A Parallel Decision Tree Builder for Mining Very Large Visualization Datasets

Kevin W. Bowyer, Lawrence O. Hall, Thomas Moore, Nitesh Chawla
University of South Florida / Tampa, Florida / 33620-5399
kwb, hall, tmoore4, chawla @csee.usf.edu

W. Philip Kegelmeyer
Sandia National Labs / P.O. Box 969, MS 9214 / Livermore, CA / 94551-0969
wpk@ca.sandia.gov

Abstract

Simulation problems in the DOE ASCI program generate visualization datasets more than a terabyte in size. The practical difficulties in visualizing such datasets motivate the desire for automatic recognition of salient events. We have developed a parallel decision tree classifier for use in this context. Comparisons to ScalParC, a previous attempt to build a fast parallelization of a decision tree classifier, are provided. Our parallel classifier executes on the "ASCI Red" supercomputer. Experiments demonstrate that datasets too large to be processed on a single processor can be efficiently handled in parallel, and suggest that there need not be any decrease in accuracy relative to a monolithic classifier constructed on a single processor.

1 Purpose

The DOE's Accelerated Strategic Computing Initiative (ASCI) program [1] currently generates 3D high-resolution physics data sets in the terascale size range. These simulations replace important physical experiments, and so must be conducted in exacting detail. Domain experts manually visualize the data sets to look for salient events and anomalies. Unfortunately, due to the size of the data sets, it is essentially impossible to exhaustively browse them to search for salient events. The purpose of the work described here is to create a classifier that can learn to recognize salient events. New visualization datasets would be "pre-classified," so that only the most salient regions of the dataset need to be browsed manually.

2 Method

Our approach requires that at least part of a dataset be manually labeled with the level of saliency relative to a particular event. A classifier is learned from this labeled data. Additional datasets can then be automatically labeled according to saliency. To be able to handle very large training sets, the classifier is designed to execute on a parallel architecture.

The parallel learning system discussed here works as follows. The training data is split into N subsets, one per node in the parallel system. The subsets can be either disjoint or overlapping; they are all disjoint in the experiments reported in this paper. A learning algorithm is applied by each processor to its training data. The resultant model is saved for later use by any other processor.

In this paper, the learning algorithm used is a version of the that embodied in C4.5 release 8 [2, 3]. Release 8 of C4.5 has significantly improved handling of continuous attributes, which is important for large-scale visualization data sets which will only have continuous attributes. We call our variant of C4.5 "V5." V5 has been updated from C4.5 in two ways. One, it has been ported to run on the ASCI Red parallel supercomputer[1, 13]. Two, it also allows for test results from a validation set to be stored as weights at the leaves of the tree.

To determine the class of an unseen example, the example is applied to each of the N models, resulting in N classifications. In the case of unweighted classifications, a majority vote is taken to determine the class of the tested example. In the case of weighted classifications, the sum of the weights is obtained for each class and the example is assigned to the class with the

0-7803-6583-6/00/$10.00 © 2000 IEEE

greatest weight.

The advantage of our parallel learning algorithm is that it allows each processor to work entirely independently, with no interprocessor communication. Hence it is fully parallelizable, unlike approaches such as pasting bites [4] or boosting [5, 6], each of which requires that proceeding classifiers be built on training sets determined from already built classifiers.

It is possible that a combiner or arbiter [7, 8, 9] could be utilized with an appropriate training set to provide improved accuracy. The cost would be one more sequential step in the training process and a slightly more time-consuming testing procedure.

We have implemented our parallel decision tree builder on the ASCI Red supercomputer, and have run initial experiments using datasets from the UC Irvine repository, datasets from the ScaleParC data generator [10], and an ASCI simulation dataset.

3 Comparison with ScalParC

We have evaluated the ScalParC parallel decision tree builder [10] for comparison to our own work. For large data sets the ScalParC approach promises a single decision tree that is equivalent to the tree that would be built on a single processor [11, 12]. ScalParC is claimed to present a scalable approach to building decision trees on a parallel processor – "detailed analysis of applying this paradigm to the splitting phase shows that the overall communication overhead of the phase does not exceed O(N), and the memory required to implement the phase does not exceed O(N/p) per processor" and "ScalParC could classify 6.4 million records in just 77 seconds on 128 processors. This demonstrates that large classification problems can be solved quickly using ScalParC" [10]. Here N is the number of examples in the training set and p is the number of processors.

