DWMiner : A tool for mining frequent item sets efficiently in data warehouses

Bruno Kinder Almentero, Alexandre Gonçalves Evsukoff and Marta Mattoso

COPPE/Federal University of Rio de Janeiro,
P.O.Box 68511, 21941-972 Rio de Janeiro RJ, Brazil
Tel: (+55) 21 25627388, Fax: (+55) 21 22906626
kinder@cos.ufrj.br; evsukoff@coc.ufrj.br; marta@cos.ufrj.br

Abstract. This work presents DWMiner, an association rules efficient mining tool to process data directly over a relational DBMS data warehouse. DWMiner executes the Apriori algorithm as SQL queries in parallel, using a database PC Cluster middleware developed for SQL query optimization in OLAP applications. DWMiner combines intra- and inter-query parallelism in order to reduce the total time needed to find frequent item sets directly from a data warehouse. DWMiner was tested using the BMS-Web-View1 database from KDD-Cup 2000 and obtained linear and super-linear speedups.

1 Introduction

The application of data mining tasks on huge databases requires an increasingly processor and memory capacity. Currently most data to be mined resides in Data Base Management Systems (DBMS). An increasing number of organizations are installing large data warehouses using relational database technology. There is a huge demand for nuggets of knowledge from these data warehouses [16]. Nevertheless, most of the mining algorithms do not operate directly over a data warehouse. The integration of Data Mining (DM) tools with DBMS is now more than a trend, it is a reality. The major DBMS vendors have already integrated DM solutions within their products. In addition, the main DM suites have also provided the integration of DM models into DBMS through modeling languages such as the Predictive Model Markup Language (PMML). It is thus a fact that solutions on new DM tools and methods must consider their integration with DBMS.

In this paper, we present DWMiner, an efficient mining tool to process data directly over a relational DBMS data warehouse. Our solution takes advantage of a cluster of PCs running a Database Cluster middleware.

DBMS query processing techniques have been optimized to take advantage of PC Clusters without having to do a new physical database design through Database Cluster solutions [11] [6] [5]. They preserve the application and DBMS autonomy while providing high performance query processing in PC clusters. The database cluster combines a low cost solution with an excellent performance. Briefly, a database cluster is a middleware between the application and the DBMS that runs on a set of PC servers interconnected by a dedicated high-speed network, each one having its own processors and hard disks, and running an off-the-shelf DBMS [5].

This work addresses the mining of association rules task, more specifically, the search for frequent item sets. The procedure was based on the Apriori algorithm, developed by Agrawal and Srikant [4]. The Apriori algorithm for finding frequent item sets makes multiple passes over the data. Each pass consist of two phases. The first is the candidate generation phase where all the candidate item sets are generated. Then, data is scanned to count, for each transaction, the occurrences of a candidate item set in a transaction. Our implementation simply transforms every database search into an SQL query.

Many parallel algorithms have been proposed based on Apriori. Count Distribution, Data Distribution and Candidate Distribution [3] are some examples. However, these algorithms do not work with a DBMS.

The Apriori algorithm was modified in DWMiner to deal with SQL queries and a DBMS instead. DWMiner executes SQL queries in parallel using a database cluster middleware techniques proposed by Lima et al. [9] and [10]. Such middleware is based on parallel query processing techniques developed for SQL query optimization in OLAP (On-Line Analytical Processing) applications. This database cluster has become an open source solution named ParGRES [11], [13] and is publicly available at http:// forge.objectweb.org/projects/pargres/. Each cluster node can run any non parallel relational DBMS. In this work we use PostgreSQL [14] which is open source. DWMiner combines intra- and inter-query parallelism in order to reduce the total time needed to find frequent item sets directly from a data warehouse. We ran DWMiner using the BMS-Web-View1 database from KDD-Cup 2000 [8] and obtained linear and super-linear speedups in cases where the support threshold is small like, for instance 0.01.

This paper is organized as follows. Section 2 describes the Apriori algorithm used as a basis for our implementation. Section 3 describes how we changed the Apriori algorithm to access a data warehouse and the parallel techniques used in DWMiner. Section 4 describes our prototype implementation and experimental results and Section 5 concludes.

