International Journal of Research

**International Journal of Research (IJR)**

**e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015**

Available at http://internationaljournalofresearch.org

# Fast data mining using association rules in distributed databases

**Sk. Reshma [1]& N.Venkateswara rao [2]**

[1]PG Scholar, Dept of CSE, Rao & Naidu Engineering College, Ongole, Prakasam Dist,Andhra Pradesh

[2]Associate Professor, Dept of CSE, Rao & Naidu Engineering College, Ongole, Prakasam Dist,Andhra Pradesh

**Abstract:**

*Data mining can extract important knowledge from large data collections – but sometimes these collections are split among various parties. Privacy concerns may prevent the parties from directly sharing the data, and some types of information about the data. This paper addresses secure mining of association rules over horizontally partitioned data. The methods incorporate cryptographic techniques to minimize the information shared, while adding little overhead to the mining task.*

*The proposed is simple, yet powerful, methods to generate SQL code to return aggregated columns in a horizontal tabular layout, returning a set of numbers instead of one number per row. This new class of functions is called horizontal aggregations. Horizontal aggregations build data sets with a horizontal de normalized layout (e.g. point-dimension, observation-variable, instance-feature), which is the standard layout required by most data mining algorithms.*
*The proposed method used three categories to evaluate horizontal aggregations: CASE: Exploiting the programming CASE construct; SPJ: Based on standard relational algebra operators (SPJ queries); PIVOT: Using the PIVOT operator, which is offered by some DBMSs. Experiments with large tables compare the proposed query evaluation methods. A CASE method has similar speed to the PIVOT operator and it is much faster than the SPJ method. In general, the CASE and PIVOT methods exhibit linear scalability, whereas the SPJ method does not.*

## I INTRODUCTION

Data mining methodology has emerged as a means of identifying patterns and trends from large quantities of data. Data mining go hand in hand: most tools operate by gathering all data into a central site, then running an algorithm against that data.. This paper addresses the problem of computing association rules within such a scenario. We assume homogeneous databases: All sites have the same schema, but each site has information on different entities. The goal is to produce association rules that hold globally, while limiting the information shared about each site. Computing association rules without disclosing individual transactions is straight forward. In a relational database, especially with normalized tables, a significant effort is required to prepare a summary data set that can be used as input for a datam mining or statistical algorithm. Most algorithms require as input a data set with a horizontal layout, with several Records and one variable or dimension per column. That is the case with models like clustering, classification, regression and PCA; consult. Each research discipline uses different terminology to describe the data set. In data mining the common terms are point-dimension. Statistics literature generally uses observation-variable. Machine learning research uses instance-feature. This paper introduces a new class of aggregate functions that can be used to build data sets in a horizontal layout (de normalized with aggregations), automating SQL query writing and extending SQL capabilities. We show evaluating horizontal aggregations is a challenging and interesting problem and we introduced alternative methods and optimizations for their efficient evaluation.

