Enhancement of a data warehouse performance using association rules technique

Walid Moudani, Mohammad Hussein, Mirna Moukhtar, Felix Mora-Camino

Enhancement of a Data Warehouse Performance using Association Rules Technique

Walid Moudani
Lebanese University
Doctorate School of Sciencess and Technologies
Tripoli, Lebanon

Mohammad Hussein
Lebanese University
Doctorate School of Sciencess and Technologies
Tripoli, Lebanon

Mirna Moukhtar
Lebanese University
Doctorate School of Sciencess and Technologies
Tripoli, Lebanon

Félix Mora-Camino
Air Transportation Dept., ENAC/DGAC, 7 Avenue Edouard Belin, 31055 Toulouse, France.

ABSTRACT

The data warehouse holds information management and turns it into meaningful management information, from which, very interesting patterns can be discovered by applying knowledge discovery process. As the update of the Data Warehouse is not too frequent, it is possible to improve query performance while storing the data retrieved by them in a cache. However, the most powerful systems have a small capacity to store the entire database in memory cache. The caching chunks technique is designed to keep in cache the query results in the form of chunks of values, instead of storing them in large tables. In this paper, we propose a new technique for caching multidimensional queries based on association rules. Using this technique will allow all users to enjoy the benefits of Data Warehousing in the best manner, and also to improve performance and also increase the use of the system while reducing the response time. The technique is build using an architecture comprising a data warehouse, a memory cache on the server and a one on each user's machine, in which the association rules and query results are stored. These results are kept in the form of chunks to enjoy all the advantages of the technique of fragmentation into chunks. This approach has been implemented and tested over a real huge data followed by displaying the results and analyzes.

Keywords

Data Warehouse, Data Mining, OLAP, Association Rules, Cache based on chunks

1. INTRODUCTION

In the modern business environment, Business Intelligent System should provide data access for managers, analyzers and decision makers. Business Intelligent System has become a key contributor to mark a significant competitive advantage for companies. It includes a Data Warehouse (DW) environment [9], a system for On Line Analytical Processing (OLAP), tools for Data Mining (DM) [2, 12, 14, 15, 17], etc. Many companies have adopted DWs from operational databases. Thus, the DW associated to analysis tools OLAP are an effective solution for business intelligence. These systems are based on the multidimensional paradigm, which, based on the concepts of dimension, fact, measurement and OLAP operators, enable a multidimensional analysis of large amounts of data. There are two types of DW: ROLAP in which data is represented in a relational database and MOLAP in which data is represented in multidimensional data cubes [1]. In this work, we focus on the second one. The large size of most DWs leads to a significant cost of queries processing and leads us to find a cache efficient technique for reducing delay in multidimensional processing. The DW stores information that is collected from multiple heterogeneous information sources to answer complex queries and analyze information [5, 6, 8, 10, 16]. It collects copies of data from remote data sources and integrates this information into a repository for reporting and monitoring of strategic decisions. An OLAP system provides tools to explore and navigate through data cubes to extract valuable information [6]. Systems or applications of decision support running both queries in multidimensional data using complex techniques, and also use DM techniques to extract valuable information and useful in the considered application domain. DM techniques provide powerful extraction and knowledge discovery such as Association Rules (ARs), classification, and segmentation, etc. A DM algorithm is a well-defined procedure that takes input data and produces output as templates [1]. The DM algorithms allow the user to find partial information of great importance from the DW. Nevertheless, OLAP is not capable of explaining relationships that could exist in a DW. ARs are a kind of DM techniques to find associations between data.

In this work, we present a new technique for caching multidimensional queries. This technique is based on extracting the necessary information based on ARs, caching the queries in order to reduce the processing time of them. Also, we propose a new replacement policy named MLAR, Multi-Layer caching based on ARs. It is based on a combination of cache replacement policies, already proposed in the literature with ARs leading to improve replacement requests when the cache becomes full. The remainder of the paper is organized as follows. In Section 2, we describe the context and the related works on caching. The section 3 contains a detailed explanation of our caching technique by illustrating our policy MLAR. Section 4 discusses the performance evaluation. Finally, in section 5 we conclude and we describe future works.

2. CONTEXT AND RELATED WORKS

This section is divided in two parts: in the first part, we present the context of our work. The second part discusses the related works.

2.1 Context

Businesses are often subjected to development and expansion by continually creating branches and subsidiaries spread geographically. Consequently, a single centralized DW may be too costly or difficult to build [14, 15, 18]. Therefore, many companies tend to use a number of smaller DWs, located at a distance. So in order to answer the queries, the decision support systems carry on the requested task by performing aggregation operations on data stored on these small DWs which are located either at the user node, or at the data server (Middle-Tier). Usually, each node collects data independently. The major problem of distributed OLAP system is the query processing cost. So we have a technique that reduces the response time for distributed OLAP queries. Caching is an essential solution to improve the performance of many applications since the primarily use of repeated data is expensive, specifically in computing and fetching data. Thus, by caching the data, the application needs to calculate or retrieve the data once. When data is queried, the application can retrieve data from the cache instead of recalculating or fetching them from a DW located remotely. Due to the fact that a small number of queries are of fundamental importance and are most often requested, and the fact that these queries are related to ARs that are most frequently requested. Then it is efficient to store these rules in the cache of the user (client cache), or even in the server cache (Middle-Tier), so that these data are now easily accessible by all users with a reduced response time. However, due to the large volume of data requested by users through all possible queries and across different levels of dimensions, it is more efficient to store query results with the related ARs in the cache.