2 The Apriori Algorithm

The problem of mining association rules was initially presented by Agrawal [1] and today is one of the most popular data mining algorithms. Association rule mining, also known as market basket analysis, finds interesting association relationships among a large set of data items. Typically, the data is a set of record where each record represents a transaction containing a set of items. The main goal of the algorithm is to find associations on items that are often present in the same transaction.

Association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. But before describing the procedures that generate association rules we first need to formally define the terms item set, confidence, support and an association rule. According to Agrawal [2] an association rule is an implication of the form X => Y where X and Y are sets of items. The intuitive meaning of such a rule is that transactions of the database which contains X tend to contain Y. An item set is a set of items in a lexicographic order. A k-itemset is an item that contains k items. Support and confidence are the two measures most used in association rules. The support or the occurrence frequency of an item set is the number of transactions that contains the item set. This is taken to be the probability, $P(A \cup B)$. A frequent item set is an item set with a support value higher than a minimum threshold. The confidence of a rule X => Y is the percentage of transactions that contains X and also contains Y. This is taken to be the conditional probability, $P(B \mid A)$, i.e.:

Support (X => Y) = $P(A \cup B)$

Confidence (X => Y) = $P(B \mid A)$

The algorithm of mining association rules can be divided in two sub problems: (i) find all the combinations of items whose support are higher than the minimum support, called frequent item sets; and (ii) find the association rules with confidence greater than or equal to the minimum confidence, based on frequent item sets generated earlier. We are particularly interested in the first sub problem: finding the frequent item sets. There are many algorithms to generate frequent item sets such as the AIS [1], the SETM [7] and the AprioriTid [4]. Among these algorithms the Apriori is considered one of the most important and widely used. So, we chose Apriori to be the basis of our implementation. The pseudo-code for the Apriori algorithm is as follows.

```
Input: Database,D, of transactions; minimum support threshold
min_sup
Output: frequent item sets in D
$C_k$: Candidate item set of size k;
$L_k$: frequent item set of size k;

1.  $L_1$ = {frequent 1-itemsets};
2.  for ($k$ = 2; $L_{k-1}$ !=∅; $k$++) {
3.     $C_k$ = candidates generated from $L_{k-1}$;
4.      for each transaction $t$ in database do{
5.        increment the count of all candidates in $C_k$
6.        that are contained in $t$
7.      }
8.    $L_k$ = candidates in $C_k$ with min_sup
9.  }
10. return $\cup_k L_k$;
```

Step 1 of Apriori finds L_1, the frequent 1-itemsets (line 1). In the next step the frequent item set L_{k-1} is used to generate the candidate k-itemsets C_k (line 3). Then, the dataset is scanned to find the support values for candidates (lines 4 to 7). Finally, the frequent k-itemsets are determined (line 8). The final solution is the union of the frequent k-itemsets (line 10).

3 Apriori implementation in DWMiner

Discovery of association rules is an important Data Mining problem. Parallel algorithms are required [3] to cope with the databases to be mined which are often very large (measured in gigabytes or even in petabytes). However, most of the parallel solutions do not deal with a DBMS. In DWMiner we combine parallel techniques with DBMS advantages to efficiently mine frequent item sets from large databases. In this section we present how we adapted Apriori to issue queries to run in a database cluster.

3.1 Database Clusters

Database Cluster is a middleware that provides parallel query processing in applications that use a sequential DBMS [15]. In a database cluster, each node of the cluster runs its own sequential DBMS as a *black-box* component. Clients submit transactions to the middleware which is responsible to distribute queries through the cluster nodes.