## II. LITERATURE SURVEY

We study here the problem of secure mining of association rules in horizontally partitioned databases. In that setting, there are several sites (or players) that hold homogeneous databases, i.e., databases that share the same schema but hold information on different entities. The goal is to find all association rules with given minimal support and confidence levels that hold in the unified database, while minimizing the information disclosed about the private databases held by those players. That goal defines a problem of secure multiparty computation. In such problems, there are M players that hold private inputs, x1, . . . , xM, and they wish to securely compute y = f(x1, . . . , xM) for some public function f. If here existed a trusted third party, the players could surrender to him their inputs and he would perform the function evaluation and send to them the resulting output. In the absence of such a trusted third party, it is needed to devise a protocol that the players can run on their own in order to arrive at the required output y. Such a protocol is considered perfectly secure if no player can learn from his view of the protocol more than what he would have learnt in the idealized setting where the computation is carried out by a trusted third party. Yao was the first to propose a generic solution for this problem in the case of two players. Other generic solutions, for the multi-party case, were later proposed in [2,4,10].2 T. Tassa In our problem, the inputs are the partial databases, and the required out-put is the list of association rules with given support and confidence. As the above mentioned generic solutions rely upon a description of the function f as a Boolean circuit, they can be applied only to small inputs andfunctions which are realizable by simple circuits. In more complex settings, such as ours, other methods are required for carrying out this computation. In such cases, some relaxations of the notion of perfect security might be inevitable when looking for practical protocols, provided that the excess information is deemed benign (see examples of such protocols in e.g. [12,20,23]). Kantarcioglu and Clifton studied that problem in [12] and devised a protocol for its solution. The main part of the protocol is a sub-protocol for the secure computation of the union of private subsets that are held by the different players. (Those subsets include candidate itemsets, as we explain below.) That is the most costly part of the protocol and its implementation relies upon cryptographic primitives such as commutative encryption, oblivious transfer, and hash functions. This is also the only part in the protocol in which the players may extract from their view of the protocol information on other databases, beyond what is implied by the final output and their own input. While such leakage of information renders the protocol not perfectly secure, the perimeter of the excess information is explicitly bounded in and it is argued that such information leakage is innocuous, whence acceptable from practical point of view. Herein we propose an alternative protocol for the secure computation of the union of private subsets. The proposed protocol improves upon that in terms of simplicity and efficiency as well as privacy. In particular, our protocol does not depend on commutative encryption and oblivious transfer (what simplifies it significantly and contributes towards reduced communication and computational costs). The protocol that we propose here computes a parameterized family of functions, which we call threshold functions, in which the two extreme cases correspond to the problems of computing the union and intersection of private subsets. Those are in fact general-purpose protocols that can be used in other contexts as well. Another problem of secure multi-party computation that we solve here as part of our discussion is the problem of determining whether an element held by one player is included in a subset held by another. Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy n comp**any strength. Once these things are** satisfied, ten next steps is to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external suppor. This support can be obtained from senior programmers, from book or from websites. Before building

## III.OBJECTIVES AND MOTIVATIONS

Objectives generally, data mining (sometimes called data or knowledge discovery database (KDD) is the process of analyzing data from different perspectives and summarizing it into useful information. Information that can be used to increase revenue, cuts costs, or both data mining software is one of a number of analytical tools for analyzing data. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among different fields in large relational databases. Building a suitable data set for data mining purposes is a time- consuming task. This task generally requires writing long SQL statements or customizing SQL Code if it is automatically generated by some tool. There are two main ingredients in such SQL code: joins and aggregations; we focus on the second one. The most widely-known aggregation is the sum of a column over groups of rows. Some other aggregations return the average, maximum, minimum or row count over groups of rows.
There exist many aggregations functions and operators in SQL. Unfortunately, all these aggregations have limitations to build data sets for data mining purposes. The main reason is that, in general, data sets that are stored in a relational database (or a data warehouse) come from On-Line Transaction Processing (OLTP) systems where database schemas are highly normalized. But data mining, statistical or machine learning algorithms generally require aggregated data in summarized form. Based on current available functions and clauses in SQL, a significant effort is required to compute aggregations when they are desired in a cross tabular (Horizontal) form, suitable to be used by a data mining algorithm. Such effort is due to the amount and complexity of SQL code that needs to be written, optimized and tested. There are further practical reasons to return aggregation results in a horizontal (cross-tabular) layout. Standard aggregations are hard to interpret when there are many result rows, especially when grouping attributes have high cardinalities.

To perform analysis of exported tables into spreadsheets it may be more convenient to have aggregations on the same group in one row (e.g. to produce graphs or to compare data sets with repetitive information). OLAP tools generate SQL code to transpose results (sometimes called PIVOT). Transposition can be more efficient if there are mechanisms combining aggregation and transposition together. With such limitations in mind, we propose a new class of aggregate functions that aggregate numeric expressions and transpose
results to produce a data set with a horizontal /.mnvlayout. Functions belonging to this class are called horizontal aggregations. Horizontal aggregations represent an extended form of traditional SQL aggregations, which return a set of values in a horizontal layout (somewhat similar to a multidimensional vector), instead of a single value per row. This article explains how to evaluate and optimize horizontal aggregations generating standard SQL code

### A. Data Mining Techniques

The most commonly used techniques in data mining are:
1. **Clustering**: Data items are grouped according to logical relationships or consumer preferences. For example, data can be mined to identify market segments or consumer affinities.

