Free Parallel Data Mining

Bin Li
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
binli@cs.nyu.edu

Dennis Shasha
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
shasha@cs.nyu.edu

Abstract

Data mining is computationally expensive. Since the benefits of data mining results are unpredictable, organizations may not be willing to buy new hardware for that purpose. We will present a system that enables data mining applications to run in parallel on networks of workstations in a fault-tolerant manner. We will describe our parallelization of a combinatorial pattern discovery algorithm and a classification tree algorithm. We will demonstrate the effectiveness of our system with two real applications: discovering active motifs in protein sequences and predicting foreign exchange rate movement.

1 Introduction

We will demonstrate a system that enables data mining applications to run on existing hardware without interfering with normal applications. The underlying technology of our system is Persistent Linda (PLinda) [1, 4]. PLinda follows the Linda paradigm [3], providing a set of language constructs to facilitate writing parallel programs. In addition, PLinda automatically harnesses idle cycles on networks of workstations (NOW). Furthermore, PLinda uses "lightweight transactions" and "continuation committing" to achieve fault-tolerance.

In the following sections, we will first give a brief overview of PLinda. We will then discuss the computation models of two data mining applications and describe our parallel implementation on the PLinda platform. We also give performance figures from our experiments. Finally, we outline our demonstration and explain how to obtain our software.

2 Overview of PLinda

Linda provides a set of extensions that can be added to any programming language to facilitate writing parallel programs in that language. PLinda adds a set of extensions to Linda to support high-performance fault-tolerant parallel computation on intermittently idle, heterogeneous workstations. The three major extensions PLinda offers are:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '98 Seattle, WA, USA
© 1998 ACM 0-89791-995-5/98/006...$5.00

- Lightweight transactions. In PLinda, a process is executed as a series of transactions. If a process fails while executing a transaction, the runtime server detects the failure and aborts the transaction automatically. Runtime mechanisms ensure that no other processes can access uncommitted modifications to the tuple space, so the aborted transaction has no effect on other transactions. These transactions are lightweight in the sense that they do not commit their modifications to stable storage. This is acceptable for data mining because we care only about the final result. Our transaction mechanism *guarantees that a completed PLinda computation, with or without failures, achieves the same final state as a failure-free execution of the associated Linda program*, where the associated Linda program is identical to the PLinda program but without the transaction and recovery statements. (The very similar Paradise [ref] model does not quite achieve this guarantee.) A transaction starts with the `xstart` command and ends with the `xcommit` command.

- Continuation committing. The `xcommit` operation takes a tuple as a parameter. This tuple is used to save local state, known as a *continuation*. A continuation consists of the live variables of a process and an indication of which transaction in a process last successfully completed. Continuation committing is used to allow processes to migrate from one site to another.

- Checkpoint-protected tuple space. The PLinda runtime server manages the entire tuple space and saves it to disk periodically—*checkpointing*. If the tuple space server fails, it restores the latest checkpointed state from disk on recovery and resumes execution from that state.

The PLinda system automatically detects idle workstations on the network and starts PLinda processes on these machines. PLinda models non-idleness (e.g. user returns to workstation) as failure. In this way, PLinda programs can run in a manner that is fault-tolerant and does not disturb the owners of the workstations at all. (We have tested this empirically.) This is a suitable platform on NOW for long-running parallel computations such as data mining applications.

3 Combinatorial Pattern Discovery

Combinatorial databases store structures such as sequences, trees, and graphs. Combinatorial pattern discovery is to find

patterns in such a database that each approximately characterizes a set of structures given a pattern metric. Finding active motifs in sets of protein sequences is an example of combinatorial pattern discovery. Consider a database of imaginary protein sequences $\mathcal{D}$={FFRR, MRRM, MTRM, DPKY, AVLG} and the query "Find the patterns P of the form $*X*$ where P occurs in at least 2 sequences in $\mathcal{D}$ and the size of P $|P| \geq 2$." (X can be a segment of a sequence of any length, and $*$ represents a variable length don't care.) The good patterns are *RR* (which occurs in FFRR and MRRM) and *RM* (which occurs in MRRM and MTRM).