Parallel query processing of database clusters is based on two techniques known as intra-query and inter-query parallelism. In intra-query parallelism, a query is decomposed in sub-queries that scan different subsets of the data. The sub-queries are executed in parallel in the cluster nodes. Fig. 1 (a) shows an example of intra-query parallelism where the query Q_1 is decomposed in n sub-queries. The database is replicated at all nodes involved with the intra-query processing. Each sub-query is responsible to process a different range of data at each node in parallel. Finally the sub-results are combined to produce the final query result. This technique aims to reduce the execution time of heavy-weight queries, i.e., queries that access large amounts of data and may perform complex operations, thus taking a long time to be processed. In the inter-query technique, queries are executed as they are really, which means no decomposition. Distinct queries are distributed and executed concurrently in the cluster nodes to enhance database system throughput. Fig. 1 (b) shows an example of inter-query parallelism where queries Q_1 to Q_n are distinct and distributed over the cluster nodes to be executed in parallel.

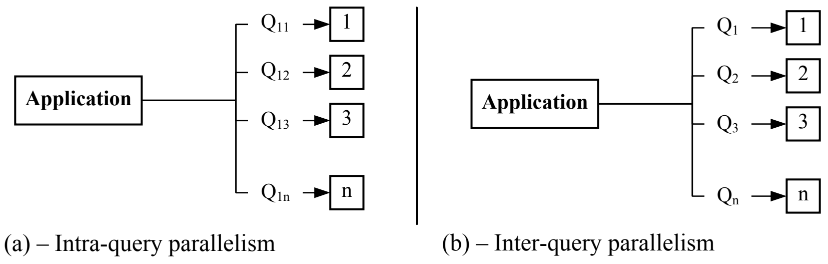

Fig. 1. Parallel query processing techniques

These two techniques are not exclusive, but most database clusters provide either inter-query [6], [15] or intra-query [5] parallelism. However, they have been successfully combined in [10] and [11], so in DWMiner both inter and intra-query parallelism are explored. When receiving a heavy weight query we can use intra-query parallelism, and, in the case of simple queries, the inter-query parallelism should be more appropriate. In addition, a query being processed by intra-query parallelism can run concurrently with other queries through inter or intra-query parallel processing.

3.2 Adapting Apriori to DBMS Access

Now we describe how we changed the Apriori algorithm to generate SQL queries, instead of reading data from a flat file to main memory. First of all we name a set of items which contains *k* items as a *k*-item set. Hence, the first step of the algorithm (*k=1)* is to generate the *1*-itemset and find the support for each element. Thus, we simply have to scan all the transactions in order to count the number of occurrences of each item. In our case, the table was called *bmswebview1* with two attributes: *a_item* and *a_tid,* where *a_item* is the item identification and *a_tid* the transaction identification. The SQL query generated by our implementation is Q_1, described as follows. This query corresponds to the line 1 of the pseudo-code described earlier.

```
Q1: Select   a_item, count(*) as total
    from     bmswebview1
    group by a_item
    having count(*) >= minimum_support
```

In Figure 2, we show the architecture of the typical Apriori algorithm and DWMiner with Apriori accessing data to be mined directly from the data warehouse through a DBMS driver interface.

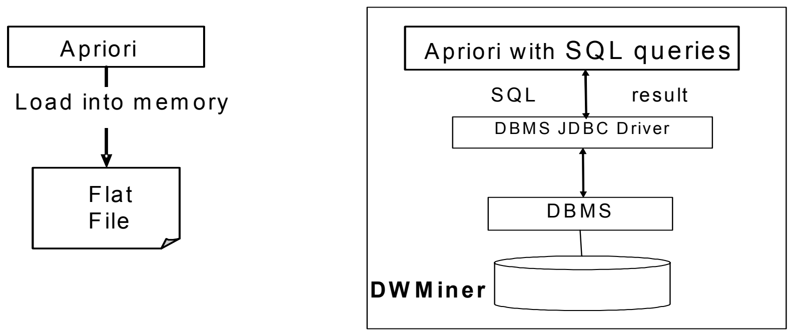

Fig. 2. Apriori data access and DWMiner database cluster access

The next step is to generate the candidate 2-item sets from which we will find the frequent 2-itemset. In this case the SQL query Q2 generated by DWMiner is described as follows.

```
Q2: Select count(a_tid)
    from   bmswebview1
    where  a_item = item1
    and exists (select a_tid
                from   bmswebview1
                where  a_item = item2)
```

