Data Mining: A Tightly-Coupled Implementation on a Parallel Database Server

Mauro Sousa Marta Mattoso Nelson F. F. Ebecken
COPPE - Federal University of Rio de Janeiro
P.O. Box 68511, Rio de Janeiro, RJ, Brazil, 21945-970
mauros, marta@cos.ufrj.br, nelson@ntt.ufrj.br

Abstract

Due to the increasingly difficulty of discovering patterns in real-world databases using only conventional OLAP tools, an automated process such as data mining is currently essential. As data mining over large data sets can take a prohibitive amount of time related to the computational complexity of the algorithms, parallel processing has often been used as a solution. However, when data does not fit in memory, some solutions do not apply and a database system may be required rather than flat files. Most implementations use the database system loosely-coupled with the data mining algorithms. In this work we address the data consuming activities through parallel processing and data fragmentation on the database server, providing a tight integration with data mining techniques. Experimental results showing the potential benefits of this integration were obtained, despite the difficulties to process a complex application.

1. Introduction

Recent years have shown the need of an automated process to discover interesting and hidden patterns in real-world databases, due to the difficulty of analyzing large volumes of data using only OLAP tools. Data mining techniques have increasingly been studied [10], 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. Sampling has been used for a long time, but subtle differences among sets of objects in the database become less evident.

Knowledge discovery through data mining techniques requires data intensive computation. Machine learning algorithms, such as decision trees, have to scan the database intensively, and, depending on the characteristics of the data, distribution of values, etc., these tasks usually require a lot of disk I/O. Parallel processing and database techniques are natural candidates to address this sort of computation, although database engines are not specifically optimized to handle data mining applications.

Many data mining implementations have already used databases. DBMiner [5], for instance, is a system that implements data mining techniques integrated to database technology. Although the idea of using DBMSs is not new, most implementations use databases only to issue queries that are going to be processed by a client machine [6], thus resulting in poor performance.

The Quest data mining system [2, 14, 8] has been addressing the development of fast, scalable algorithms. These algorithms run against data in flat files as well as DB2 family of database products, but databases are accessed in a loosely-coupled mode using dynamic SQL.

Freitas and Lavington [7] have presented a series of primitives for rule induction (RI) algorithms, whose core operation of their candidate rule evaluation procedure is a primitive called Count by Group. This primitive often provides a potential degree of parallelism in parallel DBMSs.

Experimental results showing the potential benefits of a tightly-coupled implementation, parallel processing and data fragmentation were obtained using Oracle Parallel Server configured to run on a multiprocessor IBM RS/6000 SP2 system. The case study application was extracted from a data warehousing environment, which is used to analyze information concerning an insurance company. Despite the difficulties to process this complex application, we have extracted rules and obtained performance improvements for all data intensive activities.

The remainder of this work is organized as follows. Section 2 presents some database issues related to a

tightly-coupled implementation of a data mining algorithm in parallel databases. Section 3 describes the case study used during performance measurements. Section 4 presents some details of implementation and special features of DBMSs exploited and Section 5 describes a subset of our practical experiments. Finally, Section 6 presents our conclusions and observations.

2. Database Issues

There are a variety of data mining algorithms constructed to run in parallel, taking advantage of parallel architectures using specific programming routines. On the other hand, our implementation is based on parallel database systems.

2.1. Advantages and Shortcomings

There are many advantages of using databases when implementing data mining techniques. Implementation becomes simpler, it is possible to work with data sets considerably larger than main memory, and data may be updated or managed as a part of a larger operational process. Furthermore, ad-hoc queries can be used to validate discovered patterns in a painless manner.

On the other hand, as the database system itself is responsible for parallelization, there is less control of how parallelization occurs. Hence, the algorithm may become dependent on some characteristics of the parallel DBMS used.

2.2. Loosely x Tightly-Coupled Integration

Most of current data mining applications that use databases have a loose connection with them. They treat databases simply as containers from which data is extracted directly to the main memory of the computer responsible for running the data mining algorithm. This approach limits the amount of data that can be handled, forcing applications to filter information, and use only a part of it to discover patterns, which is not the best approach to be used.

