Hitachi America, Ltd.
vsingh@hitachi.com

Abstract

One of the important problems in data mining is discovering classification models from datasets. Application domains include retail target marketing, fraud detection, and design of telecommunication service plans. Highly parallel algorithms for constructing classification decision trees are desirable for dealing with large data sets. Algorithms for building classification decision trees have a natural concurrency, but are difficult to parallelize due to the inherent dynamic nature of the computation. In this short paper, we present parallel formulations of classification decision tree learning algorithm based on induction. We describe two basic parallel formulations, Synchronous Tree Construction Approach and Partitioned Tree Construction Approach. We propose a hybrid method that employs the good features of these methods. Our experimental results of the hybrid method on an IBM SP-2 demonstrate excellent speedups.

1 Introduction

Data mining [3, 5, 7, 8, 13, 14] is the efficient and possibly unsupervised discovery of interesting, useful and previously unknown patterns from databases. One of the important problems in data mining is discovering classification models from datasets. Given an input training dataset with a number of attributes and a class label, the discovery task is to build a model of the class label such that the model can be used to classify new datasets. Application domains include retail target marketing, fraud detection, and design of telecommunication service plans. Several classification models like neural networks [10], genetic algorithms [6], and decision trees [12], have been proposed. Among these classification models, decision trees are probably the most popular since they obtain reasonable accuracy [4], they are relatively inexpensive to compute and they are easy to interpret.

In the data mining domain, the data to be processed tends to be very large. Hence, it is highly desirable to design computationally efficient as well as scalable algorithms. Classification decision tree construction algorithms have natural concurrency, as once a node is generated, all of its children in the classification tree can be generated concurrently. Furthermore, the computation for generating successors of a classification tree node can also be decomposed by performing data decomposition on the training data. Nevertheless, parallelization of the algorithms for constructing the classification tree is challenging for the following reasons. First, the shape of the tree is highly irregular and is determined only at runtime. Furthermore, the amount of work associated with each node also varies, and is data dependent. Hence any static allocation scheme is likely to suffer from major load imbalance. Second, even though the successors of a node can be processed concurrently, they all use the training data associated with the parent node. If this data is dynamically partitioned and allocated to different processors that perform computation for different nodes, then there is a high cost for data movements. If the data is not partitioned appropriately, then performance can be bad due to the loss of locality.

In this short paper, we present parallel formulations of classification decision tree learning algorithm based on induction. We describe two basic parallel formulations. One

* A significant part of this work was done while Anurag Srivastava and Vineet Singh were at IBM TJ Watson Research Center. This work was supported by NSF grant ASC-9634719, Army Research Office contract DA/DAAH04-95-1-0538, Cray Research Inc. Fellowship, and IBM partnership award, the content of which does not necessarily reflect the policy of the government, and no official endorsement should be inferred. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute, Cray Research Inc., and NSF grant CDA-9414015.

is based on *Synchronous Tree Construction Approach* and the other is based on *Partitioned Tree Construction Approach*. We propose a hybrid method that employs the good features of these methods. Experimental results of the hybrid method on an IBM SP-2 demonstrate excellent speedups. Extended version of this paper is available at [15].

2 Sequential Classification Rule Learning Algorithms

Most of the existing induction–based algorithms like *C4.5* [12], *CDP* [1], *SLIQ* [11], and *SPRINT* [13] use Hunt's method [12] as the basic algorithm. Here is a recursive description of Hunt's method for constructing a decision tree from a set T of training cases with classes denoted $\{C_1, C_2, \ldots, C_k\}$.

Case 1 T contains cases all belonging to a single class C_j. The decision tree for T is a leaf identifying class C_j.

Case 2 T contains cases that belong to a mixture of classes. A test is chosen, based on a single attribute, that has one or more mutually exclusive outcomes $\{O_1, O_2, \ldots, O_n\}$. Note that in many implementations, n is chosen to be 2 and this leads to a binary decision tree. T is partitioned into subsets $T_1, T_2, \ldots, T_n$, where T_i contains all the cases in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test, and one branch for each possible outcome. The same tree building machinery is applied recursively to each subset of training cases.