Pattern discovery in sets of sequences concerns finding commonly occurring subsequences (sometimes called *motifs*). The structures of the motifs we wish to discover are regular expressions of the form $*S_1 * S_2 * \ldots$ where $S_1, S_2, \ldots$ are *segments* of a sequence, i.e., subsequences made up of consecutive letters and $*$ represents a variable length don't care (VLDC). In matching the expression $*S_1 * S_2 * \ldots$ with a sequence S, the VLDCs may substitute for zero or more letters in S. Segments may allow a specified number of mutations; a mutation is a insertion, a deletion, or a mismatch [6].

The computation model for sequence pattern discovery follows a generate-and-test paradigm—generate a candidate pattern, then test whether it is active. Furthermore, there is some interdependence among the patterns that gives rise to pruning; if a pattern occurs too rarely, then so will any of its superpatterns. For example, if *R* is not an active pattern, neither *FR* nor *RF* will be.

The PLinda implementation of sequence pattern discovery consists of a master and a worker. A worker calculates the number of occurrences of a pattern and determines whether it is active. When a pattern is active then the same worker or other ones will explore its superpatterns. The master coordinates the whole computation and collects the results.

4 Bounded Optimal Split Classification Trees

Classification trees are a method of predicting the value of a dependent attribute, say whether to buy or sell Yen, based on independent attributes, say the history of exchange rates between Yen and U.S. Dollar over the last few days or the last few years. Building a classification tree entails the hypothesis that the historical relationship between independent and dependent attributes will remain relevant in the future. An algorithm for constructing classification trees based on historical data is called a *classification tree algorithm*. NyuMiner is a classification tree algorithm we have developed based on CART [2] and C4.5 [5]. It generates classification trees with optimal multi-branch splits on independent attributes, which can take either categorical or numerical values.

The core of any classification tree algorithm is the process of building a full-grown tree by *splitting*—attaching child nodes to a node n in which the data elements in the children of n partition the data elements in n and they are more "pure". (Each data element is a record containing the values of both independent and dependent attributes at some point in history.) A set of data elements is pure if every element has the same value of the dependent attribute. An impure set is one in which there is an equal distribution of dependent attributes among the elements. The leaves of a classification tree should be pure.

At each tree node, each unused independent attribute is considered a potential splitting attribute. And each potential splitting attribute may have an infinite number of possible splits, for each of which an impurity score is calculated. (The number of possible splits can be limited by the maximum number of branching allowed.) Generating all possible splitting points and calculating impurity scores is computationally expensive, but fortunately can be done in parallel. The best attribute/split is chosen to partition the data elements into sub-nodes. This goes on until all leaves of the tree are pure.

Like the windowing technique used in C4.5, NyuMiner employs a method called "multiple incremental sampling". It starts with a randomly selected subset of data elements, which are used to build an initial classification tree. This tree is then used to classify the remaining data elements (those that are not selected); usually some are misclassified. A selection of these "difficult" data elements is then added to the initial training set. This enlarged training set is used to build a second tree, which is again tested on the remaining data elements. The cycle is repeated until a tree from the current training set correctly classifies all the remaining data elements or all data elements are used to build the tree. The final training set, usually a small portion of the data elements, can be thought of as a screened set of data elements that contains all the ones needed to guide the tree-building.

Different initial training sets generally lead to different initial trees, which usually have different resubstitution error rates. (The resubstitution error rate of a classification tree is the percentage of misclassified data elements.) Therefore, we grow several alternative trees in parallel and select the one with the smallest resubstitution error rate. The PLinda implementation of NyuMiner includes a master, a tree builder, and a worker. A worker calculates impurity scores for all possible splits for a given attribute on a tree node. A tree builder uses workers' service to build a complete tree with a particular initial training set. The master starts as many as tree builders as user specifies and chooses the best tree from one of them.