In the remainder of this paper we based on a example (Figure 1) illustrating the drugs sales in a large pharmacy. In this example, we considered three dimensions: Drugs, Branch, and Time, and a fact table. The table 1 describes the dimensions and their respective hierarchies. The model of the dimension hierarchy is as follows:

d_i = dimension of order « i »; i = 1, ... , N
N = number of dimensions
L_i = total number of levels for dimension d_i

Here, the level varies from 0 to li - 1. Level 0 indicates that the dimension is aggregated at all. If the level value increase, the concept of hierarchy is explored to view more detailed information on one dimension. Table 1 shows the three dimensions and their hierarchy levels used as a running example in this work. This allows an OLAP decision maker to display information of the sale at various levels and combinations of dimensions in a hierarchy, and identify the drugs/ Branch /time which present exceptional circumstances. We present in the following dimensions of the considered DW through a real data.

Drugs (Drug_key, Drug_Description, Drug_Category, Drug_Brand)
Branch (Branch_key, Branch_Country, Branch_Region, Branch_City)
Time (Time_key, Time_Year, Time_Quarter, Time_Month)
FactProfit (Time_key, Drug_Key, Branch_Key, Profit, ProfitLevel)

Table 1. Dimensions' hierarchies and their levels

	Drugs	**Branch**	**Time**
Level	Dg_Key	Br_Key	Ti_Key
0	ALL	ALL	ALL
1	Dg_Category	Br_Country	Ti_Year
2	Dg_Brand	Br_Region	Ti_Quarter
3	Dg_Description	Br_City	Ti_Month

2.2 Related works to caching technique

The use of cache is widely deployed to minimize the expenses incurred during the operations distributed OLAP [6, 7, 13, 18]. Thus, the cache can be used in OLAP systems to reduce processing time. In this section, we present the importance of caching OLAP queries, the chunk caching technique. This technique introduced the decomposition of the multidimensional space into chunks where each dimension is divided into separate ranges [3].

Thus, a mapping structure called Domain Index is defined to maintain the correspondence between a dimension value and its ordinal number. The idea of using chunks comes from its utility for systems that use MOLAP multidimensional arrays to represent data. Thus, instead of storing a large table to a particular row or column, it is broken into chunks and stored in an appropriate chunked format [18, 19]. Distinct values for each dimension are divided into ranges and the chunks are created based on this division. In the chunk based-caching system, query results, stored in cache, should be cut into chunks to be cached. When a new query is requested, the chunks needed to respond this request are determined. According to the contents of the cache, the list of chunks is divided in two parts. A part is extracted from the cache.

The other part consists of the missed chunks in the cache that must be fetched and calculated from other data sources. It is important to reduce the cost of a missed chunk. This means that the missing chunks are to be calculated efficiently from other data sources.

Consider the following query (Q1) which demands to know the monthly rental of a Drug category, for the first two quarters. (Between January and June) we consider that the results of the query (Q1) are cached. Now, if we have two queries Q2 and Q3 which are sought after Q1. Q2 demands the sum of profits from the rental information for the months between January and May, while Q3 demand the sum of profits from the rental information for the months between April and September.

To answer Q2 and Q3 (Figure 2), the query evaluator is presented with two alternatives. The first solution is to evaluate queries using query results cached while the second alternative is to recover from original data sources. The traditional approach of caching was that leads to cache the entire results of the query. It analyzes the contents of the request to determine whether a given query can be answered using the cache. For example, Q2 can be fully evaluated using the cache because it is contained in Q1. However, if we analyze the contents of the query Q3, it cannot take advantage of cached results, despite the fact that some partial results are cached, i.e., information rent for the months of April, May and June To overcome the drawbacks of caching query level, we need a mechanism to determine if partial results for the query can be retrieved using the cache. In addition, we need to decompose the query so that a portion is evaluated entirely in the cache while the other is delivered to remote data sources. A naive method to determine if the partial results of a query are in the cache requires overlapping the application requested with all queries cached. The model chunk-based overcomes these problems by dividing the multidimensional query space uniformly into pieces, and then place them in cache. The query results are contained in a whole number of chunks, because the pieces are at the lowest level of granularity that caching query level can be reused to compute the partial result of an incoming request.

Using a fast mapping between the constants of queries and the number of chunks, we can determine all the pieces necessary to fully answer a query. Since query results are associated with an integer number of chunks, the replacement policy can benefit from hosting the chunks to the difference of caching query level.

