Efficient Mining for Association Rules with Relational Database Systems

Karthick Rajamani
Dept. of Elec. and Comp. Engg., Rice University
karthick@rice.edu

Alan Cox
Dept. of Computer Science, Rice University
alc@rice.edu

Bala Iyer
IBM Santa Teresa Labs
balaiyer@us.ibm.com

Atul Chadha
IBM Santa Teresa Labs
achadha@us.ibm.com

Abstract

With the tremendous growth of large-scale data repositories, a need for integrating the exploratory techniques of data mining with the capabilities of relational systems to efficiently handle large volumes of data has now risen. In this paper, we look at the performance of the most prevalent association rule mining algorithm - Apriori, with IBM's DB2 Universal Database system. We show that a multi-column (MC) data model is preferable over the commonly used single column (SC) data model for association rule mining. We obtain factors of 4.8 to 6 improvement in performance for the MC data model over commercial implementations for the SC data model. We provide a new relational operator, called **Combinations***, for efficient SQL implementation of Apriori in the database engine - this results in trivial parallelizability, reliability, and portability for the mining application.*

Keywords: *Database mining, association rule mining, operator for association.*

1. Introduction

There is an increasing usage of relational database systems by organizations to store vast amounts of data. The compelling need to extract useful information from this data has led to wide-spread adoption of data mining techniques. In addition to being the storage system for the mining data, relational data base systems provide highly reliable large-volume data handling capability, efficient query indexing and parallelization facilities, and widening portability across a multitude of system architectures. All these have made the integration of data mining tasks with the database system an attractive proposition. The integration of mining applications and database systems, however, requires appropriate data organization, some modifications/enhancements in the database system and either changes in or entirely new mining techniques.

The problem of association rule mining was introduced initially for market basket data analysis [1]. The simple and easily understandable nature of these rules has led to widespread adoption of this technique in data analysis. In this paper, we study the issues involved in integrating association rule mining with the database system.

In our study of association rule mining, we find that the physical data model used for the mined data has a significant impact on performance. The *single-column* (SC) data model is a common physical data model used in market-basket analysis. However, it does not yield sufficient performance for association rule mining. We propose an alternate physical data model for association rule mining, called *multi-column* (MC), that can considerably improve its performance in the database. We use an object-relational extension to the database system, called user-defined function (UDF), to find association rules in MC data. We compare the performances of a commercial implementation of the Apriori association rule mining algorithm for the SC model, with our own UDF prototype of Apriori for the MC data model. Our experimental results indicate that the MC model's performance is up to 6 times better than the SC model's.

When data resides in a RDBMS, using SQL to work with the data increases the reliability and portability of an application. In the case of RDBMS supporting parallelizable queries, the SQL implementation can be easily parallelized when written with due care. Further improvements in performance can also be obtained through improvements in performance of the related querying facilities. We provide an SQL implementation for association rule mining of data in the MC model. We define the semantics for a new relational operator, **Combinations**, which can be effectively used for a very efficient SQL implementation of the Apriori algorithm for association rule mining. We implement a prototype of the operator using new object-relational extensions of the DB2-UDB [7] database system. Our experiments indicate that the performance of this implementation is comparable to that of a more complicated implementation of the Apriori algorithm that was designed for good performance. A more detailed account of our work can be obtained from our technical report [17].

In summary, we make two main contributions to the task of discovering association rules in databases:

i) We identify the physical data model that is most suitable for association rule mining, based on the access pattern of the application.

ii) We provide an efficient SQL implementation for association rule mining using a new relational operator. We provide a prototype using object-relational extensions of DB2 to implement the operator. The SQL implementation enables an easily parallelizable and portable approach to association rule mining.