We have used this system to analyze exchange rates among several major currencies from the past 27 years. We have discovered rules that can predict the next day's price movement with confidence (and make a profit). Other applications include:

- Predicting the cellular localization sites of proteins.
- Predicting whether annual income exceeds $50K based on census data.
- Predicting people described by a set of attributes as good or bad credit risks.
- From multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, classifying the central pixel in each neighbourhood.
- Predicting whether mushroom is poisonous or edible based on attributes of physical characteristics.

5 Experiments

Figure 1 shows the experiment results of our sequence pattern discovery program running on 5, 10, 15, 20, 25, 30, 35, 40, and 45 Sun Sparc 5 workstations. These experiments were run after 5pm at a major research lab. Good speedup is achieved even when as many as 45 machines joined the computation. For 15 and fewer machines, the speedup is particularly good.

Our experiments with parallel NyuMiner show similar speedup and scalability results.

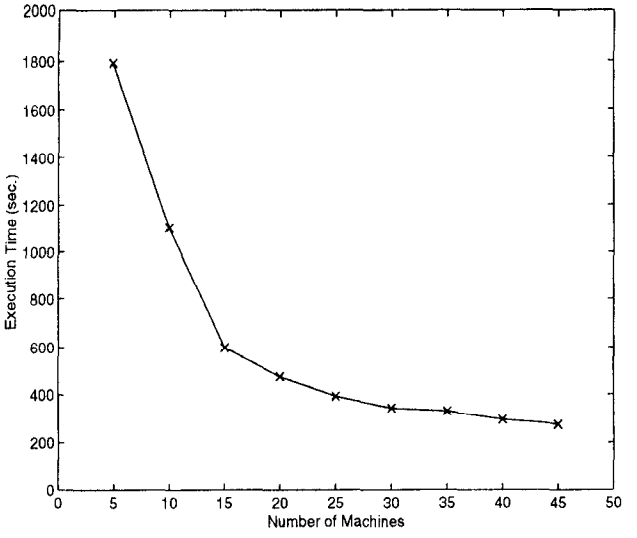

Figure 1: Running time of our parallel sequence pattern discovery program on 5, 10, 15, 20, 25, 30, 35, 40, and 45 machines. Sequential running time is 4686 seconds.

6 The Demo

We plan to bring a small network of PC workstations to SIGMOD. Some will run Linux and some will run Solaris. PLinda will be pre-installed on these machines. We will demonstrate our parallel protein sequence discovery program and our parallel NyuMiner program on this small network. Speedups of these programs versus their sequential versions can be observed. We will also simulate machine failures and non-idleness during executions and compare final results with ones from failure-free executions.

All software used in the demonstration will be freely available, including source code. For PLinda software, please download from http://merv.cs.nyu.edu:8001/~binli/plinda/. For the combinatorial pattern discovery software and the NyuMiner software, please download from http://merv.cs.nyu.edu:8001/~binli/datamining/.

Acknowledgments

The authors would like to thank Chin-Yuan Cheng, James C. Lin, Jason Tsong-Li Wang, and the anonymous referees for their help.

References

[1] B. Anderson and D. Shasha. Persistent Linda: Linda + transactions + query processing. In J. P. Banatre and D. Le Metayer, editors, *Research Directions in High-Level Parallel Programming Languages*, volume 574, pages 93–109. Springer-Verlag Lecture Notes in Computer Science, June 1992.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

[3] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[4] K. Jeong, D. Shasha, S. Talla, and P. Wyckoff. An Approach to Fault Tolerant Parallel Processing on Intermittently Idle, Heterogeneous Workstations. *The Twenty-Seventh International Symposium on Fault-Tolerant Computing*, pages 11–20, Seattle, Washington, June 24–27, 1997.

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

[6] J. T. L. Wang, G.-W. Chirn, T. G. Marr, B. A. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 115–125, Minneapolis, Minnesota, May 1994.