The idea of executing user-defined computation within databases, thus avoiding unnecessary network traffic (Figure 1), has been a major concern in many DBMSs. One example of this sort of implementation is the stored procedure in commercial databases. For instance, Oracle provides the possibility of implementing procedural code, and storing it within the data dictionary in a pre-compiled form.

Some authors have also developed a methodology for tightly-coupled integration of data mining applications with database systems [3], selectively pushing

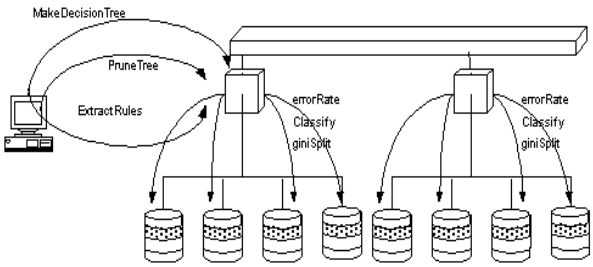

Figure 1. Tightly-coupled integration. Simplified approach showing client calls and PL/SQL functions issued within SQL statements.

parts of the application program into the database system, instead of bringing the records into the application program. 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.

Because of the nature of SQL offered by current database systems, it is not possible to perform only one SQL statement that returns the list of rules discovered analyzing the attributes of the mine relation. The way we have used to implement this kind of feature was coding a series of procedures and functions, which were logically grouped together in an Oracle object named package. Therefore, the user can call a single procedure (e.g. MakeDecisionTree), interactively informing all available parameters. This procedure runs on the server where the data resides, in order to avoid network traffic.

3. A Case Study

We have used a data set based on a life insurance system, which controls information about customers, insurance contracts and components of insurance tariffs. This database is primarily used in a data-warehousing environment so that people in management positions can extract information and take decisions based on it. Our main goal is to classify customers based on all available information.

3.1. Mine Relation

We have constructed a single mine relation with all information to be used during the data mining algorithm. Denormalization was necessary to solve performance problems associated to join operations, which are time consuming tasks, even when performed in par-

allel. The mine relation fits in memory, and uses about 50Mb, 200.000 tuples and 70 attributes.

Information about customers and their dependents includes year of birth, job, sex, marital status, and household income data. Besides, the system holds data about contracts, such as the year when a contract begins and ends, modus of payment, name of the agent responsible for the contract, and type of insurance. Finally, the database keeps information about price of each tariff that is associated to an insurance contract. This data set represents both discrete and continuous attributes with different characteristics.

3.2. Classification Algorithm

Our implementation is based on decision tree algorithms, because they represent well-suited classifiers 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. Furthermore, decision trees can be easily constructed from and transformed into SQL statements [1], 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 [4], ID-3 and its successor C4.5 [13], and SPRINT (SLIQ's parallel implementation) [14, 8].

A decision tree 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 [4, 8]. The recursive partitioning method imposed to the training examples often results in a very complex tree that overfits data [13]. We have also implemented a pruning module, which removes parts of the tree that do not contribute to classification accuracy, producing less complex, and more comprehensible trees.

Even after pruning a tree, it can still represent a complex, incomprehensible structure at first glance. Finally, there is also a rule extraction module, which tends to solve this problem, as soon as irrelevant expressions are eliminated from the originally generated tree. A more detailed description of implemented decision tree algorithm can be found in [15].

4. A Tightly-Coupled Implementation

In this section we present typical problems and solutions associated to machine learning algorithms and parallel database servers.

