

# Data Mining on Parallel Database Systems

Mauro Sousa

Marta Mattoso

Nelson Ebecken

COPPE/UFRJ - Federal University of Rio de Janeiro  
P.O. Box 68511, Rio de Janeiro, RJ, Brazil, 21945-970  
Fax: +55 21 2906626

**Abstract** *Recent years have shown the need of an automated process to discover interesting and hidden patterns in real-world databases, handling large volumes of data. This sort of process implies a lot of computational power, memory and disk I/O, which can only be provided by parallel computers. Our work contributes with a solution that integrates a machine learning algorithm, parallelism and a tightly-coupled use of a DBMS system, addressing performance problems with parallel processing and data fragmentation.*

*Keywords:* data mining, parallel processing, parallel database server

## 1 Introduction

With recent explosion of information, and the availability of cheap storage, it has been possible to gather massive data during the last decades. The eventual goal of this massive data gathering is the use of this information to gain competitive advantages, by discovering previously unknown patterns in data, which can guide the decision making.

Recent years have shown that it is becoming difficult to analyze data using only OLAP tools, showing the need of an automated process to discover interesting and hidden patterns in data. Data mining techniques have increasingly been studied [1, 2], especially in their application in real-world databases. One typical problem is that databases tend to be very large,

and these techniques often repeatedly scan the entire set. A solution that has been used for a long time is the extraction of a training sample, so that the data mining algorithm could be run on a smaller portion of data. In this case, subtle differences among sets of objects in the database will become less evident.

The need to handle large amounts of data implies a lot of computational power, memory and disk I/O, which can only be provided by parallel computers.

A variety of data mining algorithms have been proposed addressing parallel processing. Most of them use flat files and exploit parallelism through parallel programming.

Although the idea of using a DBMSs is not new, most implementations use databases only to issue queries that are going to be processed by a client machine. Such an example is the SKICAT system [3], which uses a relational DBMS (Sybase) for catalog store, modification and management. It means that most of the current data mining applications have a loose connection with databases, thus resulting in poor performance [4].

Quest data mining system [5] has been addressing the development of fast, scalable algorithms. Quest algorithms have been parallelized to run on IBM's shared-nothing multiprocessor SP2. We can cite the parallel implementation of the mining of association rules (APRIORI algorithm) and the SPRINT classification algorithm - an evolution of SLIQ - where multiple processors are able to work together to construct a classification model, namely a decision tree. These algorithms run

against data in flat files as well as DB2 family of database products, but databases are accessed in a loosely-couple mode using dynamic SQL.

Freitas [6] has presented a series of primitives for rule induction, after analyzing many KDD algorithms of the RI paradigm. His conclusion is that the core operation of their candidate rule evaluation procedure is a primitive he calls *Count by Group*. This primitive can be translated to usual *Group By* clauses in SQL statements, which often provide a potential degree of parallelism in parallel DBMSs.

Our work contributes with a solution that integrates a machine learning algorithm, parallelism and a tightly-coupled use of a DBMS system. Most of current data mining applications have a loosely-coupled integration with DBMSs. We highlight the advantages of close integration of data mining techniques to database systems, addressing performance problems with parallel processing and data fragmentation. To validate our experiments, we used Oracle Parallel Server, configured to run on a four-node IBM RS/6000 SP2 system. We have constructed a set of stored procedures written in PL/SQL, which are stored within Oracle data dictionary, providing high performance and native integration to the database.

The remainder of this work is organized as follows. Section 2 presents advantages and shortcomings of implementing a data mining algorithm in parallel databases. Section 3 describes some characteristics of the algorithm used, and section 4 presents implementation issues, such as the development of tightly-coupled routines and special features of DBMS exploited. Finally, section 5 presents our conclusions and observations.