**2. Associations Rule**: Data can be mined to identify associations. The beer-diaper example is an example of associative mining.
**3. Sequential patterns**: Data is mined to anticipate behavior patterns and trends. For example, an outdoor equipment retailer could predict the likelihood of a backpack being purchased based on a consumer's purchase of sleeping bags and hiking shoes.
**4. Artificial neural networks**: Non-linear predictive models that learn through training and resemble biological neural networks in structure.
**5. Genetic algorithms**: Optimization techniques that use process such as genetic combination, mutation, and natural selection in a design based on the concepts of natural evolution.

**6. Decision trees**: Tree-shaped structures that represent sets of decisions. These decisions generate rules for the classification of a dataset. Specific decision tree methods include Classification and Regression Trees (CART) and Chi Square Automatic Interaction Detection (CHAID) CART and CHAID are decision tree techniques used for classification of a dataset. They provide a set of rules that you can apply to a new (unclassified) dataset to predict which records will have a given outcome.

**7. Nearest neighbor method**: A technique that classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset (where k 1) sometimes called the k-nearest neighbor technique.

**8. Rule induction**: The extraction of useful if-then rules from data based on statistical significance.

**9. Data visualization**: The visual interpretation of complex relationships in multidimensional data. Graphics tools are used to illustrate data relationships.

## VI. IMPLEMENTATION:

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective The implementation stage involves careful planning, investigation of the existing system and it’s constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

The CwFT algorithm is a workflow scheduling algorithm extended from the HEFT algorithm for distributed en-vironments with multiple heterogeneous processing nodes. Instead of optimizing only the workflow

### A. COST WITH FINISH TIME-BASED ALGO-RITHM:

makespan as usual, CwFT algorithm also considers reducing the monetary cost that CCs need to pay in a computing framework with the combination between numerous Cloud node and a local system. Similar to HEFFT, the CwFT algorithm is comprised of two phases: Task Prioritizing to mark the priority level for all tasks and Node Selection to select tasks in a descending or-der by the priority level and then schedule each select-ed task on an appropriate processing node to optimize the value of the utility function.

### B. OBJECTIVES:

Objectives Generally, data mining (sometimes called data or knowledge discovery database (KDD) is the process of analyzing data from different perspectives and summarizing it into useful information. Informa-tion that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analyzing data. It allows users to an-alyze data from many different dimensions or angles, categorize it, and summarize the relationships identi-fied. Technically, data mining is the process of finding correlations or patterns among different fields in large relational databases. Building a suitable data set for data mining purposes is a time- consuming task. This task generally requires writing long SQL statements or customizing SQL Code if it is automatically generated by some tool. There are two main ingredients in such SQL code: joins and aggregations; we focus on the sec-ond one. The most widely-known aggregation is the sum of a column over groups of rows. Some other ag-gregations return the average, maximum, minimum or row count over groups of rows. There exist many ag-gregations functions and operators in SQL.

Unfortunately, all these aggregations have limitations to build data sets for data mining purposes. The main reason is that, in general, data sets that are stored in a relational database (or a data warehouse) come from On-Line Transaction Processing (OLTP) systems where database schemas are highly normalized. But data min-ing, statistical or machine learning algorithms generally require aggregated data in summarized form. Based on current available functions and clauses in SQL, a signifi-cant effort is required to compute aggregations when they are desired in a cross tabular (Horizontal) form, suitable to be used by a data mining algorithm. Such ef-fort is due to the amount and complexity of SQL code that needs to be written, optimized and tested.