The rest of the paper is organized as follows. Section 2 presents the Apriori algorithm for association rule mining. Section 3 presents the SC and MC data models for association data. In section 4, we give a brief overview of our method for handling the MC data model, and the results of our experiments with both the data models. In section 5, we provide our SQL implementation for association rule mining with the MC data model. We show how our operator, *Combinations*, can be used along with other common SQL querying facilities to implement the Apriori algorithm. In section 6 we present some of the related work and present our conclusions in section 7.

2. Background on Association Rule mining

This section is based largely on the description of association rule mining given by Agrawal et al. [1, 3]. Let **I** = $\{I_1, I_2, ..., I_n\}$ be the domain of literals called *items*. A record called a *transaction* contains a set of items $I_1, I_2, ..., I_k \subset$ **I**. The input to the association rule mining algorithm is a set of transactions, **T**. We call any set of items $I_1, I_2, ..., I_m \subset$ **I** collectively an *itemset*. An *association rule* is a relation of the form $A \Longrightarrow B$ in **T**, where A, B are itemsets, and $A \cap B = \emptyset$. A is the antecedent of the rule and B is the consequent of the rule.

An itemset has a measure of statistical significance associated with it called *support*. Support for an itemset X in **T** (*support(X)*) is the number of transactions in **T** containing X. For a rule, $A \Longrightarrow B$, the associated support is *support($A \cup B$)*. A *support fraction* is the ratio of the support value to the total number of transactions. The strength of a rule is given by another measure called *confidence*. The confidence of $A \Longrightarrow B$ is the ratio *support($A \cup B$)*/*support(A)*.

The problem of association rule mining is to generate all rules that have support and confidence greater than some user-specified thresholds. Itemsets that have support greater than the user-specified support are called *large* itemsets. For a large itemset S, if $A \subset S$ and *support(S)*/*support(A)* $\geq$ confidence threshold, then $A \Longrightarrow S - A$ is an association rule. The problem of association rule mining is, thus, broken down into two tasks:

(a) The task of determining all large itemsets. This stage is split into two parts - a) *candidate-generation phase* - a set of itemsets called *candidate itemsets* are chosen, this set is chosen such that it contains all potential *large itemsets*, b) *large-itemset generation phase* - the support for the candidates are counted, those with support greater than or equal to the user-specified minimum support qualify to become *large*.

(b) The task of determining the rules with enough confidence, from the large itemsets.

Once the large itemsets are determined, generating the rules is a straightforward task and is not very time consuming. The bulk of the time and memory requirement is for identifying the large itemsets and their support values. The Apriori algorithm proposed by Agrawal and Srikant [3] provides an efficient method for generating association rules. It obtains its efficiency by using potentially smaller candidate sets when counting the support for itemsets to identify large itemsets. It uses the fact that for a given support all the subsets of a large itemset need to be large itemsets themselves. For example, if the set (A,B,C) is a large itemset - i.e it has at least the minimum support, then all of its subsets (A), (B), (C), (A,B), (B,C) and (A,C) also have at least the required minimum support.

Apriori makes multiple passes over the input data to determine all the association rules. Let L_I denote the set of large itemsets of size *I* and let C_I denote the set of candidate itemsets of size *I*. Before making the *I*th pass, Apriori generates C_I using L_{I-1}. Its candidate-generation process ensures that all subsets of size *I-1* of C_I are all members of the set L_{I-1}. In the *I*th pass, it then counts the support for all the itemsets in C_I. At the end of the pass all itemsets in C_I with a support greater than or equal to the minimum support form the large itemset L_I. Figures 1 and 2 provide the pseudocode for the Apriori algorithm. In both the figures, the *subset(s, N)* function gives all the subsets of size N in the set s.

This method of pruning the C_I set using L_{I-1} results in a much more efficient support-counting phase for Apriori when compared to the earlier algorithms. In addition, the usage of a *hash-tree* data structure for storing the candidates provides a very efficient support-counting process (large-itemset generation phase).