We modified the original ScalParC implementation obtained from the authors so that it would run on the ASCI Red, print a description of the decision tree that it constructs, and optionally allow the use of the information gain metric [3] in order to facilitate comparisons with C4.5 release 8. In analyzing the results of experiments with ScalParC, we discovered that it constructs incorrect decision trees in some cases. The problem can be corrected, but it appears that the solution would necessarily have a substantial negative impact on scalability. We have not pursued development of a corrected implementation of ScalParC. As a result, we currently cannot make useful execution time comparisons to ScalParC.

Table 1: Description of Synthetic Data Set.

example	attr. 1	attr. 2	attr. 3	attr. 4	class
1	0.0	1.0	1.0	1.0	0
2	1.0	1.0	1.0	1.0	1
3	2.0	1.0	1.0	1.0	2
4	3.0	1.0	1.0	1.0	3
5	4.0	1.0	1.0	1.0	4
6	5.0	1.0	1.0	1.0	5

The problem in the ScalParC implementation reveals itself as a tree with "impossible splits." These are splits that logically cannot occur given the earlier splits in the tree. This problem is caused by the algorithm not correctly maintaining its parallel hash table.

The problem in the trees created with ScalParC can be observed in a simple synthetic data set with 6 examples, distributed in 6 classes, with each example having 4 attributes. Consider the synthetic data set described in Table 1, and compare the C4.5-produced and ScalParC-produced decision trees illustrated in Figures 1 and 2, respectively.

The C4.5-produced decision tree can readily be verified to provide a solution to the classification problem. However, the ScalParC-produced decision tree contains two impossible splits. These are identified by the dashed boxes in Figure 2. They are inconsistent with splits occurring earlier in the tree. They are also incorrect attribute splits for the labels of the associated leaves, where only 1 example exists and it cannot correctly contain the label given. For example, the first impossible split labels its leaf as class 1 with a split of *Attribute* $1 < 1.00$. The class 1 example has *Attribute* $1 \geq 1$, so it would not exist at this leaf. If the information gain metric is used instead of the Gini metric, then on this example data set the program effectively hits an infinite loop. All of these symptoms trace to the same underlying problem – the maintenance of the parallel hash table during the computation. Effectively, a child node does not get the set of data items that the parent node intends to pass on as a result of the split. The correct number of examples are received, but not the intended set of examples.

The correct set of examples must be indexed at tree nodes as splitting proceeds. This appears to require changes to the data structures used to keep track of the tree. One possible approach is used in Sprint [12]. This solution produces trees which match those generated sequentially, but has been shown to be unscalable [10]. For any possible solution, more inter-processor

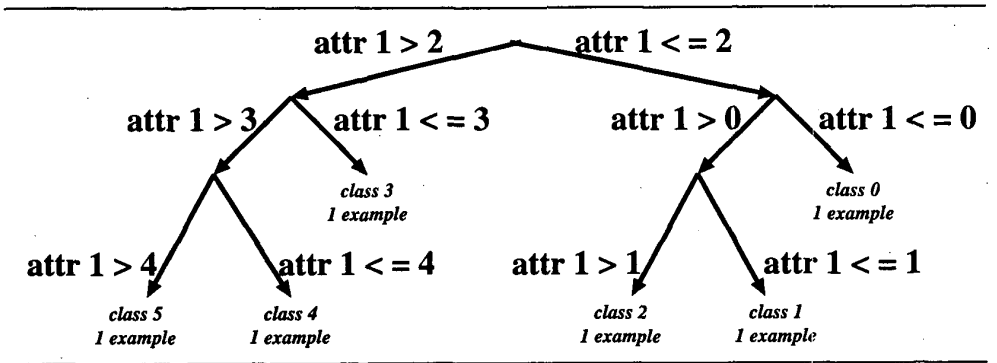

Figure 1: Depiction of the C4.5 tree resulting from the synthetic data set.