There are further practical reasons to return aggregation results in a horizontal (cross-tabular) layout. Standard aggregations are hard to interpret when there are many result rows, especially when grouping attributes have high cardinalities. To perform analysis of exported tables into spreadsheets it may be more convenient to have aggregations on the same group in one row (e.g. to produce graphs or to compare data sets with repetitive information). OLAP tools generate SQL code to transpose results (sometimes called PIVOT). Transposition can be more efficient if there are mechanisms combining aggregation and transposition together. With such limitations in mind, we propose a new class of aggregate functions that aggregate numeric expressions and transpose results to produce a data set with a horizontal layout. Functions belonging to this class are called horizontal aggregations. Horizontal aggre-gations represent an extended form of traditional SQL aggregations, which return a set of values in a horizontal layout (somewhat similar to a multidimensional vector), instead of a single value per row. This article explains how to evaluate and optimize horizontal aggregations generating standard SQL code

## C. HORIZONTAL AGGREGATION:

Introduce a new class of aggregations that have similar behavior to SQL standard aggregations, but which produce tables with a horizontal layout. In contrast, we call standard SQL aggregations vertical aggregations since they produce tables with a vertical layout. Horizontal aggregations just require a small syntax extension to aggregate functions called in a SELECT statement. Alternatively, horizontal aggregations can be used to generate SQL code from a data mining tool to build data sets for data mining analysis. We start by explaining how to automatically generate SQL code

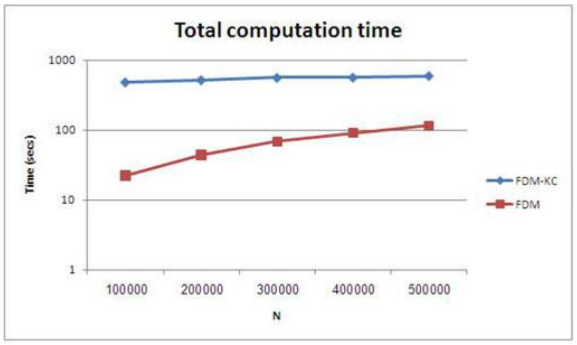
2.**Proposed Syntax in Extended SQL** : We now turn our attention to a small syntax extension to the SELECT statement, which allows understanding our proposal in an intuitive manner. We must point out the proposed extension represents non -standard SQL because the columns in the output table are not known when the query is parsed.

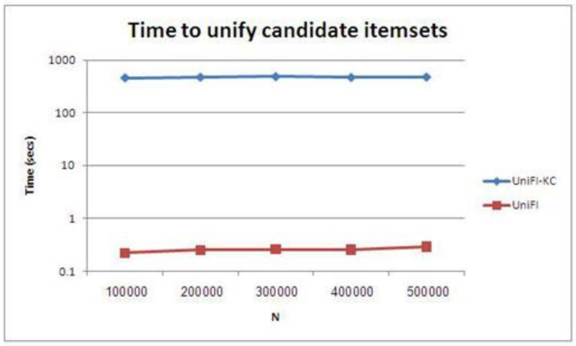
**3.SQL Code Generation**: Query Evaluation Methods We proposes three methods to evaluate horizontal aggregations. The first method relies only on relational operations. That is, only doing select, project, join and aggregation queries; we call it the SPJ method. The second form relies on

the SQL "case" constructs; we call it the CASE method.
Each table has an index on its primary key for efficient join processing.. The third method uses the built in PIVOT operator, which transforms rows to columns (e.g. transposing). An overview of the main steps to be explained below (for a sum () ) aggregation.