## 2 Opportunities for Parallel Databases in Data Mining Algorithms

There are a variety of data mining algorithms constructed to run in parallel, taking advantage of parallel architectures using specific pro-

gramming routines. On the other hand, our implementation is based on parallel database systems. There are many advantages of this approach, and we cite some here:

- *The implementation becomes simpler.* There is no need to use parallel routines such as MPI libraries. The parallel DBMS is responsible itself for parallelizing queries that are issued against it. We have to structure our queries so that they can fully exploit parallelism offered by a particular DBMS.
- *DBMS can choose between serial and parallel execution plans transparently to application.* Depending on the data mining algorithm and technique used (classification, clustering, association rules, etc.), it is clear that parallelism is more useful in the beginning and intermediate phases of the process. In the construction of decision trees, for example, the training set is recursively partitioned and processed. Thus, at some point of the processing, parallelism may not be useful because the communication overhead between coordinator and slave processes does not compensate, because there are not many training examples being searched.
- *It is possible to work with data sets that are considerably larger than main memory.* We have to concentrate our efforts in trying to translate most of the data mining algorithm into SQL statements. If data does not fit in memory, the database itself is responsible for handling information, paging and swapping when necessary.
- *Simplified data management.* A DMBS itself has its own tools to manage data.
- *Data may be updated or managed as a part of a larger operational process.* Data may become available using transformation techniques in data warehouses, or it can be the dataware itself.

- *Extensibility to mine complex data types.* This characteristic is becoming more realistic as emerging object-relational databases are providing the ability to handle image, video, voice and other multimedia objects.
- *Ad-hoc and OLAP queries can be used to validate discovered patterns.* Queries can be issued to verify rules discovered by the data mining process, providing means to easily evaluate results. Ad-hoc queries and OLAP can be used to complement the whole process.
- *Opportunity for database fragmentation.* Database fragmentation provides many advantages related to reduced number of page accesses necessary for reading data. Irrelevant data could be just ignored depending on the filter specified by SQL statements. Furthermore, some DBMSs can process each partition in parallel, using different execution plans when applicable. In general, DBMS systems provide automatic management of such functionality.

However, some drawbacks have to be considered, such as:

- *Less control of how parallelization will occur.* Although there is the possibility of some customizations, such as setting the nodes that will participate in the process, the DBMS itself is in charge for parallelization.
- *The algorithm may become dependent on some characteristic of the parallel DBMS used.* In a parallel database, we have some chance to direct the optimizer to use a specific execution plan, but our algorithm may be dependent on some characteristic of this DBMS.
- *Overhead of the database system kernel.* A kernel of a database system is projected to handle a large set of operations, such as OLTP transactions, OLAP

queries, etc. Although it is possible to minimize the overhead and customize the environment to take the best available advantages of the corresponding architecture, there will always exist some functionality implemented that is not applicable to the data mining algorithm, which can degrade performance when compared to access to flat files.

## 3 The Database Mining Model

### 3.1 Decision Tree Algorithm

Decision tree methods are a kind of machine learning algorithm that uses a divide-and-conquer approach to classify cases using a tree-based representation [7]. Decision tree classifiers are well-suited for data mining problems, producing similar accuracy when compared to other methods for classification. Besides, they represent one of the most widely used and practical methods for inductive inference, and they can be built considerably faster than other machine learning methods. Another advantage that led us implement this sort of algorithm is that decision trees can be easily constructed from and transformed into SQL statements [8], which can be used to query a database system, particularly a parallel one.

There are many implementations using decision tree learning algorithms, such as CART [9], ID3 and its successor C4.5 [10], and SPRINT (SLIQ's parallel implementation) [11, 12].

### 3.2 Attribute Types

There are two types of attributes to be handled:

- *Continuous Attribute.* We have to analyze each of the  $n - 1$  splitting points, where  $n$  represents the number of distinct values. This calculation may constitute a time consuming task when there are too many

distinct values for an attribute. Hence, a discretization is an advisable approach.

- *Discrete Attribute.* There are several approaches to deal with this kind of attribute. We can create one subtree for each distinct value, two subtrees and segment values into these two groups, create how many subtrees are necessary or treat a discrete attribute as it was a continuous one. When there are some distinct values, a greedy algorithm is often applied to divide values into groups.

### 3.3 Splitting Point

A decision tree classifier interactively subdivides the training set until each partition represents cases totally or dominantly belonging to the same class. In order to partition training data, a statistical test is used as the splitting criteria. The test we implemented is one called gini index. For a data set  $T$  containing examples from  $n$  classes,  $gini(T)$  is defined as

$$gini(T) = 1 - \sum_{j=1}^n p_j^2$$

where  $p_j$  is the relative frequency of class  $j$  in  $T$ .