As originally proposed, the Apriori algorithm uses its own data structures for measuring the support and does not

```
$L_1$ = {large 1-itemsets};
**for**(k = 2;$L_{k-1} \neq \emptyset$; $k++$)
    $C_k$ = **Generate_candidates**($L_{k-1}$);
    **forall** records t $\in D$
        $C_t = C_k \bigcap$ subset(t, k);
        **forall** candidates $c \in C_t$
            $c.count++$;
    $L_k = \{c \in C_k \mid c.count \geq support\}$
$Answer = \bigcup_k L_k$;
```

Figure 1. Apriori Algorithm

```
**insert into** $C_k$
**select** p.item*1*, p.item*2*, ...p.item*k-1*, q.item*k-1*
**from** $L_{k-1}$ p, $L_{k-1}$ q
**where** p.item*1* = q.item*1*,...,
    p.item*k-2* = q.item*k-2*,p.item*k-1* $<$ q.item*k-1*;

**forall** itemsets $c \in C_k$
    **forall** s $\in$ subset(c, k-1)
        **if** (s $\notin L_{k-1}$) **then**
            **delete** c from $C_k$;
```

Figure 2. Generate_candidates(L_{k-1})

use any SQL queries in its implementation.

3. Physical Data Models for Association

Inspite of significant research efforts on association rule mining techniques, there has been no clear specification for a physical data model for the input data. Consider the example of a retail sales data. Here, the record is a single sale *transaction*, where the association is between the items sold in the transaction. The data is quite often organized in a schema of the form (Transaction_id, Item), hereafter referred to as SC model. This schema would result in one entry for every item sold in the transaction. With the (Transaction_id, Item) schema the Transaction_id value would be repeated for every item bought in that transaction.

The SC data model would be useful for performing conventional relational queries against items bought in transactions. Some of the early work in association rule mining [11] propose the use of such relational queries for discovering association rules, and work with this data model. However, later work [3], have shown significant performance improvement by using Apriori-based algorithms that did not use relational queries in their implementation. But, to the best of our knowledge, no change in the data model was proposed. Even commercial offerings of the these better algorithms have stayed with the SC model.

For the purpose of association rule mining, we propose a data model of the form (Transaction_id, Item1, Item2, ..., ItemC). We will refer to this as the multiple-column (MC) model. In this model, the column Item*i* contains the *i*th item of that particular transaction. The domain of every Item*i* column is the entire set of items. If **C** is the (average) number of non-null items in a row in the MC model of the input data, the SC model requires (C-1) times more rows (albeit shorter ones) to be read in, for every pass over the table. Figure 3 provides an example to illustrate the different representations with SC and MC. In a sense, the MC model provides a horizontal representation of the data while the SC model provides a vertical representation of the data.

SC model

TID	Item
1	Apple
1	Coke
1	Bread
2	Bread
2	Apple
3	Carrot
3	Mango
3	Coconut

MC model

TID	Item1	Item2	Item3
1	Apple	Coke	Bread
2	Bread	Apple	
3	Carrot	Mango	Coconut

Figure 3. Example illustrating the SC and MC data models

3.1. Analysis of Data-Access Costs

We use a simplified table-scan cost model to illustrate the benefit of the MC model. Consider a table with **R** *transactions*, each one involving **C** number of items on the average. For the CPU costs for accessing the input data table, let t_R denote the time associated with a row access - for concurrency control, page-fix etc. Let t_C denote the CPU time required for a single column access. Then the CPU costs associated with both the models are:

$$\mathbf{cpu\_cost}_{MC} = t_R * R + t_C * R * C$$
$$\mathbf{cpu\_cost}_{SC} = t_R * R * C + t_C * R * C$$

The cost difference comes out to :

$$\mathbf{cpu\_cost}_{SC} - \mathbf{cpu\_cost}_{MC} = R * t_R * (C-1)$$

Thus we see that the cpu-cost difference increases as the number of rows in the table increases and also as the average number of items in a transaction grows.