Case 3 T contains no cases. The decision tree for T is a leaf, but the class to be associated with the leaf must be determined from information other than T. For example, *C4.5* chooses this to be the most frequent class at the parent of this node.

In case 2 of Hunt's method, a test based on a single attribute is chosen for expanding the current node. The choice of an attribute is normally based on the entropy gains of the attributes. The entropy of an attribute is calculated from class distribution information. For a discrete attribute, class distribution information of each value of the attribute is required. For a continuous attribute, binary tests involving all the distinct values of the attribute are considered. Once the class distribution information of all the attributes are gathered, each attribute is evaluated in terms of either *entropy* [12] or *Gini Index* [2]. The best attribute is selected as a test for the node expansion.

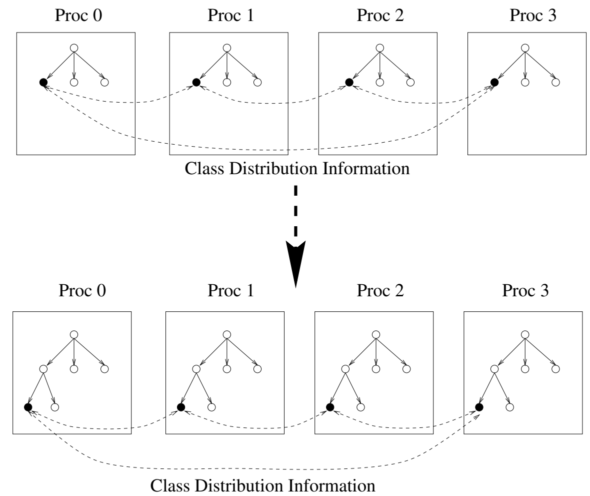

Figure 1. Synchronous Tree Construction Approach with Depth–First Expansion Strategy

3 Parallel Formulations

In this section, we give two basic parallel formulations for the classification decision tree construction and a hybrid scheme that combines good features of both of these approaches. We focus our presentation for discrete attributes only. Our approaches are also applicable when continuous attributes are present. Detailed discussion of handling continuous attributes is given in [9, 13, 16]. In all parallel formulations, we assume that N training cases are randomly distributed to P processors initially such that each processor has N/P cases.

In Synchronous Tree Construction Approach approach, all processors construct a decision tree synchronously by sending and receiving class distribution information of local data. Figure 1 shows the overall picture. The root node has already been expanded and the current node is the leftmost child of the root (as shown in the top part of the figure). All the four processors cooperate to expand this node to have two child nodes. Next, the leftmost node of these child nodes is selected as the current node (in the bottom of the figure) and all four processors again cooperate to expand the node.

In Partitioned Tree Construction approach, whenever feasible, different processors work on different parts of the classification tree. In particular, if more than one processors cooperate to expand a node, then these processors are partitioned to expand the successors of this node. At the beginning, all processors work together to expand the root node of the classification tree. At the end, the whole classification tree is constructed by combining subtrees of each processor.