This query Q2 corresponds to the steps 4 to 6 in the Apriori pseudo-code. So, we are counting the transactions that contain both *item1* and *item2.* The number of queries generated is equal to the number of candidate 2-item sets. This process continues until there is no more candidate item sets left. Every query generated in this loop corresponds to the steps 4 to 6 in the Apriori pseudo-code. They will be different depending on the value of k, from the current k-itemset being analyzed. Thus, to generate the query to find the 3-itemset support we just need to add one more level of nested select in Q2 generating the following query.

```
Q3: Select count(a_tid)
  from   bmswebview1
  where  a_item = item1
  and exists (select a_tid
              from   bmswebview1
              where  a_item = item2
              and exists (select a_tid
                          from bmswebview1
                          where a_item = item3)
```

Then, the number of nested selects is directly related to the item set being analyzed. If we are analyzing the k-item set then we will have k levels of nested selects. Once a candidate item set is created we can build SQL queries for each element and process them in parallel because they are independent.

3.3 Adapting Apriori to Database Clusters

The main goal of DWMiner is to reduce the total time of database searching. In order to do that, DWMiner adopts inter and intra-query parallelism available in ParGRES database cluster. Intra-query parallelism is obtained by using a virtual partition technique (VP) [10]. This technique breaks one heavy weight query into sub-queries by adding selection predicates as proposed in [5]. Each DBMS receives a sub-query and is forced to process a different subset of data items. Each subset is called a "virtual partition".

The SQL Q_1 query generated in the first step of the Apriori algorithm to find the frequent 1-itemset involves a *group by* and a *having* operation. Such operations are time consuming since a full scan on a large relation is needed. To overcome this problem at this point DWMiner takes advantage of intra-query parallelism involving all of the cluster nodes. Thus, the Q_1 generated for the first step would be rewritten by the database cluster as the following Q_{1i} sub-queries, where i varies from 1 to n being the number of nodes involved on the intra-query processing.

```
Q1i: Select a_item, count(*) as total
     from bmswebview1
     and  bmswebview1_key > :v1 and bmswebview1_key <= :v2
     group by a_item
     having count(*) >= minimum_support
```

The difference between Q_1 and Q_{1i} is the range predicate "bmswebview1_key > :*v1* and bmswebview1_key <= :*v2*". We call *virtual partitioning attribute* (VPA), the attribute chosen to virtually partition the data. Usually the VPA is the primary key of the table being selected, in this case bmswebview1_key. The values used for parameters *v1* and *v2* vary from node to node and are computed according to the total range of the VPA and the number of nodes. Let us assume that the interval of values of bmswebview1_key is [1; 6,000,000] and we have 4 nodes, then, 4 sub-queries must be generated. The intervals covered by each sub-query are the following: Q_{11}: *v1*=0 and *v2*=1,500,000; Q_{12}: *v1*=1,500,000 and *v2*=3,000,000; and so on. In spite of each node having the same replica of bmswebview1 table, virtual partitioning forces each one to process a different data subset of bmswebview1. Besides, full replication makes it possible to allocate any node to process any sub-query. After sub-query execution, it is necessary to compose the partial counting produced by each one in order to have the final result.

In Fig. 3, we show the architecture of the Apriori algorithm and DWMiner with respect to accessing data to be mined through a database cluster middleware. In this case DWMiner is issuing query Q_1 to the database cluster, which decides to process it through intra-query parallelism. Thus Q_1 is decomposed as Q_{1i} sub-queries to access n different virtual partitions of table bmswebview1. Such middleware can be C-JDBC or ParGRES or any other database cluster. However, if C-JDBC is used, Q1 cannot be processed through intra-query parallelism.

For the queries of the following steps of Apriori, DWMiner tries to find a balance between inter and intra-query parallelism. For example, once the k-item set is analyzed, a candidate item set is created. Each element query can be processed independently in parallel through inter-query. Therefore, DWMiner sends each query to a cluster node. However, the time needed to process one such query may be relatively large. In this case, the query is decomposed and its sub queries are processed in parallel.