The I/O cost is proportional to the volume of data scanned for each model. Let S_h be the size of the header associated with each row in the database table, S_i be the number of bytes per item and S_t denote the size of the transaction id. In the case of the MC model, since the items are already grouped by transaction id into each row, the transaction ids need not be scanned. However, for the SC model since the grouping by transaction id has to be done, their values also need to be scanned in. The volume of data scanned for both the models are:

$$\mathbf{vol}_{MC} = R * (S_h + C * S_i)$$
$$\mathbf{vol}_{SC} = R * C * (S_h + S_i + S_t)$$

The volume difference comes out to :

$$\mathbf{vol}_{SC} - \mathbf{vol}_{MC} = R * (S_h * (C - 1) + S_t * C)$$

Here, we see that the I/O-cost difference also increases with increase in the number of rows and also with an increase in the average number of items in a transaction.

In the above models, we ignored the cost for handling the null entries in the MC model. If we consider t_n as the cpu cost for processing a null item, and N as the total number of columns in the MC model, we get the condition $t_R/t_n > N/(C-1)$ for SC to have a higher cpu cost. Similarly, if we consider S_n to be the size of the null indicator, for the I/O cost of SC to be greater than the I/O cost of MC we get the condition $(S_h * (C-1) + S_t * C) > S_n * N$. For reasonable values of N (relative to C) these conditions would hold for most databases. An additional cost for the SC model arises because of the grouping on transaction id that is required to obtain all the items sold in a transaction together. This is eliminated in the MC model because of the implicit grouping provided by having all such items in a single row.

3.2. Other Aspects

Regarding additional storage for storing the null entries in the MC data model, most commercial database systems provide facilities for efficient compression of data. For short records (lot of nulls) a good compression mechanism would easily reduce the extra storage for the nulls (all identical fields - so good compressibility) to just a marginal increase. Iyer and Wilhite [14] provide a comprehensive analysis of various data compression options and discuss them in the context of IBM's DB2 database management system. In addition to minimizing storage costs, compression could even speed up the mining process by trading off higher I/O latencies with lower compression/uncompression overheads for very large datasets.

A key advantage of the SC model is the extensibility of the domain of items after the initial table has been created. This is retained in our MC model, as no column for the items has any specific association with a particular item. Column Item*i* is just a place-holder for the *i*th item in a particular transaction (in the SC model the Item column is the place-holder for any of the items in a transaction). Thus, any addition to the domain of items, does not need to alter the schema for the MC table in any way. The limit of the number of columns in the table places a corresponding limit on the maximum number of items that can be part of one record. However, as can be seen in figure 4 only an extremely low number of transactions are truncated because of this. These should probably have been treated as outliers anyway.

The MC model is limited in the nature of relational queries over the items. Data analysis where neither the grouping of items of same transaction nor the efficiency of obtaining all the data in the table are relevant could still require the SC model. However, when the primary application has a grouping requirement like that of association rule mining (or needs to do complete scans of the input table) the potential performance benefit justifies the usage of the MC model. The performance benefits are even greater when repeated runs of such an application are required for data analysis, such as is often the case with association rule mining. It is also our experience that many real-life datasets are found in the MC data model. But they get converted to the SC model, at significant conversion costs for storage and computation, because the underlying assumption for the implementation of the mining algorithm is that of the SC data model. By pointing out that better performance can be obtained with the MC data model we would hopefully discourage such practice.

4. Implementation and Results for MC Data Model

In our study, we compare the performance of two implementations for association rule mining using the Apriori algorithm. We use the highly optimized implementation of Apriori for the SC model from IBM's Intelligent Miner (IM) Data Mining Suite [13]. For the MC model, we use one of DB2's object-relational extensions [5], called the user-defined function (UDF). A UDF provides an application the means to perform computation on retrieved records, inside the database system. Similar facilities are available on other commercial database systems like Informix's Illustra or Oracle v8. In our implementation, we pass every row of data from the MC table to the UDF function. All the computations and generation of rules is done by the code inside the UDF.