Figure 2 shows an example. First (at the top of the fig-

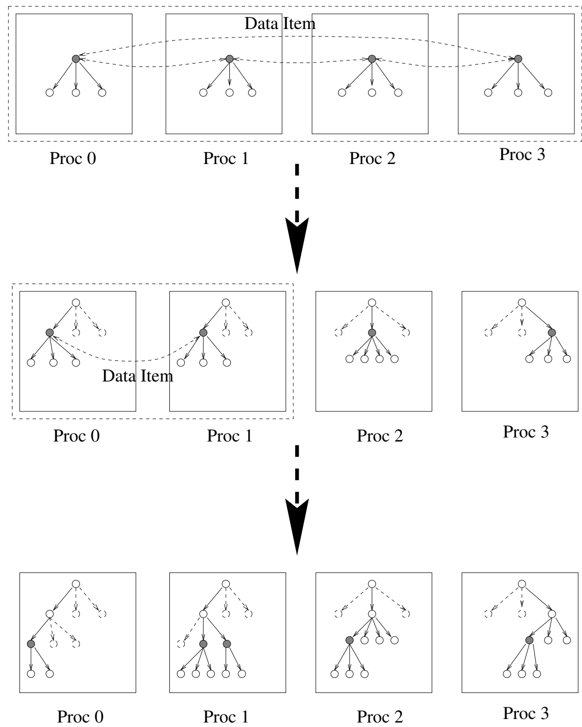

Figure 2. Partitioned Tree Construction Approach

ure), all four processors cooperate to expand the root node just like they do in the synchronous tree construction approach. Next (in the middle of the figure), the set of four processors is partitioned in three parts. The leftmost child is assigned to processors 0 and 1, while the other nodes are assigned to processors 2 and 3, respectively. Now these sets of processors proceed independently to expand these assigned nodes. In particular, processors 2 and processor 3 proceed to expand their part of the tree using the serial algorithm. The group containing processors 0 and 1 splits the leftmost child node into three nodes. These three new nodes are partitioned in two parts (shown in the bottom of the figure); the leftmost node is assigned to processor 0, while the other two are assigned to processor 1. From now on, processors 0 and 1 also independently work on their respective subtrees.

Our hybrid parallel formulation has elements of both schemes. The *Synchronous Tree Construction Approach* incurs high communication overhead as the tree deepens. The *Partitioned Tree Construction Approach* incurs high cost of data movements and load balancing after each partition of the tree expansion frontier. The hybrid scheme keeps continuing with the first approach as long as the communication cost incurred by the first formulation is not too high. Once this cost becomes high, the processors as well as the current frontier of the classification tree are partitioned into

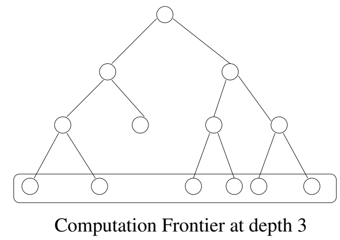

Figure 3. The computation frontier during computation phase

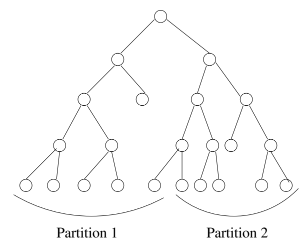

Figure 4. Binary partitioning of the tree to reduce communication costs

two parts.

As an example of the hybrid algorithm, Figure 3 shows a classification tree frontier at depth 3. So far, no partitioning has been done and all processors are working cooperatively on each node of the frontier. At the next frontier at depth 4, partitioning is triggered, and the nodes and processors are partitioned into two partitions as shown in Figure 4.

A key element of the algorithm is the criterion that triggers the partitioning of the current set of processors (and the corresponding frontier of the classification tree). If partitioning is done too frequently, then the hybrid scheme will approximate the partitioned tree construction approach, and thus will incur too much data movement cost. If the partitioning is done too late, then it will suffer from high cost for communicating statistics generated for each node of the frontier, like the synchronized tree construction approach. One possibility is to do splitting when the accumulated cost of communication becomes equal to the cost of moving records around in the splitting phase. More precisely, splitting is done when

$$\sum (Communication\ Cost) \geq MovingCost + LoadBalancing$$

4 Experimental Results

We have implemented the three parallel formulations using the MPI programming library. We use binary splitting at each decision tree node and grow the tree in breadth first manner. For generating large datasets, we have used the widely used synthetic dataset proposed in the *SLIQ* paper [11] for all our experiments. Ten classification functions were also proposed in [11] for these datasets. We have used the function 2 dataset for our algorithms. In this dataset, there are two class labels and each record consists of 9 attributes having 3 categoric and 6 continuous attributes. Experiments were done on an IBM SP2. The results for comparing speedup of the three parallel formulations are reported for parallel runs on 1, 2, 4, 8, and 16 processors. More experiments for the hybrid approach are reported for up to 128 processors. Each processor has a clock speed of 66.7 MHz with 256 MB real memory. The operating system is AIX version 4 and the processors communicate through a high performance switch (hps).