4.1. Typical Queries in Rule Induction Algorithms

Most of queries issued in rule induction algorithms, particularly decision trees, 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 marital_status, class, count(*)
From mine_relation
Group By marital_status, class
```

Parallel database servers can easily process typical queries, and it is possible to take advantage of the resources of parallel machines.

4.2. User-defined Functions

We have constructed many user-defined functions, which are called within SQL statements, enforcing a tightly-coupled integration with the database and its query language. They decrease the number of calls to the database, avoid network traffic, and make our code considerably simpler.

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

The above query presents an example of a SQL statement used during the growing phase of a decision tree. It is used to calculate the best splitting point of an internal node. The inner SQL statement 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 with the same attribute value. Hence, the splitting criteria is computed faster than before, using only one database call per attribute in each phase.

Besides, during the pruning phase, a user-defined function, which is created based on the original tree, is used to classify test cases in parallel. Function Classify is used to compute the number of hits and misses during classification.

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

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. Rule generalization process issues many SQL statements, in which WHERE conditions represent rules discovered in previous phases. These statements, each one processed in parallel, return the number of false positives and false negatives examples that are going to be used in conjunction with a well-known method called Minimum Description Length (MDL), as it is done in C4.5 [13].

As in the pruning case, the resulting 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.

4.3. Fragmentation Techniques

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. Fragmentation techniques come to play an important role in this context, once they tend to eliminate irrelevant information during query execution.

Several commercial databases provide horizontal fragmentation, most of them in a transparent manner to the application. Some parallel DBMSs, particularly Oracle Server, are able to use these functions (table constraints) to eliminate some partitions when reading data, depending on the filter used in the corresponding SQL statement. Each partition can be processed in parallel, using different execution plans when applicable.

Another relevant problem is fragment allocation. Suppose we define F fragments to be distributed among N nodes. If we simply allocate F / N fragments to each node, there will be a load imbalance, since the decision tree algorithm will work on only a few fragments beyond a certain step, of course if the partitioning functions were appropriately defined.

Declustering (Figure 2) exploits I/O parallelism and minimizes skew, but involves a higher startup and termination costs because a process has to be started and terminated on each of the nodes where relation resides [9].

A typical approach used that allows user queries to deal with smaller relations, causing a smaller number of page accesses, is the vertical fragmentation [11]. In the case of decision trees, it would be necessary to create one relation for each pair attribute/class as it is done with flat files in SPRINT [14]. Remember that decision trees work with lists of attributes, in each phase trying to find the best splitting point among all attributes.

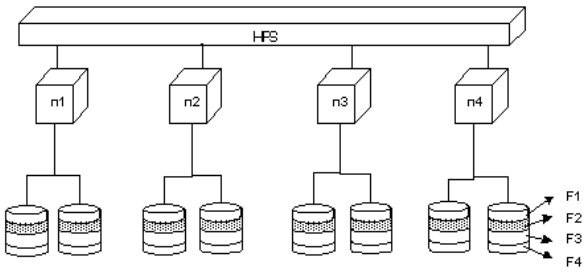

Figure 2. Load balance. Data is spread among all available disks. Grayed disks portions represent fragments being currently read.

5. Practical Experiments

The implementation involves the use of embedded routines written in PL/SQL (Oracle's SQL extension that adds procedural constructs), which are called within SQL statements, enforcing a tightly-coupled integration with the database and its query language. We have used Oracle Parallel Server and Oracle Parallel Query Option, configured to run on a multiprocessor IBM RS/6000 SP2 system. As PL/SQL is not a parallel programming language, all parallelization should be achieved using SQL.

5.1. Oracle Parallel Server and MPP Architecture