UDFs in DB2, can be run in two modes, either *fenced* or *unfenced*. In the unfenced mode, they share the database's address space, avoiding the overhead of switching from the database's address space to the application's address space, when data is passed from the database to the application. In the fenced mode, UDFs run in the application's address space which is distinct from the database's. Agrawal and Shim [2] have shown the benefit of executing UDFs in the unfenced mode. We chose to conduct our experiments with UDFs running in the fenced mode, to provide the same environment for our implementation as for the Intelligent Miner (IM) implementation of Apriori where all computations are performed in the application's address space.

We also performed tests with a UDF implementation for the SC data model. But its performance was upto a factor of 2-3 times slower than the highly optimized IM implementation. Hence, we decided to use IM's implementation as representative of the SC model. The data structures and the computation modules are quite similar for both the IM and UDF implementations. The only difference lies in the extraction of data from the database. IM obtains the data from a table with data in the SC model, and the UDF from a table in the MC model.

Our experiments were conducted on an IBM PowerStation 590, that has a 66Mhz Power2 processor with 512MB memory, with a 1.07GB Serial-link disk drive. We used DB2 v2.1 as our RDBMS. Our mining data was drawn from the daily sales data of a large chain of grocery stores in Canada, the data collected over a seven month period. The original data was stored in a format similar to that of the MC data model, but had other attributes per item in each transaction. We use just the items sold in each transaction as our input. The data has an average of 12 items per sale. So the SC model has about 12 times the number of rows in the MC model. Our UDF implementation for the MC model supported a maximum of 60 items per transaction (the input data under both SC and MC models were identical). Figure 4 gives the distribution for the number of items per transaction. From it we see that there are only a negligible number of transactions that have more than 60 items per transaction. We ran our experiments for a minimum support of 0.5% varying the total number of transactions from 100K-400K.

Figure 5 shows the time curves for IM and the UDF implementations. The ratio of the association time for IM to the UDF association time varies between 4.8 to 6. This factor, significant even for single runs, becomes very important when repeated association runs need to be performed over the input data, with varying support and confidence. Data mining techniques in general, and association rule mining in particular, are typically applied to the data sets repeatedly with varying parameters. Thus any improvement in the execution time of a single run of the application has a much bigger effect in the overall performance gains for that application. The results, clearly indicate the superiority of the MC model over the SC model for database tables used as input for association rule mining.

5. SQL implementation for Association

For implementing Apriori in SQL over data in the MC data model, we propose a new relational operator, **Combinations**. *Combinations* is a specific form of the generic *Powerset* operator - it produces all subsets of the powerset of a specified size. Hence, it is a natural addition for any system supporting set operations, such as an RDBMS. *Combinations* takes a set of items *S* and a number *I* as input, and returns the different combinations of size *I* from *S* as rows in a new relation. The items in a row (i.e the items in each combination) are in lexicographic order. The resulting table has *I* columns and ${}^{|S|}C_I$ rows. Formally,

$$Combinations(S, I) = \bigcup_{all} X_I,$$
$$\text{where } X_I = \{i_1, i_2, .., i_I\},$$
$$i_j \in S,\ (1 \leq j \leq I) \text{ and } i_1 < i_2 ... < i_I.$$

For example, for an input set $S = \{$A,B,C,D$\}$ and $I = 2$, the output of operator *Combinations* would be $\{\{$A,B$\}$, $\{$A,C$\}$, $\{$A,D$\}$, $\{$B,C$\}$, $\{$B,D$\}$, $\{$C,D$\}\}$. For use with the MC model, the *Combinations* operator takes as its input set, the elements in the row of a table, unlike traditional SQL operators that take a column of a table as their input set. The operator performs the required *subset* function that is used in both the large-itemset and candidate-itemset generation phases of the Apriori algorithm (figures 1 and 2). For an input table Data with N columns, the SQL statements using *Combinations* for generating large itemsets of size *I* and candidate itemsets of size *I* are given in figures 6 and 7.