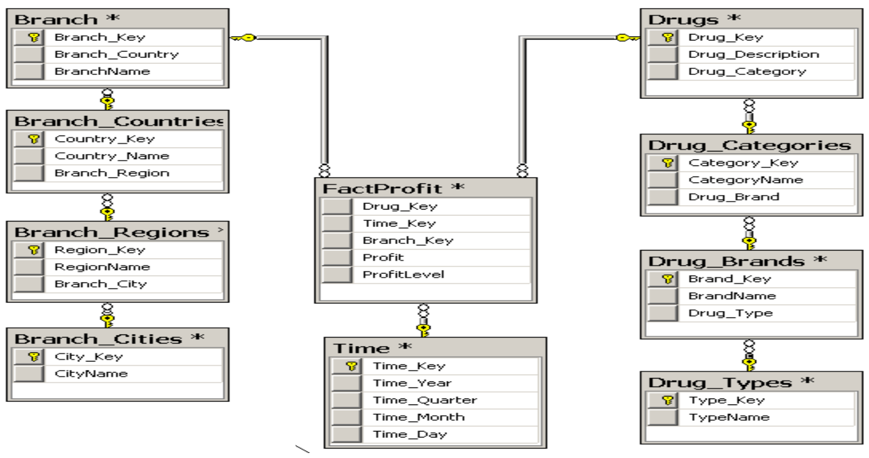

Fig 1: Drugs sales schema

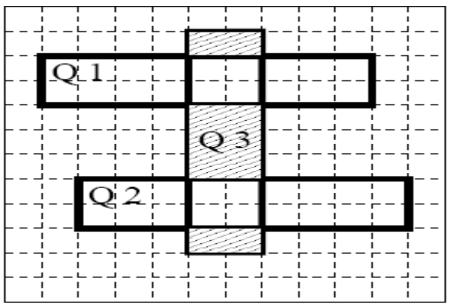

Fig 2: Reusing of cache.

The advantages of Chunk caching approach can be summarized as following: (i) Granularity of cache: Caching based chunks instead of entire queries improves the granularity of caching, (ii) Uniformity: The notion of regions semantically uniform in the form of chunks of query allows a reuse less complex, (iii) Closure property of chunks: The technique of chunking can be applied to any level of aggregation. Consistency provides a simple mapping between chunks at different levels of aggregation, and (iv) Avoid redundant storage: If the cache is used so that each query is cached in its entirety, although some requests will have results that overlap with the results of other queries. Replication of these partial results reduces the effective memory available for caching. Therefore, the success rate of the cache is negatively affected by reducing the advantage of having a large cache.

3. DESCRIPTION OF THE PROPOSED SOLUTION METHOD

In order to improve the response time for users on different distributed machines based on three-level architecture composed of 3 layers: an End user layer (client machine), a layer of data processing (OLAP Server) and a layer of data storage (Data Warehouse Server). We tried to capture all the benefits of the caching techniques, and at the same time preserve the performance of the system, while trying to minimize the cost in terms of response time as shown in our study. This allowed us to combine firstly, the benefits of caching query results into chunks, and secondly, the use of a cache replacement algorithm based on combined policies. The principle is to keep only in cache the most requested ARs by users (including their associated results) and maintain the data source up to date as much as possible. The technique of ARs uses large itemset property. It is easily parallelized and easy to implement. Moreover, it permits to select the appropriate level of information needed by most decision makers. The search process is interleaved between the examinations of ARs, and then determines a drill-down path to a more appropriate level of detail. The remainder of this section is organized as following: firstly, we describe the system architecture and its functions. Secondly, we introduce the database schema to storing the ARs. Thirdly, we present the Multi-layers caching mechanism and the replacement policy.

3.1 Description of the System architecture and its functions

We present the three-tier architecture shown in Figure 3. The system consists of a DW, a middle-tier server and remote distributed users. OLAP data are stored in the DW, as well as ARs. As multidimensional queries require non-negligible execution time, it became necessary to reuse the results already calculated and stored in a caching based on multilayer mechanism instead of sending these queries to the DW. The middle tier server is connected to the DW via a wired connection and acts as a communication link between the DW and the client computers. The middle tier server and the client machine are equipped of cache to store the ARs related to requested queries and the associated results.

At a new request by a user for a multidimensional query or an AR, this request is sent to the cache of the user to locate the requested information (if any). In case of existence, the cache sends the requested information to the user in an access time remarkably reduced. If the information does not exist in the cache of the user, the request will be sent to the next level of architecture that corresponds to the intermediate server. Similarly, the searched information corresponding to this request is done first in the intermediate server's cache. In case of existence, it transmits the information found at the intermediate cache server to the user who asked this query in a short time. However, a copy of the ARs and its results are stored in the cache of the user. In cases where no information is found in the cache of the intermediary server then the request is

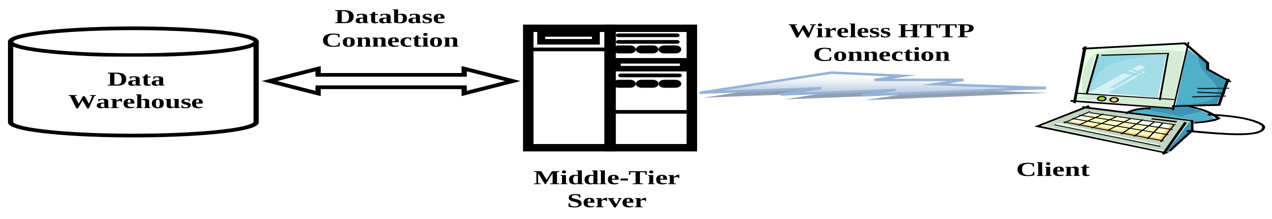

Fig 3 : Three-tier architecture system

forwarded to the DW where it processes the request and sends its results to the user via the intermediary server. This keeps a copy of the AR and its associated results in the cache of the user and middle-their server. Recovery time of the results was significantly higher compared to the case where information is found in the caches associated with other levels of the architecture. User computers include a cache that serves reducing the access time of a request.