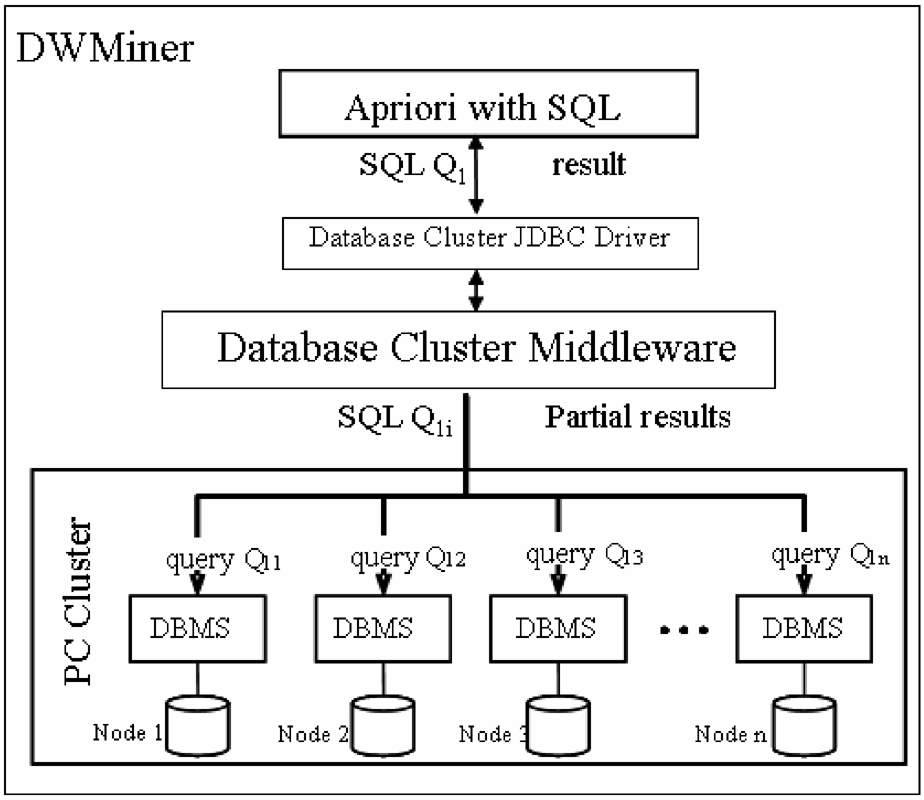

Fig. 3. DWMiner using Apriori accessing a database cluster

4 Experimental Results

To evaluate DWMiner techniques we have used a Linux based PC cluster and PostgreSQL 8.0 DBMS [14]. The dataset used in our experiment is the BMS-Web-View1 which contains several months' worth of click stream data from an e-commerce web site. A portion of their data was used in KDD-Cup 2000 competition [8]. This dataset has a total of 56,902 transactions and 497 distinct items, its maximum transaction size is 267 and the average transaction size is 2.5. Our experiments run on top of the cluster system of the Paris team at INRIA [12]. Our tests have used up to 32 nodes of this cluster system, each node configured with dual

2.2 GHz Opteron processors with 2 GB of main memory. The cluster is interconnected by a standard Ethernet network.

The results from our experiments are shown in Fig. 4. We plot times taken by our implementation for values of support ranging from 0.1% to 2% using only inter-query parallelism. We ran DWMiner varying the number of nodes from 1 to 32 for each support value. In order to improve reading and analysis, we use logarithmic scale. Although, DWMiner implements its own inter-query mechanism we also used C-JDBC[6] to perform inter-query parallelism as an alternative successful open source database cluster solution.