Every row in L_I contains a large itemset of size *I*. The first *I* columns contain the items of the itemset and the last column contains the support value. Every row in C_I contains one candidate itemset of size *I*, with each column containing one item. Once all the large itemsets are generated, the rules can be generated from them quite easily. We can again easily use the *Combinations* operator to generate the rules using SQL queries [17].

We implement the *Combinations* operator using UDB's object-relational extension, Table-Function [7, 18]. Like a UDF the table-function is a user-defined method that can be used in SQL queries. But, the table-function returns a table to the query invoking it and is used in the FROM clause of an SQL query. This extension provides a highly flexible table construction utility to be used along with SQL.

Performance

In the *large-itemset generation phase*, using *Combinations* on all the items in a transaction would generate too many unwanted combinations (those not found in the candidate itemsets). To reduce such combinations we add a new clause to the operator - *in <Filter>*. The operator is now modified to emit combinations of only those items in the input set which are also present in set *<Filter>*. C_I is used as the *<Filter>* set for generating L_I. Incorporating this clause into our table-function was done easily by passing the small hash table containing the items in C_I as a binary-large-object (BLOB) argument to the table function.

There is yet another optimization that can improve the performance of the SQL approach using *Combinations*. We found that during the large itemset generation phase the output of the table function for *Combinations* is created

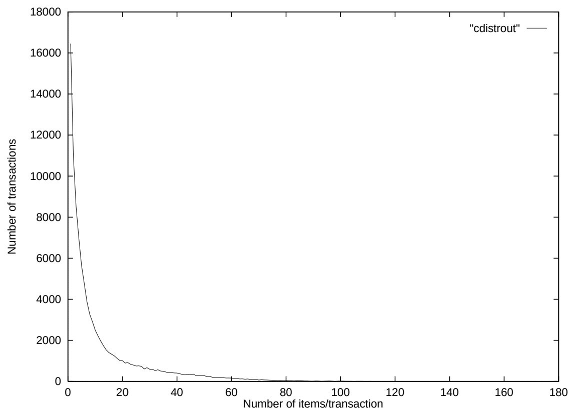

Figure 4. Distribution for Number of Items/Transaction

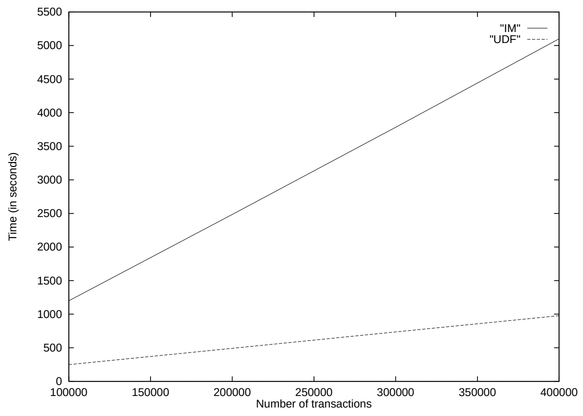

Figure 5. Execution time for IM and UDF implementations

```
insert into $L_I$
select C.item1, ..., C.itemI, count(*)
from Data d, $C_I$ C join
        Combinations(d.item1, ...,d.itemN, I) as g
        on C.item1 = g.item1 and ... and C.itemI = g.itemI
group by (C.item1, C.item2 ..., C.itemI)
having count(*) $\geq$ support
```

Figure 6. SQL for generating large itemsets

```
insert into $C_I$
select l1.item1, l1.item2, ...,l1.itemI-1, l2.itemI-1
from $L_{I-1}$ l1, $L_{I-1}$ l2
where l1.item1 = l2.item1 and
            :
            l1.itemI-2 = l2.itemI-2 and
            l1.itemI-1 $<$ l2.itemI-1 and
            (select count(*)
            from $L_{I-1}$ join
            Combinations(l1.item1,....,
                  l1.itemI-1, l2.itemI-1, I-1)
            ) = I
```

