# Giving Predictive Abilities to OLAP Systems' Caches

Pedro Marques and Orlando Belo
ALGORITI R&D Centre
Department of Informatics, School of Engineering, University of Minho
PORTUGAL
pcmarkes@gmail.com, obelo@di.uminho.pt

*Abstract* — **It is not new that on-line analytical processing systems arose to companies to stay. They have the ability to change the most common application scenarios that decision-makers use on their everyday tasks. The large flexibility in data exploration and high performance response levels to queries these systems have make them very useful tools for exploring multidimensional data accordingly to the most diverse analysis perspectives of decision-makers. However, despite all the computational resources and techniques we have today, sometimes, it is very hard to maintain such levels of performance for all application scenarios, analytical systems, or user demands. When context conditions and application requirements change, performance losses may occur. There are a lot of strategies, techniques and mechanism that were designed and developed to avoid (or at least to attenuate) such undesirable low performance situations with the purpose to reduce especially data servers load. On-line analytical processing systems caching is one of them, designed for maintaining previous queries and serving them upon subsequent requests without having to ask the server repeatedly. In this paper, we present an on-line analytical processing systems caching technique with the ability to identify the exploration patterns of its users, i.e., what queries a user will submit during a working session, their frequency and resources involved, and to predict what data they will request in a near future, as well as the sequence of those requests. To do that in an efficient manner, we need to maintain a positive ratio between the time spent to predict and materialize the most relevant views to users, and the time that would be spent if no prediction had been done. Using association rules and Markov chains techniques, we designed a flexible manner to provide an effective caching system for on-line analytical processing systems.**

***Keywords – on-line analytical processing; analytical servers; caching; association rules mining; Markov chains; cache content prediction.***

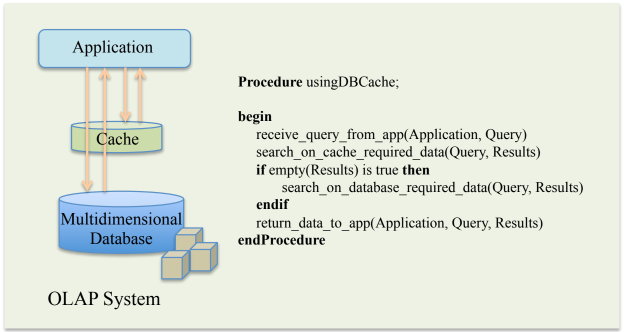
## I. INTRODUCTION

Due to the amazing increase of companies' data repositories in the last decade, attentions turned to the implementation of more powerful ways of analyzing data. As a consequence, Decision Support Systems, and more specifically, *On-line Analytical Processing* (OLAP) systems [1][2][3][4] are being implemented in a large scale, when compared to what was being done a few years ago. A little everywhere, OLAP and data mining systems have captured the attention of many research teams and creators of large software systems. OLAP systems provide sophisticated mechanisms for the analysis of large volumes of data in a very expeditious way, accordingly to the several exploration perspectives of decision makers. Based on this type of systems, decision-making processes are much more oriented and more effective, being supported by well-structured analytical information and not, as so may times happen, by the simple intuition of a decision-maker supported by a package of statistical data. OLAP mechanisms for data exploration and analysis allow for data to be related in a non-trivial manner, making possible to change the current perspectives of analysis whenever necessary. Thus, they are quite flexible. This is only possible due to the fact that data is stored in very specific structures that were especially designed for this type of analytical processes, faithfully following the multidimensional nature of the data as well as its most regular exploration processes.

The high efficiency of OLAP systems for exploring multidimensional data is based primarily on the pre-materialisation of the data that we believe to be necessary to meet the needs (and sometimes the expectations) of a decision-maker. This pre-materialization process is done recurring to the use of materialized views that potentially allow for satisfying "immediately" any question (query) that is launched to the system by its users. As this ideal case of querying satisfaction is practically impossible to achieve, due in part to the limitations of memory and processing capacity of the systems, several techniques have been developed to improve how these views can be materialized *a priori*. Caching techniques are one of them. For a long time, they were applied in Web systems with very positive results. Since the beginning of its implementation, caching techniques were seen as a way to accelerate the process for responding requests (queries) posted by users. The implementation of a caching system in a Web platform aims mainly to maintain the information that has static properties, in order to provide it through a cache (and not through the primary data source) to users that request it repeatedly. This gives us two important advantages. First of all, it allows for a great reduction in responses *time-to-user* – a caching server usually is "closer" to the user than the main information source. On the other hand, this avoids repetitive accesses to the main server, freeing it to answer requests that only it can compute.

In OLAP systems, the information to keep data in cache is very dynamic, but it is not updated very often (perhaps only once in each refreshing cycle of their data structures) and when it happens that is usually done in an incremental manner. The challenges posed by all these constraints are not easy to solve, and this is where caching can help. It is already considered as one of the key factors that contributes to a significant part of the improvement in performance of any OLAP system. As an OLAP system evolves, so must the caching system associated to it – Figure 1 illustrates a simple view of a basic caching system for a multidimensional database.