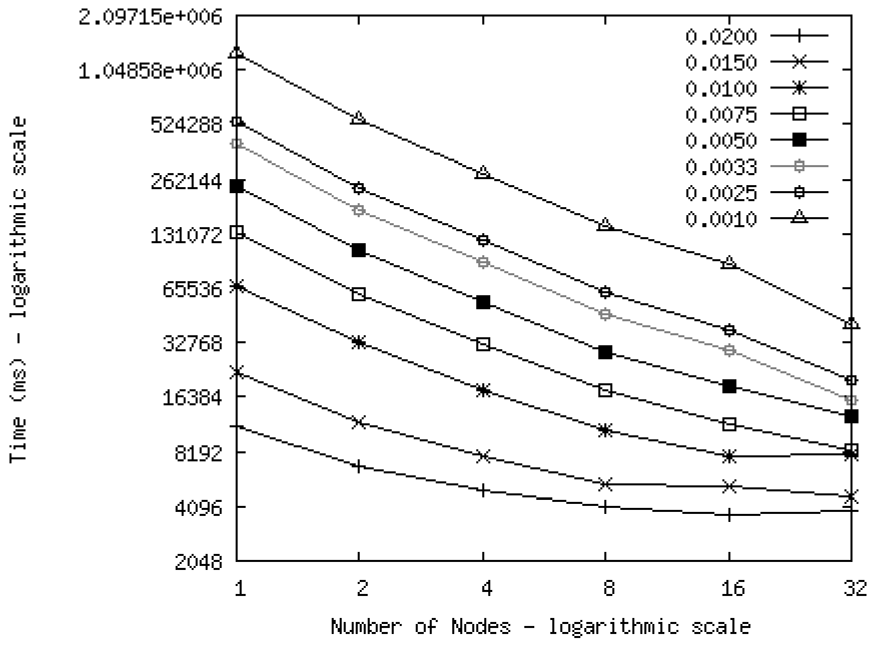

Fig. 4. Execution times for DWMiner

Most of the results in Fig. 4 present linear speedup as we increase the number of nodes, since queries sent to the database cluster are independent from each other. But, analyzing the higher support curves like 0.02 (2%) and 0.015 (1.5%) we note that the results are worse than linear. This happens because the number of candidates generated and, consequently, the number of queries is not enough to compensate the time spent to distribute these queries over the cluster nodes and receive the results. Still, DWMiner does not experience slow down factors. Table 1 gives a more accurate view of the graphic shown in Fig. 4. In the worst case, using 32 nodes is 4 times faster than using 1 node.

As we can see in Table 1, by using two nodes the execution time of DWMiner is reduced by almost 50% for the support 0.02 (2%). However, when 4 nodes are used the time reduction is linear and the execution time remains almost the same until 32 nodes. This happens because when we use 4 nodes we get too close to the situation where the time spent to distribute the queries and wait for the results is the main factor in the total time of execution. However, as the support threshold decreases, the time reduction continues to improve the performance and it is often super-linear.

Table 1. Query Execution times for DWMiner

	Number of Nodes					
Support	**1**	**2**	**4**	**8**	**16**	**32**
0.0200	11,354	6,842	4,959	4,034	3,672	3,867
0.0150	22,581	11,991	7,618	5,328	5,274	4,648
0.0100	67,707	32,651	17,878	10,821	7,653	7,952
0.0075	132,153	60,715	31,801	17,989	11,678	8,355
0.0050	238,717	105,636	55,043	29,326	18,722	12,891
0.0033	411,360	177,668	90,894	47,575	29,724	15,564
0.0025	540,933	234,830	119,517	61,941	38,767	20,246
0.0010	1,291,949	555,865	282,049	144,623	88,928	41,662

Table 2 shows the performance improvement we obtained in each case. We can see in Table 2 that most results are *quasi*-linear or super-linear. When we use 2 nodes and the supports going from 0.01 (1.0 %) to 0.001 (0.1%) the support is lower enough to generate a relatively large number of candidate item sets. For these support thresholds, a large number of queries are generated and sent to the nodes. When many queries are sent to a node the database cluster makes a wise use of the system cache instead of reading data from disk. Thus, many queries process data from memory reducing considerably the query execution time achieving, this way, super-linear speedups.