Figure 7. SQL for generating candidate itemsets

as a temporary table in the database, and then joined with the candidates table. If this temporary table is not generated, but the counts for the candidate sets directly incremented by using **hash-grouping** with the entries in the candidates table as the keys, performance would be significantly enhanced. However, this involves modifications to the RDBMS optimizer for queries involving table functions. Since this was beyond our control in the existing set up, this optimization was not performed.

We ran our experiments for the SQL implementation with DB2-UDB beta version (that provides the Table Function extension) installed on an SP2 node with 256 MB memory. Our execution times for the SQL implementation were between 2 to 6 times that of the UDF implementation of the Apriori algorithm. This performance is comparable to the best commercial SC implementation for the algorithm that we could find (the association rule mining tool in Intelligent Miner).

When the memory requirements for the UDF approach are met by the system, it provides the best uniprocessor performance. This is because of the usage of the complex hash-tree structure that provides for efficient scanning of the tuples for candidate itemsets in the UDF approach. In the SQL implementation simple database operations - *join* and *group by* are used for the same purpose (these use data structures not as efficient as the hash-tree for this task). However, by relying on the database system for providing the bulk of the storage requirements (memory/disk) and database operations for its performance, the SQL approach is much more scalable and portable. In addition to it, the queries can be trivially parallelized in an SMP, MPP or even a cluster of SMPs environment.

6. Related Work

Many researchers have proposed new operators to support data mining. DMQL [9] proposes operators for mining characteristic rules, classification rules, association rules etc. Meo et al.[16] had proposed a SQL-like operator for generalized association rule mining. The M-SQL [12] language has a special operator *Mine* to generate and query propositional rules. Tsur et al. [21] generalize the idea of Apriori-pruning to a concept called *query flocks* and show how it can be used in general-purpose mining systems and in conventional query optimizers. These proposals focus on the general querying semantics and interface for mining, as opposed to our focus on providing a specific efficient SQL-based implementation of mining for database systems.

Lakshmanan et al. [15] and Gyssens et al. [8] have proposed extensions to SQL that work on non-column data in a relational table. However, their extensions differ from ours as they are proposed for inter-operability in multi-database systems and not for providing the flexibility and functionality required by data mining applications.

Agrawal and Shim [2] show the benefit of using UDFs for the development of applications tightly coupled with the database engine.

Houtsma and Swami [11] proposed *SETM*, an SQL based algorithm for association rule mining. Their algorithm uses simple database operations - sorting and merge-scan joins. However, their joins are more expensive as they are against the input data table and they do not have an efficient candidate set pruning such as Apriori.

More recently, Sarawagi et al. [19] compare many alternatives for implementing association rule mining with relational database systems. They also examine a *Horizontal* data model which is comparable to our MC data model. However, they work under the assumption that the original data can be found only in the SC data model, and so, incorporate an additional factor for conversion in their cost analysis. Sarawagi et al. and Zaki et al. [22] consider *Vertical or tid-list* transformations to the data to achieve good performance. In this transformation, they build, for each item, a list of the ids of the transactions (tid-list) in which the item occurs. The support count for each itemset is then obtained by merging the tid-lists of the items in that itemset. They find this approach to have very good performance when the number of candidate itemsets is not large, but expensive in terms of storage requirements for doing the transformation.

In the context of providing efficient I/O access for decision-tree and Bayes classifiers, the Monet main-memory database system [10, 4] uses the Decomposed Storage Model [6] to vertically fragment all relations into Binary Association Tables (BAT). This significantly reduces the I/O costs associated with the attribute-oriented accesses required for obtaining attribute-class correlation counts at various stages of the classification algorithms. The SPRINT [20] classifier adopts a similar vertical fragmentation by creating an *attribute-list* for each attribute containing just the attribute values, class labels, and the record ids from the original multi-attribute table. This again provides significant I/O reduction in the execution of the decision-tree based algorithm used in SPRINT.