To manage the cache in best manner, we propose a description of the replacement policies that have been adopted in literature. Thus, they can replace queries already cached when the cache becomes full. In general, the replacement policies used are conventional, such as: Least Recently Used (LRU), Least Frequently Used (LFU), Adaptive Replacement Cache (ARC), etc. On the other hand, we note here that only 20% of queries are solicited about 80% of the time, so in our system, cache users retain only the queries and its corresponding ARs. In our research, we propose a combined replacement policy, called Multiple Layers Association Rules (MLAR), in order to manage the cache. We are working on the issue of caching ARs for users, data consistency, and the fact to maintain continuous data for all users to provide more reliable reports.

We chose the chunk technique to store the query results in the cache, because even the most powerful systems cannot replicate the contents of all the selected tables from data sources. On the other hand, the use of sub-tables for the cache may be an effective alternative by caching only the interesting parts of data sources' tables. For all these reasons, we introduced the use of chunk based cache strategy. In the schema of chunk based cache, the query results to be stored in the cache are broken into pieces and the pieces are cached. When a new application is requested, the pieces needed to respond to this request are determined. The combination of all these techniques with three-level architecture to enjoy all its benefits, will allow building an OLAP system fairly reliable and consistent.

The steps of our methodology can be summarized as follows:

- Generation of ARs in DW and keep only the interesting rules by eliminating the non-large items.
- Apply an intelligent algorithm to cache (of the user and/or Server OLAP) ARs highly requested.
- Cache ARs and their associated results.
- Applying a replacement policy by using a combined threshold indicators (time, Access Frequency, and positive gain of rules) to keep in the cache the results and rules which are the high updated

3.2 Database schema to storing the AR

We generate the ARs and stored them in the DW with multidimensional data according to schema described in table 2. The mapping the ARs to queries is discussed in the literature in several woks [7, 11]. In the remainder of this section, we present the storage process of two following rules:

Rule 1: (Drug_Description = 'Doliprane')
→ Profit = "Low"

Rule 2: (Drug_Description = 'Flagyl', Year = '1995')
→ Profit = "Low"

Here, the attribute Rule_ID is a unique identifier assigned to each rule of the association. Rule_Type identifies whether it is antecedent (ANT) or consequent (CONS). Attr_Type distinguish a dimension of a measure (D/M). Attr_ID / Attr_Level / Attr_Value respectively store the ID attribute in the dimension / level of the attribute / value of the attribute.Attr_ID. For example, the attribute of ID 101 is a drug with name is 'Doliprane' having level "3" and present a low profit.

3.3 Multi-layers caching mechanism for users

Initially, a user has several ARs available in his cache, and after selecting a rule looking interesting to him, he can know the data represented by this AR. This request is transmitted to the middle-tier server as an expression of OLAP query and query result is returned to the user's computer. Our caching mechanism stores the multi-layer ARs and details of their queries in relation. These layers of caching correspond to three different types of information that are interrelated: (i) the ARs generated, (ii) the terms of the corresponding multidimensional query and (iii) the results of the query. Accordingly, the three layers in our caching mechanism correspond to the ARs, query expressions and the result set, as shown in (Figure 4). We cache the ARs that are viewed by users in Layer-1. Layer-2 is responsible for caching the semantics of the OLAP query (that is, the query expressions) corresponding to the current rule.

In layer 3, we store the query results corresponding to the OLAP query expressions. It is important to note that the caching mechanism in multi-layer and three-tier architecture of our system layer-1, layer-2 and layer-3 are located on the middle-tier server and computer's users.

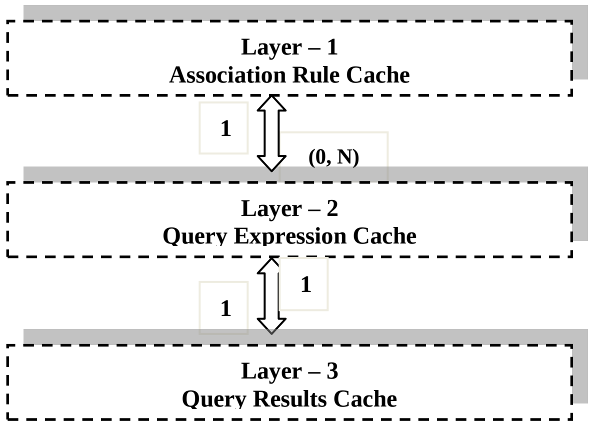

Fig 4: Multi-layers cache

Table 2 Table storing the ARs

Rule_ ID	Rule_ Type	Attr_ Type	Attr_ ID	Attr_ Lev	Attr_Value	Support	Confidence	Time_ Stamp	Access_ Frequency	Lift
1	ANT	D	101	3	Doliprane	40.5	26.6	11/06/2009	5	0.6
1	CONS	M	201	0	Low	40.5	26.6	11/06/2009	5	0.6
2	ANT	D	101	3	Flagyl	34.1	11.75	14/06/2010	4	1.7
2	ANT	D	401	1	1995	34.1	11.75	14/06/2010	4	1.7
2	CONS	M	201	0	Low	34.1	11.75	14/06/2010	4	1.7

3.3.1 Method for caching the association rules: Layer 1

When a user searches for an association rule rcurr, the monitoring mechanism is to inquire whether the rule is in Layer-1 of the user. If the rule rcurr is found, it will be presented to the user associated with the results; otherwise, the request is transferred to the intermediate server layer (Layer-2). The server searches rcurr in its own cache at the layer-2 and transfers it to the cache of the user (layer-1) if rcurr is found; otherwise the request reaches the DW, which will produce the rule and send it to the server middle tier, which then sends it to the cache of the user to use it. Before transferring the rule on the user's computer, the intermediate server caches rcurr in its cache. If space is insufficient in its cache, the appropriate rules are deleted in accordance with our replacement policy and rcurr is sent in the cache.