When a split divides  $T$  into two subsets,  $T_1$  and  $T_2$ , with  $n_1$  and  $n_2$  examples respectively, the new index of the divided data  $gini_{split}(T)$  is given by

$$gini_{split}(T) = \frac{n_1}{n}gini(T_1) + \frac{n_2}{n}gini(T_2)$$

The advantage of this measure is that it requires only the class distribution in each partition, involving a relatively simple calculating. Gini index is calculated for each attribute, and the attribute with the less gini index is used to split the current node.

This statistical test is also implemented in other classifiers, such as SPRINT [11]. CART [9] uses a variation of this test called Reduction of Gini Diversity Index.

### 3.4 Pruning

The recursive partitioning method imposed to the training examples continues to subdivide data until all the examples in the partition belong to same class, often resulting in a very complex tree that overfits the data by inferring more structure than is represented by training cases [10]. The main idea is to remove parts of the tree that do not contribute to classification accuracy, producing less complex, and more comprehensible trees.

There are three basic approaches that can be used:

- *Pre-pruning.* At the same time the algorithm is growing the tree, it searches for stopping criteria. The pre-pruning approach is faster but less reliable [9].
- *Post-pruning (Growing and Pruning).* At a first phase, we grow a tree until each leaf represents cases belonging to the same class. This approach is slower but more reliable.
- *Pre-pruning and Post-pruning.* This strategy combines both approaches described before, most of times implementing not an aggressive pre-pruning criteria, but aggressive enough to diminish CPU time needed to process the training examples.

Another problem is the prediction of error rates. Accordingly to Quinlan [10], there are two families of techniques for predicting error rates: one predicts the error rates using a separate set of cases that is distinct from the training set, and the other family uses only the training data from which the tree was built [9].

As we are interested mainly in data mining on large databases, we may assume that we have plenty of data. Thus, we have decided to use the first family of techniques, randomly dividing data into two parts, so that both portions have the same class distribution.

In order to improve parallelism, we have constructed a function in PL/SQL, which is able

to classify a test case based on the decision tree created during the first phase. The function receives the attributes used by the data mining process, reads the entire test set and computes the results in parallel.

```
Select class, Classify(age, car_type,
                       insurance_type)
```

From *mine\_relation\_test*

[Where ...]

Function `Classify` will return the predicted class. Having actual and predicted classes, we can calculate the error rate related to the training set.

The strategy implemented to prune the decision tree is the simplest one that could be used. We start from the bottom of tree, examining each subtree of decision nodes. We consider the replacement of this subtree by a leaf representing the most frequent class at current scope or by the most used branch. The alternative that leads to a smaller error rate prediction drives the pruning of the tree [10]. This process is presented in figure 1.

### 3.5 Rule Extraction

Even after pruning a tree, it can still represent a complex, incomprehensible structure at first glance. Large decision trees are difficult to understand because each node has a specific context established by the outcomes of tests at antecedent nodes [10]. Rule extraction tends to solve this problem, presenting a more compact and simpler representation, as soon as irrelevant expressions are eliminated from the originally generated tree.

In our implementation, for each original rule we interactively remove the most irrelevant condition, if any; that is, we move away the one whose removal most contributes to the accuracy of the classification model.

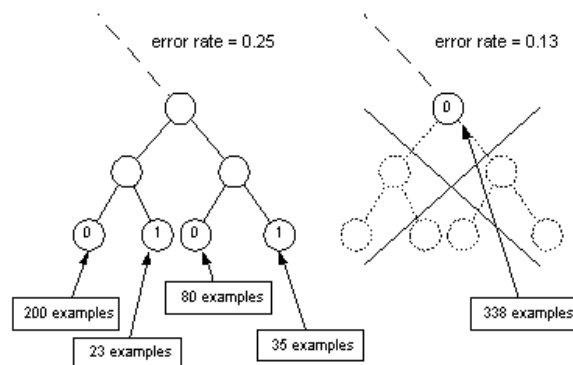


Figure 1: Pruning Phase. The root of the subtree above is being substituted by a leaf node representing the most frequent class at that scope.