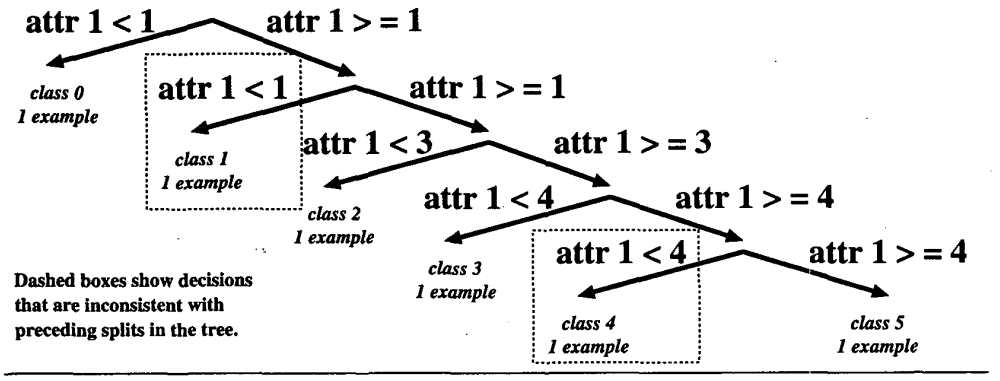

Figure 2: Depiction of the ScalParC tree resulting from the synthetic data set.

communication will be required and so scalability will be adversely affected.

4 Results

Results of this project to date fall into four areas:

- We have modified the C4.5 release 8 implementation to run on the ASCI Red parallel computer, and to maintain data for use in weighted voting among classifiers.
- We have modified the MUSTAFA visualization tool to allow labeling of selected nodes at selected time steps of a simulation with a user-specified saliency. MUSTAFA is a visualization tool developed by the Department of Energy. It is built on top of the commercial AVS Express visualization tool. MUSTAFA is used and Sandia labs in the visualization of 4-D physics simulation datasets. Our modifications to MUSTAFA are intended to allow the user to create labeled training data sets.
- We have conducted empirical experiments to compare the accuracy of a classifier built in parallel from subsets of data with that of a classifier built on one processor from all of the data. Using 15 different data sets from the UC Irvine repository, results suggest that on data sets small enough to also be processed on a single machine, our parallel decision tree approach is able to achieve essentially the same accuracy as a tree grown on all the data. See [14] for more details.
- We have conducted experiments to evaluate the speedup possible from creating classifiers on large learning sets in parallel on the ASCI Red supercomputer. The remainder of this section gives an overview of selected results of learning a classifier on large data sets using the ASCI Red.

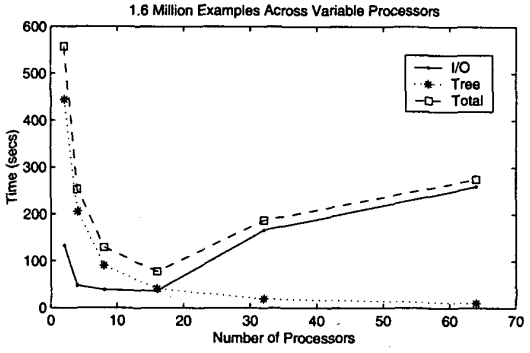

Figure 3: Learning from 1,600,000 examples split across 2, 4, ... 64 ASCI Red processors.

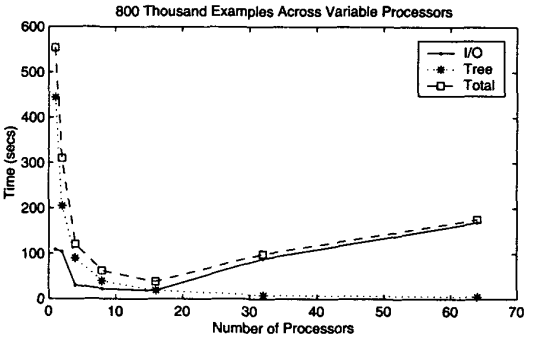

Figure 4: Learning from 800,000 examples split across 2, 4, ... 64 ASCI Red processors.

The ASCI Red is a supercomputer belonging to the U.S. Department of Energy [1]. It has a total of 4,640 "compute nodes," each of which contains two 333-MHz Pentium III processors sharing 256MB of memory. The processors run a version of the UNIX operating system. The system is based on a distributed-memory mesh architecture, and is capable of 3.15 TeraFLOPS.