First, we present results of our schemes in the context of discrete attributes only. We compare the performance of the three parallel formulations on up to 16 processor IBM SP2. For these results, we discretized 6 continuous attributes uniformly. For measuring the speedups, we worked with 1.6 million training cases and increased the processors from 1 to 16. The results in Figure 5 show the speedup comparison of the three parallel algorithms proposed in this paper.

The results show that the synchronous tree construction approach has a good speedup for 2 processors, but it has a very poor speedup for 4 or more processors. There are two reasons for this. First, the synchronous tree construction approach incurs high communication cost, while processing lower levels of the tree. Second, a synchronization has to be done among different processors as soon as their communication buffer fills up.

The partitioned tree construction approach has a better speedup than the synchronous tree construction approach. However, its efficiency decreases as the number of processors increases to 8 and 16. The partitioned tree construction approach suffers from load imbalance. Even though nodes are partitioned so that each processor gets equal number of tuples, there is no simple way of predicting the size of the subtree for that particular node. This load imbalance leads to the runtime being determined by the most heavily loaded processor. The partitioned tree construction approach also suffers from the high data movement during each partitioning phase, the partitioning phase taking place at higher levels of the tree. As more processors are involved, it takes longer to reach the point where all the processors work on their local data only. We have observed in our experiments that load imbalance and higher communication, in that order, are the major cause for the poor performance of the partitioned tree construction approach as the number of processors increase.

The hybrid approach has a superior speedup compared to the partitioned tree approach as its speedup keeps increasing with increasing number of processors. As discussed in Section 3, the hybrid controls the communication cost and data movement cost by adopting the advantages of the two basic parallel formulations. The hybrid strategy also waits long enough for splitting, until there are large number of decision tree nodes for splitting among processors. Due to the allocation of decision tree nodes to each processor being randomized to a large extent, good load balancing is possible. The results confirmed that the proposed hybrid approach based on these two basic parallel formulations is effective.

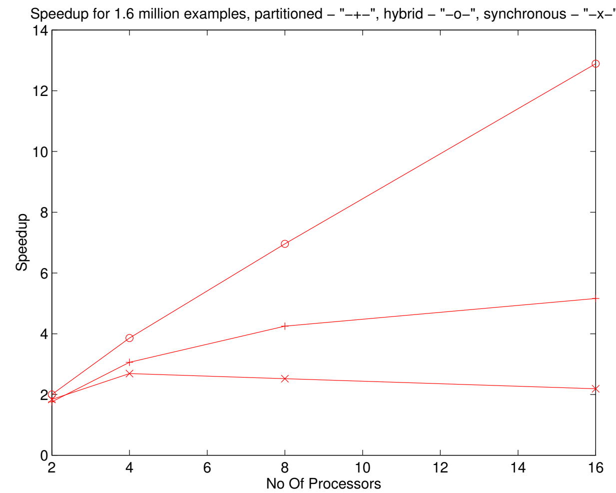

Figure 5. Speedup comparison of the three parallel algorithms.

We have also performed experiments to verify that our splitting criterion of the hybrid algorithm is correct. Figure 6 shows the runtime of the hybrid algorithm with different ratio of communication cost and the sum of moving cost and load balancing cost, i.e.,

$$ratio = \frac{\sum(\text{Communication Cost})}{\text{Moving Cost} + \text{Load Balancing}}.$$

The result was obtained with 1.6 million examples on 16 processors. We proposed that splitting when this ratio is 1.0 would be the optimal time. The results verified our hypothesis as the runtime is the lowest when the ratio is around 1.0. As the splitting decision is made farther away from the optimal point proposed, the runtime increases significantly.

The experiments on 16 processors clearly demonstrated that the hybrid approach gives a much better performance and the splitting criterion used in the hybrid approach is close to optimal. We then performed experiments of running the hybrid approach on more number of processors with different sized datasets to study the speedup and scalability. For these experiments, we used the original data set with continuous attributes and used a clustering technique to discretize continuous attributes at each decision tree node [16].