## 4 Implementing Database Mining on a Parallel Database

To validate the proposed approach, we have constructed a set of stored procedures (grouped into packages) written in PL/SQL (Oracle's SQL extension) that are stored within Oracle Server data dictionary. PL/SQL is designed to integrate closely with the Oracle database system and with SQL. Therefore, it supports all the internal datatypes native to the Oracle database.

We have been using a four-node IBM RS/6000 SP2 system with Oracle Parallel Server 7.3 installed [13].

### 4.1 Tightly-Coupled Integration

The idea of executing user-defined computation within databases, thus avoiding unnecessary network traffic (figure 2), has been a major concern in many DBMS systems. One example of this sort of implementation is the stored procedure in commercial databases.

Some authors have also developed a methodology for tightly-coupled integration of data mining applications with database systems [4], selectively pushing parts of the application

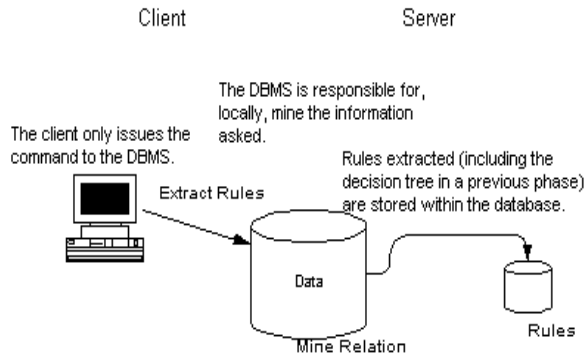


Figure 2: Implemented Approach.

program into the database system, instead of bringing the records into the application. What we experiment is a variation of these methodologies, taking advantage of parallel features available in many parallel database systems, in particular the ones provided by Oracle Parallel Server.

The majority of the data mining algorithms developed uses MPI functions (or similar libraries) to implement parallelism. They divide the processing among nodes of the system, turning the parallelism totally dependent on the constructed application. In PL/SQL this is not possible. Parallelism should be achieved through queries and not through this language, turning the parallelism dependent on the database system. The only way to parallelize a code written in PL/SQL is constructing a stored function, which can be called from within a SQL statement.

## 4.2 Decision Tree

One characteristic of SQL that can degrade the performance of a Rule Induction algorithm, in particular a decision tree one, is that its structure requires a separate query to construct the histogram for each attribute. Thus, it is not possible to perform a SQL statement that can bring us the result reading data just once.

Most of queries issued during the growing phase of a decision tree deals with two kinds of operations: sorts (*Order By*) and groupings

(*Group By*). These kinds of queries represent the time consuming tasks obtained during the implemented algorithm.

```
Select salary, class, count(*)
From mine_relation
Group By salary, class
Order By salary
```

Alternatively, we can use several tables and perform joins to process them, but this operation is very expensive, even when processed in parallel.

Parallel database servers can easily process the above query, and it is possible to take advantage of the resources of parallel machines. In MPP architectures, for example, each node can independently compute counts on the records stored locally. In the next phase, a coordinator process can consolidate these counts, and then return the result back to the user [14].

A typical approach used that allows user queries to deal with smaller relations, causing a smaller number of page accesses, is the vertical fragmentation. In the case of decision trees, it would be necessary to create one relation for each pair attribute/class (remember that decision trees work with a list of attributes, in each phase trying to find the best splitting point among all attributes), as it is done with flat files in SPRINT [11].

Our tests have shown that vertical fragmentation is useful mainly during the first step of this algorithm, when there is no splitting point yet defined; each query is made of a simple group by SQL statement, with no filtering.

```
Select attri, class, COUNT(*)
From mine_relation
Group By attri, class
Order By attri.
```

In the second phase, at most a two-table join is needed; in the third one, at most a three-table join, and so forth. Joins in DBMSs constitute a time consuming task, even when solved in parallel. In order to provide a better response time for the first phase, we suggest the construction of concatenated indexes

(attribute, class). Since the mentioned query only needs these two columns, it can perform a parallel full index scan, without referencing the original table.