To get a feel for the tradeoffs involved in creating a classifier in parallel, consider an experiment that looks at speedup versus number of processors for a fixed-size dataset. The data set used in this experiment was created using the same synthetic data generator used in the ScalParC project [10]. Using a synthetic data generator allows us to easily generate arbitrarily large training data sets. The synthetic data represents a two-class, seven-attribute problem, with each of the attributes being continuous-valued.

Figure 3 summarizes the speedup results for a 1.6 million example training set, broken across from 2 to 64 processors. The execution time for one processor does not appear in this Figure because the problem was too large to run successfully on a single processor (1.6 million examples, with seven real values and one discrete value each). The maximum speedup is approximately 14, obtained using 16 processors. The I/O time (time to read data into the compute nodes and write resultant trees to disk), tree-building time (on the compute nodes), and the total time are shown. The tree-building time is the time required for the slowest of the N processors to finish building and pruning its tree on its subset of the training data. Despite the fact that loading the data is a one-time cost, and that there is no subsequent interprocess communication, the I/O time steadily increases and becomes the dominant time. This reflects the fact that the initial loading of data into the processors is a complex task on this parallel system. The individual compute nodes do not each have their own independent channel to the disk storage. Once the four channels shared by the compute nodes are all being used, the communication to the compute nodes becomes contentious. When a program is run on a subset of the ASCI Red compute nodes while other programs run on other nodes, the observed I/O time is often affected by the I/O activity of the other programs. For this reason, the compute times plotted in Figures 3 and 4 are an average over 10 runs of the same computation, and those plotted in Figure 5 are an average over 4 runs.

If we use a data set one half the size, that is, 800,000 examples, then we observe a similar pattern of results, but the problem is small enough to also run on a single processor. Results of this experiment appear in Figure 4. The overall time decreases up to 16 processors, which results in a speedup of 14.5 times over sequential learning on one ASCI Red processor.

Another way of looking at the effectiveness of the parallel system is in terms of how well a larger data set can be handled by simply using more processors. Figure 5 shows timings on the ASCI Red using between 1 and 64 processors, always with 800,000 examples per processor. Thus at 64 processors, it takes only about 2.5 hrs to train on 51,2000,000 examples! The total time is affected mostly by I/O time, which increases approximately linearly way with the number of processors.

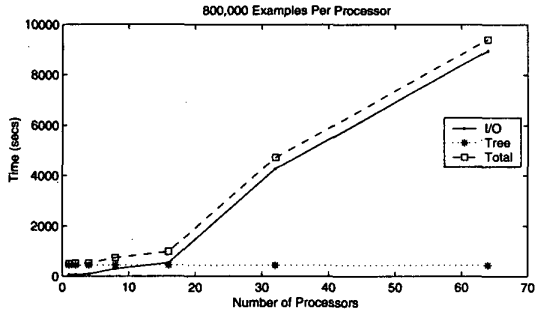

Figure 5: Experimental results from learning on 800,000 generated examples per processor on a varying number of ASCI Red processors from 1 to 64. There are 51,200,000 training examples with 64 processors.

5 Conclusions and Future Work

We are able to effectively create a classifier in parallel on the ASCI Red supercomputer. The accuracy to be expected from the classifier created in parallel is roughly equivalent to that of a single classifier that could in principle be created from the complete dataset on a single machine [14]. This work is novel in (a) being aimed at data mining on datasets so large that they can only be handled on supercomputers such as the ASCI Red, and (b) combining classifiers created in parallel on distinct subsets of a dataset.

Several important issues remain to be considered in this research project. One important topic of continuing work involves the creation of labeled training data. One issue in this is how to make the process of labeling the data as effortless as possible for the domain expert. Another issue is the amount of training data actually needed to create an accurate classifier. A complete, relevant ASCI data set may be a terabyte in size, but it seems likely that not all of the data set needs to be labeled in order to learn a useful classifier. One possibility is to have the learning tool guide the labeling of training data. For example, the learning tool might suggest a minimum number of data elements to be labeled for each saliency category, then tentatively classify the remainder of the data set, and then present the least-certain classifications to the user for confirmation. If the user corrects tentative classifications, the cycle is repeated.