Each cached rule is associated with some useful information such as: the date of creation, lift of requested ARs and the frequency of requests, in order to be used in the replacement algorithm. Similar to the server, the same events occur in the layer-1 at the user's computer cache.

3.3.2 Caching semantics of the query and its results: Layers 2 and 3

Since the layer-2 and layer-3 are mutually dependent, we consider them together in our discussion. When a user requests data corresponding to a multidimensional query Q (rcurr), it is implicit that the user has already selected the corresponding AR, caching at the time the observation, so the rule_id that is sent to the server identifies reasonably the general rule, while minimizing the amount of information to be transmitted.

When a user requests an OLAP query Q (rcurr), the layer-2 is searched first. If it is found, the corresponding query results are retrieved from the layer-3 and presented to the user; otherwise, the demand shifts on the server or even the DW and Q (rcurr) are sent to the server, then the user's computer that can invoke replacement policy. It is important to note that our caching mechanism at various levels identifies the characteristics of the user computer and middle-tier server: On the computer of the user, the cached data represent only the requests of this user, while on the server; they represent the overall demand for many users.

3.4 Replacement Policy

Our replacement policy is used when there is not enough space in the cache to hold the data requested. To store an AR in the layer-1 where the free space is insufficient, we must first seek the candidate rule to be deleted. This is based on three parameters: date of creation of the AR (TIME_STAMP), the frequency of access to this rule (Frequency_Access) and the lift of an AR existing in the cache (LIFT). The principle of operation of the first parameter, date of creation of the AR (TIME_STAMP), is to keep cached the most recent rules. As for the second parameter, frequency of access to this rule (Frequency_Access), it is to remove from the cache the less frequently used rules. Finally, the third parameter, the lift of an AR, we keep the ARs where the antecedents and consequents are positively dependent. Having identified and removed the Victim(s) Association Rule(s) (VAR), which exist in the cache according to the used policy, to obtain the required space in memory cache, the new rule becomes a member of the cache.

4. Performance evaluation

The purpose of this section is to present the results of performance analysis of the proposed replacement algorithm (MLAR) through a comparison with the results of algorithms Adaptive Replacement Cache (ARC) proposed in the literature. This section is organized as follows: firstly, we describe the experimental platform and data set. Secondly, we describe and discuss the results of performance evaluation.

4.1 Experimental platform

To simulate our proposed algorithm, we implemented our own application using the Visual Basic.NET programming language with a database in SQL Server 2005. In order to evaluate the performance of the proposed methodology, we perform a numerical comparison based on the example described in the section 2.1. We consider two platforms:

Platform 1	Platform 2
Server Cache Size = 600	Server Cache Size = 300
Client Cache Size = 500	Client Cache Size = 200
Access Frequency Threshold = 3	Access Frequency Threshold = 3
Time Threshold = 30	Time Threshold = 20

4.2 Results performance evaluation

In this section, we perform a numerical comparison of our methodology. We are interested in our comparison to analyze some criteria keys such as: Request Search time of the query and Replacement policy.

4.2.1 Time analysis of the search query

In this section we present and compare the searched query time of the proposed method and the convention method proposed in the literature. We realize two evaluations under two different platforms: platform 1 and platform 2.

4.2.1.1 First evaluation

This evaluation is realized under platform 1. The following figure (Figure 5) contains a comparison of the performed results of our proposed method. The results obtained by our method are encouraging compared to those obtained in the case where there is no cache. We note that the search time of the query was reduced after the implementation of our method with the cache. The obtained results are presented in the following table showing the response time (RT) associated to each case of query.

4.2.1.2 Second Evaluation

This evaluation is realized under platform 2. It brought an example of 'Chunk Based Cache'. The following table shows that the technique "Chunk Based Cache" has improved the response time of queries. This example is presented in the last three lines. The dimension *Region = 'Haute-Garonne'* in our DW consists of two Cities '*Toulouse*' and '*Auch*'.

We supplied the details of the City 'Toulouse' first (Q 5.a) and then the information from the Region 'Haute Garonne'. With the use of technology of 'Chunk Based Cache', the query (Q 5.b) is half the cache of the client. It consists of gathering the information related to the city of 'Auch' which represents the remainder part relates to information concerning the region of 'Haute-Garonne'. We note the effectiveness of this technique (Figure 6). On the other hand, we note that by decreasing the size of the cache, the response time of the query increases because of the replacement algorithm's call. The obtained results are presented in the following figure (Figure 6).

4.2.2 Analysis of the two replacement policies: « MLAR» and « ARC »

The following figure (Figure 7) contains a comparison of the performance results of the two policies "MLAR" AND "ARC". The results show that our policy "MLAR" has reduced the replacement time if there is a need for replacement of a single query. When there is a need to replace several queries, the method "ARC" is more efficient. For example, the query (*Q 6*) requests a replacement of a single query and generates less computing time using MLAR than ARC method. The query (*Q 10*) requests a replacement of 6 queries and generates more computing time using MLAR than ARC method.