Table 2. Perfomance evaluation of DWMiner

	Number of Nodes (Linear Speedup)				
Support	**2 (50.00%)**	**4 (25.00%)**	**8 (12.50%)**	**16 (6.25%)**	**32 (3.13%)**
0.0200	60.26%	43.68%	35.53%	32.34%	34.06%
0.0150	53.10%	33.74%	23.60%	23.36%	20.58%
0.0100	48.22%	26.40%	15.98%	11.30%	11.74%
0.0075	45.94%	24.06%	13.61%	8.84%	6.32%
0.0050	44.25%	23.06%	12.28%	7.84%	5.40%
0.0033	43.19%	22.10%	11.57%	7.23%	3.78%
0.0025	43.41%	22.09%	11.45%	7.17%	3.74%
0.0010	43.03%	21.83%	11.19%	6.88%	3.22%

The graphic in Fig. 5 compares the results of inter-query only by using C-JDBC with the results of intra-query combined with inter-query through ParGRES. We also compared inter-query only using C-JDBC and inter-query only using ParGRES. In both implementations queries are distributed to cluster nodes in a round robin fashion. We obtained very similar results in both database clusters. Therefore, in Fig. 5 we kept the legend as inter *versus* inter/intra rather than C-JDBC *versus* ParGRES. In the combination case, intra-query was implemented using only two nodes. Queries were decomposed in two sub-queries and executed in parallel in the cluster concurrently with other queries. So, when running with 32 nodes, it means that 16 different queries can be executed in parallel. However, intra-query demands an aggregation phase for each query to compose the two partial results.

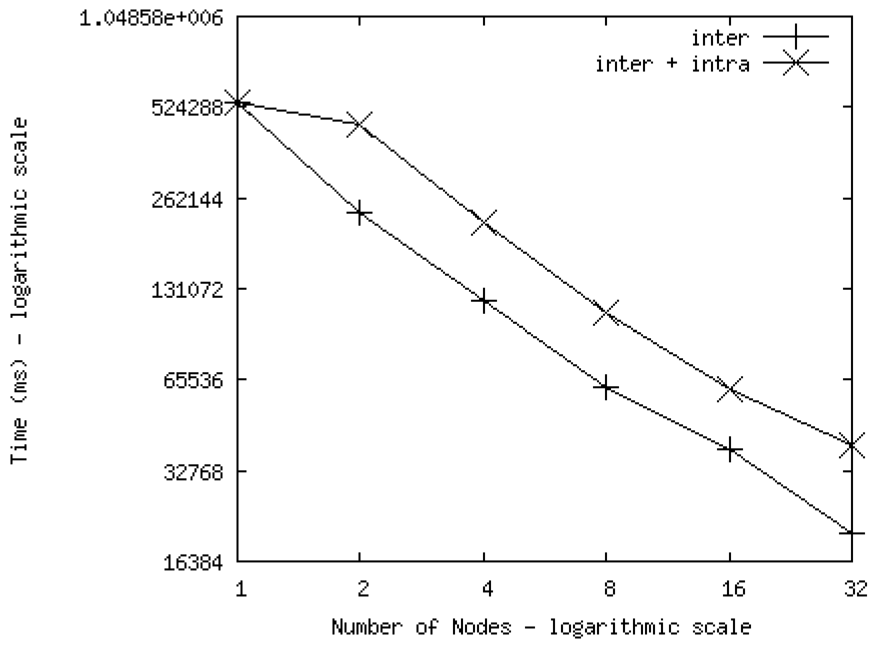

Fig. 5. Inter-query versus (inter + intra) query

As shown in Fig. 5, the inter-query parallelism alone is better than the combination between inter and intra-query parallelism. Since we are using a relatively small database, individual queries could not be considered to be heavy weight queries. So, the time needed to aggregate the partial results of the sub-queries was relevant with respect to overall query reduction. Nevertheless, the combination of inter with intra-query achieved linear and super-linear speedups.

5 Conclusions and Future Work

One of the best advantages in using a DBMS is that it already provides efficient techniques to deal with large datasets. These techniques need to be re-implemented in part if we want to work with flat files that do not fit in the available memory.

Most of the mining algorithms demand a flat file to be in a special format. These algorithms need an extra step to extract the information they need to a flat file. Since we can have data warehouses with dozens of gigabytes or even petabytes, to generate a file from these data may require a lot of extra storage. DWMiner solution is DBMS vendor independent, thus it can be applied directly over a data warehouse system using techniques that take advantage of a low cost high performance scenario such as database clusters.