Another important topic of continuing work involves the design and evaluation of classifiers in the presence of highly-unbalanced training sets. The real-world datasets of interest in this work are highly unbalanced between classes. An extremely large percentage (higher than 99.9%) of the visualization data set is relatively uninteresting, and only a very small percentage is highly interesting. Additionally, the cost of misclassifying a highly interesting element is much greater than that of mis-classifying an uninteresting element. In this situation, it is appropriate to design and evaluate classifiers using tools such as the area under the ROC curve rather than the average accuracy. One popular approach to dealing with unbalanced training data in classifier design is to under-sample the majority class in the training data. Under-sampling by various factors will effectively sweep out an ROC curve. We are exploring methods of combined manipulation of both the minority and majority classes to create classifiers that achieve a greater area under the ROC curve.

Another topic currently being explored is the use of a bagging-like approach to improving classifier performance. Bagging traditionally involves selecting a large fraction (e.g., 80%) of the available data N times, with replacement, to generate individual classifiers whose decisions are combined with voting [15, 5]. Bagging can result in improved performance over a single classifier constructed with all of the training data. In our application, it is not feasible to select a large fraction of the training data for use at each processor. Therefore, we want to determine if the effect of bagging will hold up when the size of the individual bags is a small fraction of the training data.

Lastly, we hope to conduct experiments to evaluate the effectiveness of pre-classifying a data set for visualization. Ideally, we would have two similar manually-labeled visualization data sets. They could be used as half-half training and test divisions of data. The practical question from the standpoint of the visualization domain expert is how accurately the classifier built on one data set can predict the saliency labels for the other data set. Assuming that acceptable accuracy can be achieved using the whole training set, then the relevant question is how little of the data set is actually required to achieve this level of accuracy.

Acknowledgments

This work was partially supported in part by the United States Department of Energy through the Sandia National Laboratories LDRD program and ASCI VIEWS Data Discovery Program, contract number DE-AC04-76DO00789.

References

[1] Sandia National Labs, http://www.sandia.gov/ASCI/Red/UserGuide.htm, *ASCI Red Users Manual*, 1997.

[2] J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992. San Mateo, CA.

[3] J. Quinlan, "Improved use of continuous attributes in C4.5," *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

[4] L. Breiman, "Pasting bites together for prediction in large data sets," *Machine Learning*, vol. 36, no. 1,2, pp. 85–103, 1999.

[5] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine Learning*, vol. 36, no. 1,2, 1999.

[6] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proceedings of the Thirteenth National Conference*, pp. 148–156, 1996.

[7] P. Chan and S. Stolfo, "Sharing learned models among remote database partitions by local meta-learning," in *Proceedings Second International Conference on Knowledge Discovery and Data Mining*, pp. 2–7, 1996.

[8] P. Chan and S. Stolfo, "On the accuracy of meta-learning for scalable data mining," *Journal of Intelligent Information Systems*, vol. 8, pp. 5–28, 1997.

[9] P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection," in *Proc. KDD-98*, 1998.

[10] M. V. Joshi, G. Karypis, and V. Kumar, "Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets," in *Proceedings of the International Parallel Processing Symposium*, pp. 573–579, 1998.

[11] S. Goil and A. Choudhary, "Efficient parallel classification using dimensional aggregates," in *Proceedings of the Workshop on Large-Scale Parallel KDD Systems*, Tech. Report 99-8, Rennsselaer Polytechnic Institute, Troy, NY. 1999., 1999

[12] J. Shafer, R. Agrawal, and M. Mehta, "Sprint: A scalable parallel classifier for data mining," in *Proceedings of the 22nd VLDB Conference*, 1996.

[13] Sandia National Labs, http://www.sandia.gov/ASCI/Red/main.htm, *ASCI Red - The World's First TeraOps Super-Computer*, May 25, 2000.

[14] L. Hall, K. Bowyer, W. Kegelmeyer, T. Moore, and C. Chao, "Distributed learning on very large data sets," in *ACM SIGKDD Workshop on Distributed and Parallel Knowledge Discovery*, 2000.

[15] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.