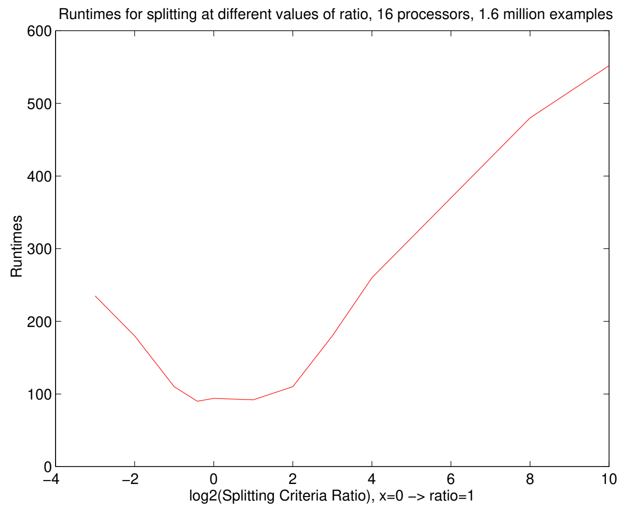

Figure 6. Splitting criterion verification in the hybrid algorithm.

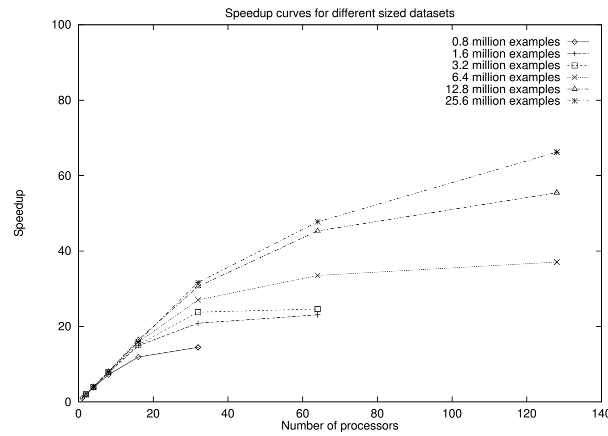

Figure 7. Speedup of the hybrid approach with different size datasets.

Note that the parallel formulation gives *almost identical* performance as the serial algorithm in terms of accuracy and classification tree size [16]. The results in Figure 7 show the speedup of the hybrid approach. The results confirm that the hybrid approach is indeed very effective.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Eng.*, 5(6):914–925, December 1993.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Monterrey, CA, 1984.

[3] M. Chen, J. Han, and P. Yu. Data mining: An overview from database perspective. *IEEE Transactions on Knowledge and Data Eng.*, 8(6):866–883, December 1996.

[4] D. S. D. Michie and C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[5] U. Fayyad, G. Piatetski-Shapiro, and P. Smith. From data mining to knowledge discovery: An overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI/MIT Press, 1996.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimizations and Machine Learning*. Morgan-Kaufman, 1989.

[7] E. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proc. of 1997 ACM-SIGMOD Int. Conf. on Management of Data*, Tucson, Arizona, 1997.

[8] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs (position paper). In *Proc. of the Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 9–13, Tucson, Arizona, 1997.

[9] M. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proc. of the International Parallel Processing Symposium*, 1998.

[10] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(22), April 1987.

[11] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. of the Fifth Int'l Conference on Extending Database Technology*, Avignon, France, 1996.

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

[13] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the 22nd VLDB Conference*, 1996.

[14] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the Fifth Int'l Conference on Extending Database Technology*, Avignon, France, 1996.

[15] A. Srivastava, E.-H. H. V. Singh, and V. Kumar. Parallel formulations of decision-tree classification algorithms. *http://www.cs.umn.edu/∼kumar*, 1998.

[16] A. Srivastava, V. Singh, E.-H. Han, and V. Kumar. An efficient, scalable, parallel classifier for data mining. Technical Report TR-97-010,http://www.cs.umn.edu/∼kumar, Department of Computer Science, University of Minnesota, Minneapolis, 1997.