Oracle Parallel Server (OPS) in RS/6000 SP2 systems uses function shipping when accessed data is in a remote node. A coordinator process subdivides the work of scanning tables in many logical partitions. The initial partitioning takes into account the physical location of data and tries to minimize disk contention. It is equivalent to a shared-everything phase. In the next step, the coordinator uses function shipping to assign a partition to each slave, representing a shared-nothing step. Nodes that have finished early their processing can "steal" one remote partition through the layer that simulates shared disks (VSD - IBM's Virtual Shared Disk), thus minimizing skew. This last phase is equivalent to a shared-disk approach [12].

5.2. Fragmentation and Allocation Problems

Partition views are objects that Oracle7 uses to implement horizontal fragmentation. Instead of defining

one very large relation, we define several smaller tables, assigning a partitioning function to each one. Afterwards, a view is created using UNION ALL operator.

Ideally, one could imagine that the best partitioning function would separate data into fragments that fit the tree growing phase, so that we can avoid reading more data than necessary. At first hand, we do not have a clear idea of how data is going to be partitioned during the algorithm. A good partitioning function is one that uses attributes from top of the tree, so that the scan of different branches would be directed to different partitions.

After the definition of horizontal fragments, we have to consider their allocation. There are two possibilities that Oracle uses to distribute data among nodes. The first one is creating one tablespace owning many data files, which would be spread throughout nodes, letting the DBMS itself manage striping. It may represent an advantage since disk affinity (equivalent to function shipping) can be used. Another possibility is letting the operating system manage striping, which is a simpler approach, but eliminates the function shipping feature.

Preliminary tests have shown that vertical fragmentation is particularly useful during the first step of the tree-growing phase of decision tree algorithms, when there is no splitting point yet defined; each query is made of a simple group by SQL statement, with no filtering. From the second phase on, joins would be necessary, generating time consuming tasks, even when solved in parallel (Oracle can perform joins in parallel using methods called nested loops and hash joins).

Alternatively, the fragmentation idea can be implemented through concatenated indexes (attribute, class), in order to provide a better response time for the first step of the tree-growing phase. Once the query mentioned only needs these two columns, it can use directly the indexes, without referencing the original table. In Oracle, this is called a fast full scan, which can be run in parallel.

5.3. Performance Results

We have performed several measurements when running the data mining algorithm. Due to lack of space, we present only results related to the tree-growing phase. We have focused on using SQLs as much as possible on the algorithm logic. These SQLs are going to be performed by the parallel database server, and they represent the most time consuming tasks of the algorithm, since they are responsible for handling large volumes of data and for performing some computation, whenever possible.

Response time associated to each SQL varies depending on the number of rows that are going to be processed and on the number of distinct values we are going to fetch. Since a decision tree repeatedly subdivides data, response time decreases as we go down the tree. This observation is true both for sequential and parallel executions.

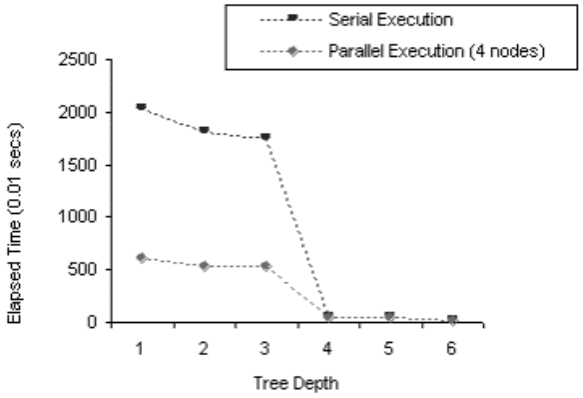

Figure 3. Reponse times for serial and parallel executions for a discrete attribute during the tree growing phase.

Analyzing Figure 3, we observe that, after the fourth level on, Oracle performs serial instead of parallel execution. In the case of Oracle7, it is not possible to perform a parallel index scan, unless all attributes used in a query are present in the index. Since from the fourth level on we analyze less data, a serial index scan performs better than a parallel full table scan.