## V. Experimental results

Fig.1 shows the values of the three measures that were listed in SectionIV-C as a function of *N*. In all of those experiments, the value of *M* and *s* remained unchanged *M* = 10 and *s* = 0.1. Fig.2 shows the values of the three measures as a function of *M*; here, *N* = 500, 000 and *s* = 0.1. Fig.3 shows the values of the three measures as a function of *s*; here, *N* = 500, 000 and *M* = 10. From the first set of experiments, we can see that *N* has little effect on the runtime of the unification protocols, UNIFI-KC and UNIFI, nor on the bit communication cost. However, since the time to identify the globally *s*-frequent item sets does grow linearly with *N*, and that Procedure is carried out in the same manner in FDM-KC and FDM, the advantage of Protocol FDM over FDM-KC in terms of runtime decreases with *N*. While for *N* = 100, 000, Protocol FDM is 22 times faster than Protocol FDM-KC, for *N* = 500, 000 it is five times faster. (The total computation times for larger values of *N* retain the same pattern that emerges from Fig.1; for example, with *N* = 106 the total computation times for FDM-KC and FDM were 744.1 and 238.5 seconds, respectively, which gives an improvement factor of 3.1.). The second set of experiments shows how the computation and communication costs increase with *M*. In particular, the improvement factor in the bit communication cost, as offered by Protocol UNIFI with respect to Protocol UNIFI-KC, is in accord with our analysis. Finally, the third set of experiments shows that higher support thresholds entail smaller computation and communication costs since the number of frequent item sets decreases.





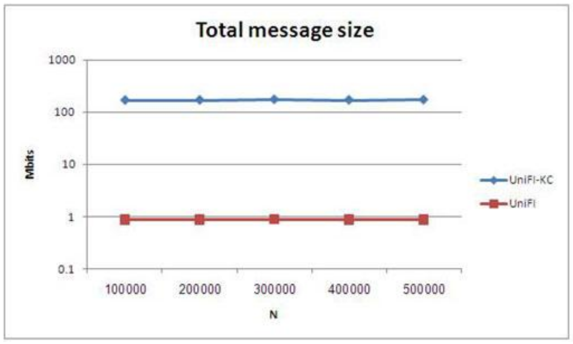


**Fig.1. Computation and communication costs versus the number of transactions *N*.**

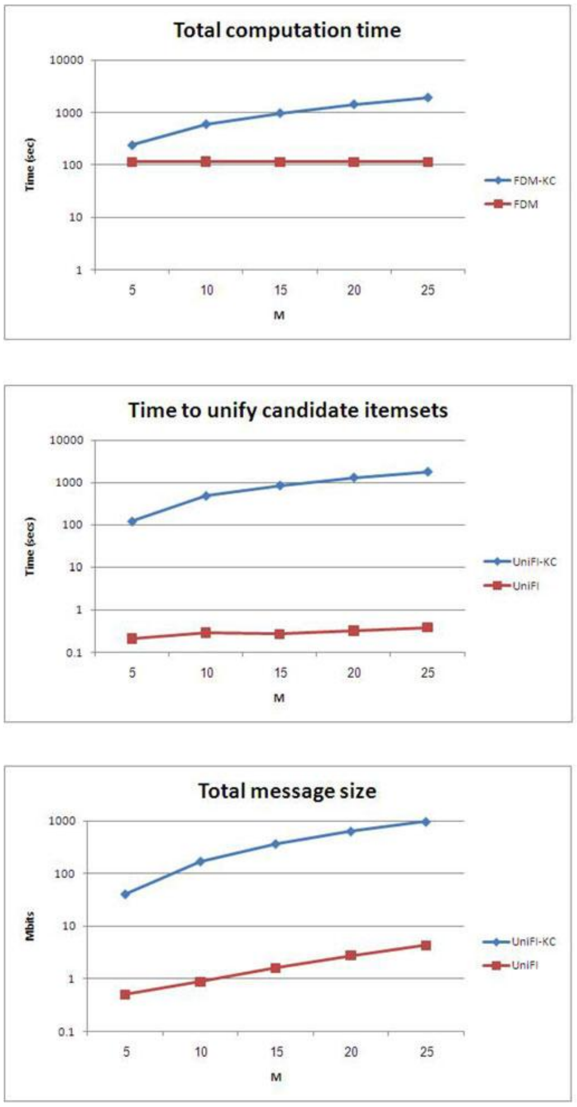


Fig. 2. Computation and communication costs versus the number of playersM.