### 4.3 Splitting Point

In order to provide a close integration to the database, the construction of user-defined functions that can be called from within SQL statements is advisable. It decreases the number of calls to the database, and makes our code considerably simpler.

```
Select min(giniSplit(attribute, class, num))
From (Select attribute, class, count(*) num
      From mine_relation
      Group By attribute, class)
```

In the above example, the inner SQL statement sorts and groups the training set based on attribute values. The *Group By* clause is used instead of a simple *Order By* to count the number of occurrences for each pair attribute/class, since we cannot split cases having the same attribute value. Hence, the splitting criteria is computed faster than before. When the attribute has too many distinct values, a discretization would be advisable.

The final result of this query is the best splitting point of a given attribute. This process is repeated for all other attributes, so that the smallest value is chosen.

### 4.4 Pruning

During the pruning phase, a user-defined function created based on the original tree is used to classify test cases in parallel. This function, which is called from within a SQL statement returns the number of hits and misses during classification.

```
Select class, Classify(...),
From mine_relation_test
```

### 4.5 Rule Extraction

Once the classification model is constructed, a set of rules can be derived reading the tree,

from top to bottom, until reaching each leaf node. The  $n$  rules originally defined (where  $n$  represents the number of leaf nodes) are generalized, and some of them may be simplified or even eliminated. As a final result we will reach an ordered rule set, such as:

```
If a > 1 And b <= 5 And c > 2 Then
  Return 1
Else
  If a <= 1 And d = 'A' Then
    Return 0
  Else ...
  Else Return 1; - default class
End If;
```

As in the pruning case, this rule set is transformed into a PL/SQL function, which can be called from within a SQL statement, enabling the classification of new cases in parallel. This function receives the attributes used during the data mining algorithm and returns the predicted classification.

## 5 Conclusions

The ability to handle large amounts of data has been a major concern in many recent data mining applications. Parallel processing comes to play an important role in this context, once only parallel machines can provide sufficient computational power, memory and disk I/O.

Our work exploits many specific characteristics of DBMS systems, providing a tightly-coupled integration to the database. It also addresses how parallel processing can contribute to solve the mentioned problem.

We have concentrated our work in structuring typical queries in data mining algorithms, in such a manner that they could exploit the highest degree of parallelism offered by the DBMS. A database native language (PL/SQL) was used for implementation, providing high performance and close integration to the database, avoiding unnecessary data transfers from server to client machines.

Furthermore, user-defined functions were specially designed in order to reduce the number of calls to the database, when used within

SQL statements. It is important to mention that the set of stored procedures / functions constructed is easily reusable, since the corresponding code is stored within Oracle data dictionary. They are just ready to use by any other application.

Preliminary results also showed the importance of data fragmentation to achieve good performance in data mining techniques, particularly horizontal partitioning in decision tree processing.

It is possible that changes to current DBMSs and SQL language could enable data mining operations to be performed more efficiently. For example, interquery optimization could enable a single data scan to serve simultaneously multiple queries that need data.

## References

- [1] J. P. Bigus. *Data Mining with Neural Networks*. McGraw-Hill, 1996.
- [2] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [3] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthuramy, editors. *Automating the Analysis and Cataloging of Sky Surveys*. AAAI Press, 1996.
- [4] R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, 1996.
- [5] R. Agrawal, M. Metha, J. Shafer, and R. Srikant. The quest data mining system. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, 1996.
- [6] A. Freitas and S. H. Lavington. *Mining Very Large Databases With Parallel Processing*. Kluwer Academic Publishers, 1998.
- [7] R. Kufirin. Decision trees on parallel processors. In *Proc. of the IJCAI Workshop on Parallel Processing for Artificial Intelligence*, pages 87–95, 1995.
- [8] R. Agrawal, S. Ghosh, T. Imelinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. VLDB Conference*, Vancouver, Canada, 1992.
- [9] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.
- [10] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1993.
- [11] J. Shafer, R. Agrawal, and M. Metha. Sprint: A scalable parallel classifier for data mining. In *Proc. of the 22th Int'l Conference on VLDB*, Mumbai (Bombay), India, 1996.
- [12] M. Metha, 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 (EDBT)*, Avignon, France, 1996.
- [13] L. Brown. *Oracle Database Administration on Unix Systems*. Prentice Hall, 1997.
- [14] Oracle Corporation. *Oracle Parallel Server Concepts & Administration Rel. 7.3*, 1997.