Queries are also influenced by the sort of attributes we are using in the data mining algorithm. Discrete attributes often result in better response times, because less splitting points are going to be analyzed and less data is going to be fetched by application. Continuous attributes (Figure 4) usually handle many distinct values and they represent a more time consuming task. However, as we go down the tree, continuous attributes are represented by fewer distinct values.

Most of parallelization occurs when handling decision nodes that are on the top of tree, so that more data is going to be analyzed. For these phases, we notice a significant performance speedup when two nodes are used. However, when adding more processing nodes, response time does not decrease in the same ratio. This situation represents the influence of remote tuple access. Since tuples are highly interrelated, the additional nodes also increased communication costs.

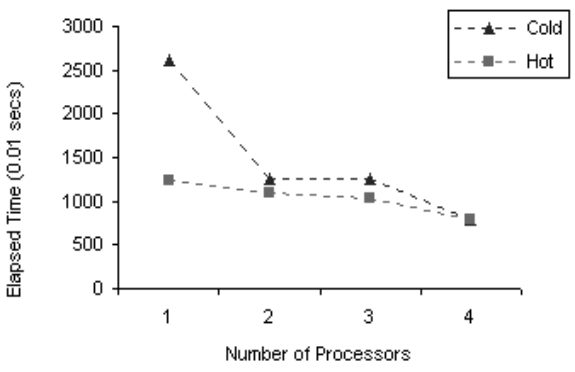

Figure 4. Reponse time associated to continuous attribute handling during the tree-growing phase.

6. Conclusions

This work evaluates the behavior of a tightly-coupled implementation of data mining techniques on a parallel database server, exploiting many specific characteristics of DBMSs to achieve performance. Instead of striving for performance on specific phases, we concentrated on implementing all phases of the data mining process, studying adverse situations such as: large number of attributes, discrete attributes, and continuous attributes with many distinct values. Experimental results showed the potential benefits of this implementation, which generated a decision tree with more than 35.000 nodes.

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, using user-defined functions whenever possible. A database native language was used for implementation, providing high performance and close integration to the database, avoiding unnecessary data transfers from server to client machines. As this implementation is strongly based on SQL, we could easily migrate it to use other relational databases that also have a native programming language.

It is possible that changes to current DBMSs and SQL language would enable data mining operations to be performed more efficiently. Other interfaces, such as those for integrating with indexing and optimization mechanisms , will be available in a near future. Currently, we are in a conversion process of our implementation to Oracle8 (an object-relational database), constructing what Oracle calls a data cartridge - a sort of plug in that can be attached to the database server. We are also motivated by improvements in Oracle8 parallel processing, such as parallel index scans.

References

[1] R. Agrawal, S. Ghosh, T. Imelinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. VLDB Conf.*, Vancouver, 1992.

[2] R. Agrawal, M. Metha, J. Shafer, and R. Srikant. The quest data mining system. In *Proc. of the 2nd Intl Conf. on Knowledge Discovery in Databases and Data Mining*, Portland, 1996.

[3] R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In *Proc. of the 2nd Intl Conf. on Knowledge Discovery in Databases and Data Mining*, Portland, 1996.

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[5] J. H. et al. Dbminer: A system for mining knowledge in large relational databases. In *Proc. Intl Conf. on Data Mining and Knowledge Discovery (KDD'96)*, Portland, 1996.

[6] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurasamy, editors. *Automating the Analysis and Cataloging of Sky Surveys.* AAAI Press, 1996.

[7] A. Freitas and S. H. Lavington. *Mining Very Large Databases With Parallel Processing.* Kluwer Academic Publishers, 1998.

[8] M. Metha, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *Proc. of the Fifth Intl Conf. on Extending Database Technology (EDBT)*, Avignon, 1996.

[9] M. Metha and D. DeWitt. Data placement in shared-nothing parallel database systems. In *VLDB Journal*, Springer-Verlag, 1997.

[10] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[11] S. B. Navathe and M. Ra. Vertical partitioning for database design: A graphical algorithm. In *Proc. SIGMOD Conference*, pages 440–450, 1989.

[12] Oracle Corporation. *Oracle Parallel Server Concepts & Administration Rel. 7.3*, 1997.

[13] J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, 1993.

[14] J. Shafer, R. Agrawal, and M. Metha. Sprint: A scalable parallel classifier for data mining. In *Proc. of the 22th Intl Conf. on VLDB*, Mumbi, 1996.

[15] M. Sousa, M. Mattoso, , and N. F. F. Ebecken. Data mining on parallel database systems. In *Proc. Intl. Conf. on PDPTA: Special Session on Parallel Data Warehousing, CSREA Press*, Las Vegas, 1998.