In this work we showed that by using such techniques we acquire significant improvement in the process of mining data directly from a DBMS. We can efficiently mine frequent item sets from a data warehouse by sending queries to be processed in parallel by the database cluster. In our experiments, we have used one representative dataset – BMS-Web-View1 – and as future work we intend to test DWMiner against some larger databases where we expect to explore the combination of inter and intra-query parallelism and take more advantage of the intra-query parallelism. Nevertheless, we achieved linear and super-linear results working with a relatively small dataset comparing to a real data warehouse.

The techniques adopted in DWMiner are not difficult to implement and maintain since they are based on SQL and take advantage of simple parallel techniques found in database clusters. In addition, DWMiner solution is all based on open-source software and commodity hardware. Such techniques can also be applied in tasks different from mining frequent item sets inside the data mining context.

Acknowledgements

The authors are grateful to the Brazilian research agencies CNPq, CAPES and FINEP for the financial support of the work. We are also grateful to the Paris team at INRIA for providing the PC cluster environment.

References

1. Agrawal, R., Imielinsk, T., Swami, A. N., 1993, "Mining association rules between sets of items in large databases". In: *1993 ACM SIGMOD International Conference on Management of Data*, pp.207-216.
2. Agrawal, R., Mannila, H., Srikant, R., et al, 1996, "Fast discovery of association rules". In U.M.Fayyad, G.Piatetsky-Shapiro, P.Smyth, and R.Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, chapter 12, AAAI/MIT Press.
3. Agrawal, R.,Shafer, J., 1996, "Parallel Mining of Association Rules", *IEEE Trans.Knowledge and Data Engineering*,v.8, pp.962-969.
4. Agrawal, R.,Srikant, R., 1994, "Fast algorithms for mining association rules". In: *20th International Conference on Very Large Databases (VLDB)*, pp.487-499.
5. Akal F., Böhm, K., Schek, H. J., 2002, "OLAP Query Evaluation in a Database Cluster: a Performance Study on Intra-Query Parallelism". In: *East-European Conference on Advances in Databases and Information Systems (ADBIS)*, Bratislava, Slovakia.
6. C-JDBC. In: http://c-jdbc.objectweb.org/, Accessed in 2005.
7. Houtsma, M.,Swami, A., 1995, "Set-oriented mining of association rules". In: *11th Conference on Data Engineering*, Taipei, Taiwan.

8. Kohavi, R., Brodley, C. E., Frasca, B., et al., 2000, "KDD Cup 2000 Organizers' Report: Peeling the Onion", In: *SIGKDD Exploration* 2 (2), pp.86-98.
9. Lima, A. A. B., Mattoso, M., Valduriez, P., 2004, "OLAP Query Processing in a Database Cluster". In: *10th Euro-Par Conference*, pp. 355-362.
10. Lima, A. A. B., Mattoso, M., Valduriez, P., 2005, "Adaptive Virtual Partitioning for OLAP Query Processing in a Database Cluster". In: *19th SBBD*, pp.92-105.
11. Mattoso, M., Zimbrão, G., Lima, A. A. B., Almentero, B.K. et al., 2005, "ParGRES: a middleware for executing OLAP queries in parallel". In: COPPE/UFRJ Technical Report ES-690, *http://pargres.nacad.ufrj.br/Documentos/ES-690.pdf*.
12. Paris Project. In: http://www.irisa.fr/paris/General/cluster.htm.
13. ParGRES In: http://pargres.nacad.ufrj.br/, Accessed in 2005.
14. PostgreSQL. In: http://www.postgresql.org, Accessed in 2005.
15. Röhm, U., Böhm, K., Schek, H. J., 2002, "FAS - A Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Components". In: *28th International Conference on Very Large Data Bases (VLDB2002)*, pp.754-765.
16. Sarawagi, S., Thomas, S., Agrawal, R., 1998, "Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications". In: *1998 ACM SIGMOD International Conference on Management of Data,* pp.343-355.