4.3 Discussion

Having tested our methodology under different scenarios, we shall evaluate the performance of its methodology. We note the following points: (1) Search time of a query is reduced after caching this query; (2) Reducing the search time of the query by using the technique of chunk based-cache; (3) Increasing the cache size leads to increased the search time of the query; and (4) The replacement time of our algorithm, MLAR is much better than the algorithm ARC, when the number of replaced queries is equal to 1.

Queries	**Description**
Q 1.a	*(Time_Year (1) = 2006) & (Drug_Description (3) = Sirop) & (Branch_City (3) = Lyon)*
Q 1.b	*(Time_Year (1) = 2006) & (Drug_Description (3) = Sirop) & (Branch_City (3) = Lyon)*
Q 2.a	*(Time_Year (1) = 2007) & (Drug_Description (3) = Sirop) & (Branch_Region (2) = Paris)*
Q 2.b	*(Time_Year (1) = 2007) & (Drug_Description (3) = Sirop) & (Branch_Region (2) = Paris)*
Q 3.a	*(Time_Year (1) = 2008) & (Drug_Category (1) = Young) & (Branch_City (3) = Auch)*
Q 3.b	*(Time_Year (1) = 2008) & (Drug_Category (1) = Young) & (Branch_City (3) = Auch)*
Q 4.a	*(Time_Year (1) = 2006) & (Drug_Category (1) = Child) & (Branch_City (3) = Auch)*
Q 4.b	*(Time_Year (1) = 2006) & (Drug_Category (1) = Child) & (Branch_City (3) = Auch)*

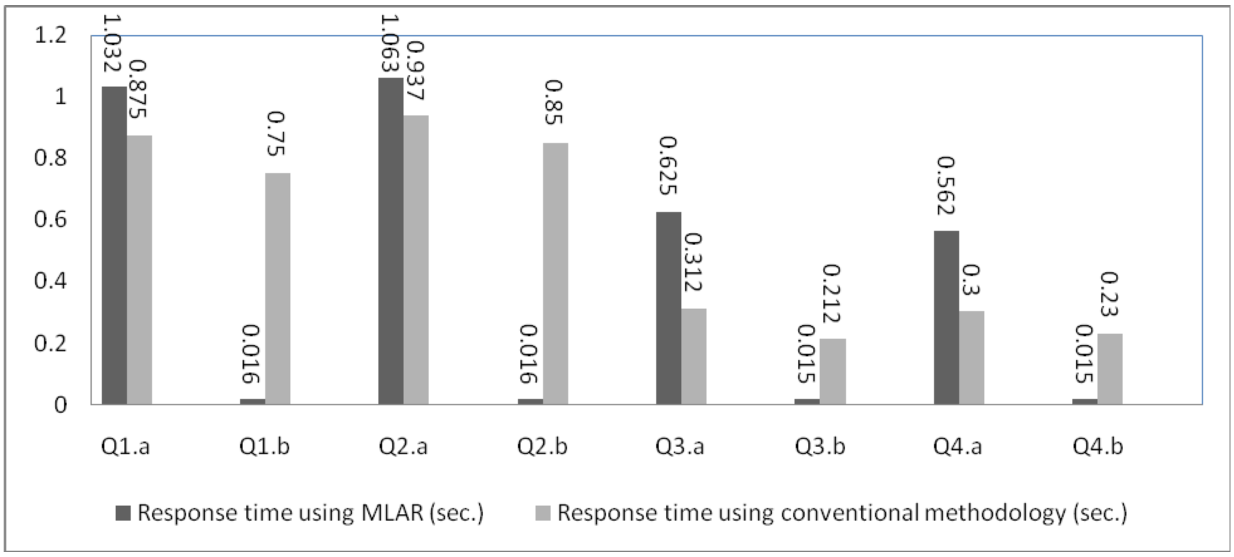

Fig 5: Table of queries and its corresponding response time

5. CONCLUSION AND PERSPECTIVES

The aim of our research was to present for multi-users the ability to discover the knowledge through the use of ARs in order to make tactical decisions. We presented a new technique that uses the three-tier architecture to enable caching the most requested ARs in two different levels: the user and the server. In this work, we proposed a methodology that combines the OLAP technology with the ARs using the technique of chunk based-cache following a Three-Tier Architecture to enable all users enjoy the benefits of data warehousing in the best manner to improve the performance and also to increase the use of the system while reducing the response time. Using three-tier architecture ensures the hierarchical structure of our system to be guided navigation through it.

We have applied a technique of management and replacement of the cache based on criteria such as frequency of access, the date of use of any rules, and the contributions of rules highly positive. This work has helped to achieve the following points: (1) We presented a caching algorithm of ARs most sought in an effective manner; (2) We have demonstrated the effectiveness of our algorithm in reducing the response time through a performance evaluation; (3) We proved algorithms in the literature to give effective results, such as the Apriori algorithm in order to generate the ARs, and the combination of several criteria to better manage the replacement policy and eliminate unwanted chunks; and (4) We used a three-tier architecture to improve system utilization.