7. Conclusions

We have shown that the physical layout of data plays an important part in the performance of the association rule mining problem. Our results show that the MC representation is the better physical data model compared to the SC model. We have also provided the *Combinations* operator that is necessary for performing a SQL-query-based implementation of Apriori over the MC data. Our experiments with a prototype implementation of this operator using the UDB's table functions show good uniprocessor performance. We also indicate how we can obtain still better performance for the SQL implementation through query optimization. The SQL implementation provides a convenient method for using the optimization and parallelization capabilities in the database system for the benefit of the application. It also provides a scalable implementation (not memory bound) for the algorithm. Our work indicates that mining using the database engine is a feasible proposition. However, it requires careful attention to the data model and access pattern, and suitable extensions to the relational database system to provide the new functionality (object-relational extensions such as UDFs and Table Functions) required by data mining applications.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.

[2] R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, August 1996.

[3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Databases*, September 1994.

[4] P. Boncz, W. Quak, and M. L. Kersten. Monet and its Geographical Extensions: a Novel Approach to High-Performance GIS Processing. In *Proceedings of the 5th International Conference on Extending Database Technology*, March 1996.

[5] D. Chamberlin. *Using the New DB2 - IBM's Object-Relational Database System*. Morgan Kaufmann, 1996.

[6] G. Copeland and S. Khoshafian. A Decomposition Storage Model. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, May 1985.

[7] DB2 Universal Database (UDB) . http//www.software.ibm.com/data/db2/. Web Document.

[8] M. Gyssens, L. Lakshmanan, and I. Subramanian. Tables as a Paradigm for Querying and Restructuring. In *Proceedings of the 1996 ACM Symposium on Principles of Database Systems*, June 1996.

[9] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. A Data Mining Query Language for Relational Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, May 1996.

[10] M. Holsheimer and M. L. Kersten. Architectural Support for Data Mining. In *International Workshop on Knowledge Discovery in Databases, Seattle*, 1994.

[11] M. Houtsma and A. Swami. Set-oriented Mining of Association Rules. Technical Report RJ 9567, IBM Almaden Research Center, October 1993.

[12] T. Imielinski, A. Virmani, and A. Abdulghani. Application Programming Interface and Query Language for Database Mining. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, August 1996.

[13] International Business Machines. *IBM Intelligent Miner User's guide*, Version 1 Release 1, SH12-6213-00 edition, July 1996.

[14] B. R. Iyer and D. Wilhite. Data Compression Support in Databases. In *Proceedings of the 20th International Conference on Very Large Databases*, September 1994.

[15] L. Lakshmanan, F. Sadri, and I. Subramanian. SchemaSQL - A Language for Interoperability in Relational Multi-database Systems. In *Proceedings of the 22nd International Conference on Very Large Databases*, 1996.

[16] R. Meo, G. Psaila, and S. Ceri. A New SQL-like Operator for Mining Association Rules. In *Proceedings of the 22nd International Conference on Very Large Databases*, 1996.

[17] K. Rajamani, B. Iyer, and A. Chadha. Using DB2's Object-Relational Extensions for Mining Association Rules. Technical Report TR 03,690, Santa Teresa Laboratory, IBM Corporation, September 1997.

[18] B. Reinwald and H. Pirahesh. SQL Open Heterogeneous Data Access. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, June 1998.

[19] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, June 1998.

[20] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proceedings of the 22nd International Conference on Very Large Databases*, September 1996.

[21] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: a Generalization of Association-rule Mining. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, June 1998.

[22] M. Zaki, S. Parthasarathy, W.Li, and M. Ogihara. New Algorithms for Fast Discovery of Association Rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, August 1997.