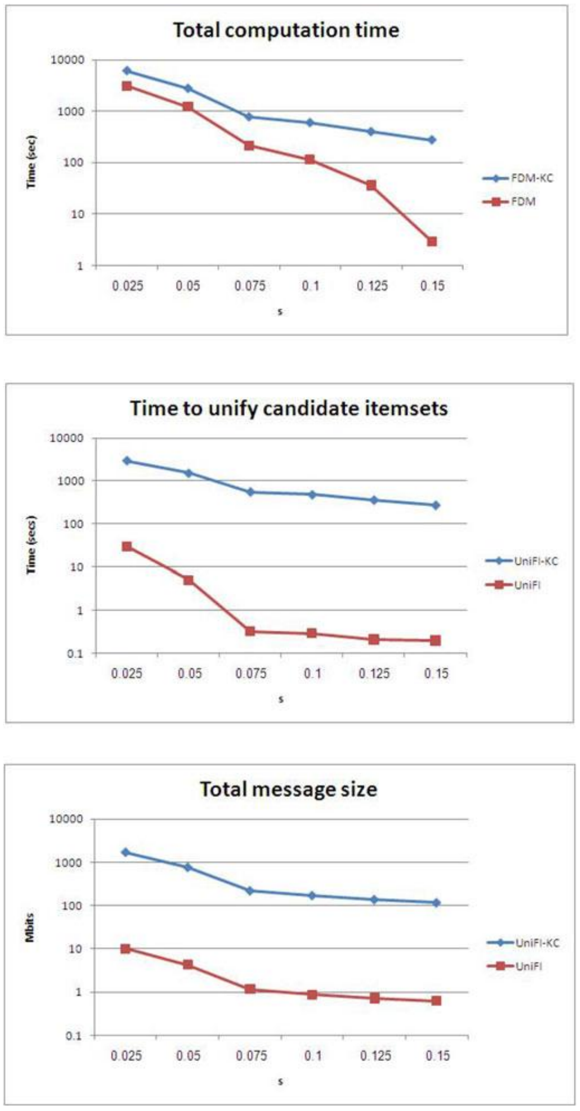


Fig. 3. Computation and communication costs versus the support threshold s.

## VI.CONCLUSION:

We proposed a protocol for secure mining of association rules in horizontally distributed databases that improves significantly upon the current leading protocol in terms of privacy and efficiency. One of the main ingredients in our proposed protocol is a novel secure multi-party protocol for computing the union (or inter-section) of private subsets that each of the interacting players hold. Another ingredient is a

protocol that tests the inclusion of an element held by one player in a subset held by another.
The latter protocol exploits the fact that the underlying problem is of interest only when the number of players is greater than two. One research problem that this study suggests was described in Section 3 namely, to devise an efficient protocol for set inclusion verification that uses the existence of a semi-honest third party. Such a protocol might enable to further improve upon the communication and computational costs of the second and third stages of the protocol of , as described in Sections 3 and 4. Another research problem that this study suggests is the extension of those tech-niques to the problem of mining generalized association rules.

## VII REFERENCES:

[1] Tamir Tassa, ―Secure Mining of Association Rules in
Horizontally Distributed Databases‖, IEEE Transactions on Knowledge and Data Engineering.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In VLDB, pages 487–499, 1994.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In SIGMOD Conference, pages 439–450, 2000.

[4] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In STOC, pages 503–513, 1990.

[5] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In Crypto, pages 1–15, 1996.

[6] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP - A system for secure multi-party computation. In CCS, pages 257–266, 2008.

[7] J.C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In Crypto, pages 251–260, 1986.

[8] J. Brickell and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In ASIACRYPT, pages 236–252, 2005.

[9] D.W.L. Cheung, J. Han, V.T.Y. Ng, A.W.C. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In PDIS, pages 31–42, 1996.

[10] D.W.L Cheung, V.T.Y. Ng, A.W.C. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. IEEE Trans. Knowl. Data Eng., 8(6):911–922, 1996.

[11] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, 31:469–472, 1985.

[12] A.V. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In KDD, pages 217–228, 2002.