To manage the replacement policy, we have proposed that ARs should be thrown out of the two caches based on their frequency of use, date and time of use, and finally the positive contribution of the rule. The first criterion ensures the maintenance of rules most wanted on the fact that only a few queries are executed the most and are most relevant for reporting tactics. The second criterion ensures the consistency of the values contained in the tables over time. The latter criterion ensures the maintenance of rules providing the most interest. Note here that the cache server allows all users to benefit from the ARs in greatest demand. In addition to the ARs and caching, we have tried to enjoy a chunk based technique which is a reliable technique. We have evaluated our application by analyzing different data sets by changing the cache sizes to ensure the reliability of our approach by comparing its optimal resolution with the conventional technique. This comparison highlighted the benefits of our work, demonstrating the decrease in response time. In addition, the evaluation showed that with the use of our new system, it can provide reliable and updated data.

We have deduced that the use of such an integrated system provides a framework that can handle many problems in the area of OLAP systems while providing a reduction in response time. Nevertheless, our technique is based on using several other techniques that we could not test all their effectiveness because of the complexity of their implementation. We tried to simulate their work which was obtained from such assessments.

Query	Description
Q 1.a	*(Time_Year (1) = 2006) & (Drug_Description (3) = Sirop) & (Branch_City (3) = Lyon)*
Q 1.b	*(Time_Year (1) = 2006) & (Drug_Description (3) = Sirop) & (Branch_City (3) = Lyon)*
Q 2.a	*(Time_Year (1) = 2007) & (Drug_Description (3) = Sirop) & (Branch_ Region (2)= Paris)*
Q 2.b	*(Time_Year (1) = 2007) & (Drug_Description (3) = Sirop) & (Branch_ Region (2) = Paris)*
Q 3.a	*(Time_Year (1) = 2008) & (Drug_Category(1) = Young) & (Branch_ City (3) = Toulouse)*
Q 3.b	*(Time_Year (1) = 2008) & (Drug_Category(1) = Young) & (Branch_ City (3) = Toulouse)*
Q 4.a	*(Time_Year (1) = 2006) & (Drug_Category(1) = Child) & (Branch_ City (3) = Toulouse)*
Q 4.b	*(Time_Year (1) = 2006) & (Drug_Category(1) = Child) & (Branch_ City (3) = Toulouse)*
Q 5.a	*(Time_Year (1) = 2006) & (Drug_Description (3) = Doliprane) & (Branch_ City (3) = Toulouse)*
Q 5.b	*(Time_Year (1) = 2006) & (Drug_Description (3) = Doliprane) & (Branch_ Region (2) = Haute Garonne)*
Q 5.c	*(Time_Year (1) = 2006) & (Drug_Description (3) = Doliprane) & (Branch_ Region (2)= Haute Garonne)*

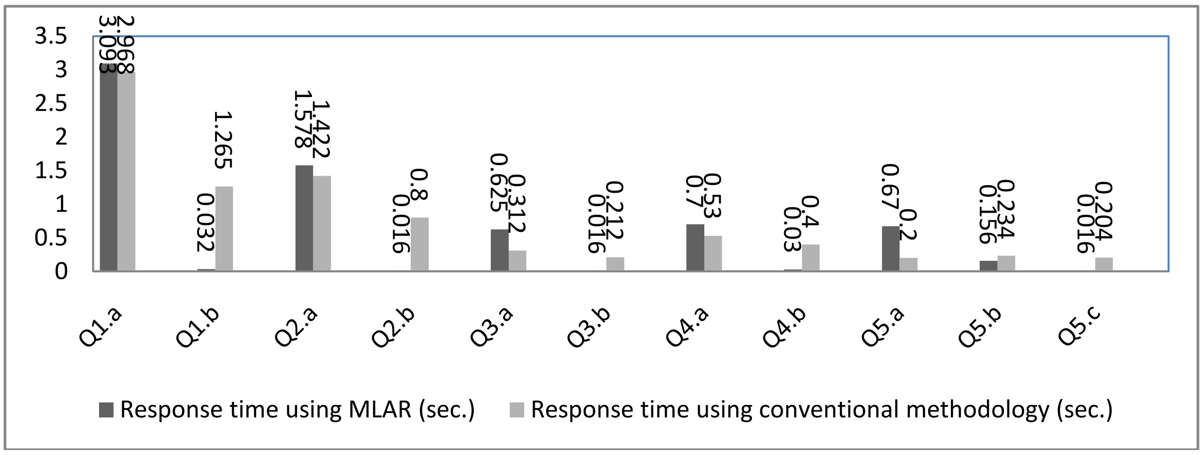

Fig 6: Example of queries and its corresponding response time using the technique of chunk caching

Query	Description	# of Repalcement
Q 6	*(Drug_Description = 'Doliprane', Time_Year = '2006', Profit = 'High')*	*1*
Q 7	*(Time_Year = '2010', Profit = 'Low')*	*1*
Q 8	*(Time_Year = '2007', Profit = 'Low')*	*1*
Q 9	*(Drug_Description = 'Flagyl', Profit = 'Low')*	*1*
Q 10	*(Drug_Description = 'Colchicine', Profit = 'Low')*	*6*
Q 11	*(Drug_ Description = 'Doliprane', Time_Year = '2004', Profit = 'High')*	*3*
Q 12	*(Time_Year = '2010', Profit ='Medium')*	*3*

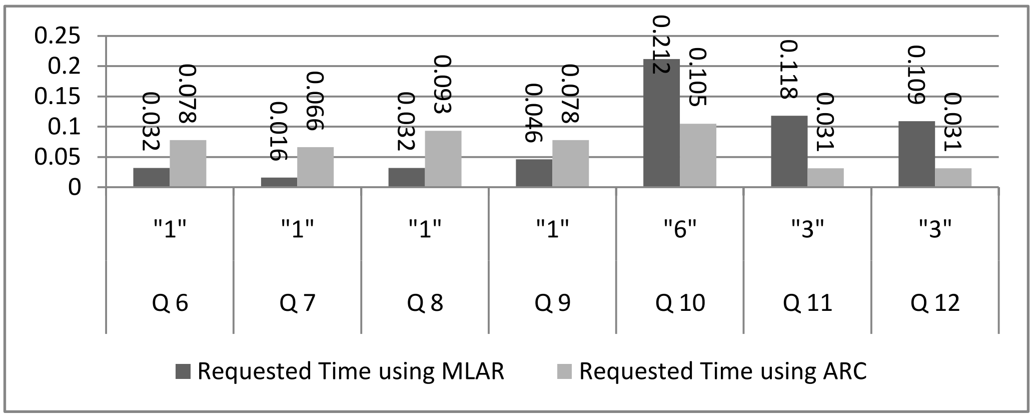

Fig 7: Comparison of the MLAR and ARC using the replacement algorithm

In addition, we could not test the effectiveness of our system in terms of the user over a distributed network with true multi-levels architecture. We can try to improve our work and minimize its limitations. First, we can consider using other methods of DM such as clustering and classification rather than ARs. Another limitation of our study is that there is always the possibility of having a multitude of incoming data that may exceed the capacity of the cache. We confronted these cases when we tried to use the standard "ALL" for aggregation in two dimensions. It would take too long to transmit large amounts of data in the cache table, so we chose to ignore these bad scenarios.

In perspective, it is expected to complete testing and full integration of our methodology on an integrated network, using all resources to be able to test its effectiveness in all circumstances. Secondly, we propose to integrate our proposed technique to cover any business. In particular, our work can be extended to cover the proxy cache because it requires no extension of the HTML protocol or changes to servers, in order to reduce the response time for Web users.

6. REFERENCES

[1] Agraval, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules. Proc. Of International Conference on Very Large Databases, pp.487-499.

[2] Awad, M. and Latifur, K. 2009. Design and Implementation of data mining tools.

[3] Deshpande, P., Ramasamy, K., Shukla, A., and Naughton, J. 1998. Caching Multidimensional Queries using chunks, SIGMOD Conference.

[4] Fangling, L., Yubin, B., Ge, Y. , Daling, W., and Yuntao, L. 2006. An Efficient Indexing Technique for Computing High Dimensional Data Cubes. Lecture Notes in Computer Science, Springer Berlin, Volume 4016, pp. 557-568.

[5] Hang, K. and Kopriva, 2006. Kernel Based Algorithms For Mining Huge Data Sets, Springer-Verlag.

[6] Keller, A. M., and Basu, J. 1996. A predicate-based caching scheme for client-server database architectures. VLDB Journal, 5(1), pp.35-47.

[7] Kumar, N., Gangopadhyay, A., and Karabatis, G. 2007. Supporting mobile decision making with association rules and multi layered caching. Decision Support Systems, Vol. 43, Issue 1, pp. 16-30.

[8] Inmon, W.H. and Kelly, C. 1993. Developping the Data Warehouse. QED Publishing Group, Boston.

[9] Inmon, W.H. 1997. Building the Data Warehouse. Second Edition, John Wiley and Sons.

[10] Imhoff, C., Galemmo, N., and Geiger, J.G. 2003. Mastering Data Warehouse Design: Relational and Dimensional Techniques. Published by Wiley Publishing, Inc., Indianapolis, Indiana.

[11] Meo, R. and Ceri, S. 1996. A new SQL-like operator for mining association rules. Proc. of 22th International Conf. on Very Large Data Bases, September 3-6, 1996, India.

[12] Pyle, D. and Kaufmann, M. 2003. Business Modeling and Data Mining.

[13] Scheuermann, P., Shim, J., and Vingralek, R. 1996. WATCHMAN: A Data Warehouse Intelligent Cache Manager, Proceedings of the VLDB.

[14] Seshadri, S., Cooper, B.F., and Liu, L. 2005. CubeCache: Efficient and Scalable Processing of OLAP Aggregation Queries in a Peer-to-Peer Network, Proc. of IEEE INFOCOM.

[15] Vercellis, C. 2009. Business Intelligence: Data Mining and Optimization for Decision Making.

[16] Widom, J. 1995. Research Problems in Data Warehouse. Proc. Of the fourth International Conference on Information and knowledge Management, Baltimore, Maryland, pp.25-30.

[17] Witten, I.W. and Eibe F. 2005. Data mining: Practical machine learning tools and techniques.

[18] Ying, F. 2004. Range CUBE: Efficient Cube Computation by Exploiting Data Correlation. Proc. of the 20th ICDE Conference.

[19] Zhao, Y., Deshpande, P., and Naughton, J. 1997. An-array based algorithm for simultaneous Multidimensional aggregates, Proceedins ACM SIGMOD Intl. Conf. on management of Data, p.159-170.