As in Web systems, caching systems were firstly centred on the individual perspective of the user that was the only one to benefit from its own caching architecture (client-side caching). Businesses were quick to realize that this somewhat less-sharing way of caching was not the best approach. Later, the development of caching systems followed the strand of sharing caches between users, whether they were available on the server side or on the customer side.



```
Procedure usingDBCache;

begin
    receive_query_from_app(Application, Query)
    search_on_cache_required_data(Query, Results)
    if empty(Results) is true then
        search_on_database_required_data(Query, Results)
    endif
    return_data_to_app(Application, Query, Results)
endProcedure
```

Figure 1. Illustration of a basic caching system for an OLAP system.

As we know, one of the great advantages of OLAP systems is the fact that they can cope with large volumes of data, and execute *ad-hoc* queries within various analysis perspectives giving to decision makers an exceptional way to get more structured insights about company's data. OLAP systems were so well accepted by decision makers that they soon started loading more and more data into them and issuing more complex queries, which quickly surfaced some critical performance issues. As fast as an OLAP Server could be, there is always some space to apply new optimization strategies, trying to improve OLAP servers' performance and OLAP users' satisfaction. Thus, the usage of caching mechanisms in OLAP platforms is a natural (and viable) technological choice when one is concerned to improve the quality of service of an OLAP platform.

Despite being widely implemented and tested, conventional caching mechanisms were not prepared to handle OLAP data. One of the reasons why this type of information was not ideal for caching was due to its dynamic nature (i.e., versus the static nature of HTML information where caching techniques have a particularly good fit). Other aspect to be considered when dealing with OLAP data is the dimension of the data to be kept in cache, both in terms of volume of data as well as in terms of data structure complexity. Comparing again with HTML data, which represents a little effort in terms of space needed to keep it in cache, OLAP data requires a great amount of space, simply due to the fact that any response to a typical MDX (*Multidimensional Expression*) query involves a lot of data, usually materialized in a multidimensional data view (a data cube). Even with the diversity of the data to be maintained, several techniques were developed to apply caching mechanisms to OLAP data [5][6][7], revealing benefits good enough to keep the focus on improving caching techniques in order to integrate them effectively on OLAP server systems.

The work we developed was based on an analysis of today's caching mechanisms and their application in the OLAP field, and based on selected information about user's querying patterns [1]. In order to obtain these patterns, OLAP server logs were fetched, analysed and mined in order to obtain a set of association rules that represent the actions (and consequences) of user's queries (usage profiles), providing us the means to predict future user's querying tendencies. Such predictions unlock the possibility of issuing a query even before a user post it, putting it in cache and finally providing it faster than if no cache was available in the OLAP platform.

Basically, having the ability to establish the exploration patterns of a community of OLAP users give us the possibility to define *a priori* the contents of a cache with a satisfactory confidence label. With this, it will be possible to have, in advance, a predefined set of materialize views that correspond to the most frequent multidimensional queries that were done previously during a certain period. Of course that the success of this strategy depends a lot from the tendencies (and routines) of data exploration that users may have. However, in general terms, an OLAP user uses to apply systematically for a particular set of queries as the starting point of his OLAP session. Knowing that, and having also an exploration profile, we can establish the initial state of a cache, adjusting it dynamically during the execution of an OLAP session accordingly the prediction we have made based on past OLAP sessions - more adequate caches' contents in a shorter time. In short, this was the goal of our work.

In the next sections, a more in-depth analysis to this process will be conducted, explaining the various stages reached along the evolution of the work, as well as discussing some of most relevant considerations needed to understand the complexity of predicting the multidimensional content of a cache for a specific OLAP platform. This paper is organized as follows: Section II presents a brief relation about some caching techniques and caching maintenance algorithms; Section III shows a detailed overview about OLAP caching, its advantages and disadvantages; Section IV presents and discusses the major characteristics of problems we could face when using high specialized caches as OLAP caches; Section V reveals our technique for a new model of OLAP caching; Section VI reveals and discusses the results of the tests we have done for validating the proposed OLAP caching model; and finally, Section VII presents final remarks and conclusions, as well as point out some lines for future research.

## II. RELATED WORK

The introduction of caching mechanisms in OLAP systems brings some great advantages. Firstly, because queries are answered directly from the cache system, decreasing accesses to the OLAP server. Secondly, since caches are usually closer to the users, the network traffic between the nodes closer to the OLAP server decreases - providing better network latency and quality of service. As a result, OLAP server availability and performance increase.

As we know, the concern of improving services of a server, and in particular of an OLAP server, is not new. Many researchers have developed effective work in this area, with particular emphasis in areas such as cache management algorithms and caching systems architectures. Usually, algorithms for managing caches are used to decide whether a given piece of data should be placed (or not) in a cache. This allocation decision affects the contents of the cache and consequently its own performance. Thus, by analyzing the consequences of these decisions positively or negatively it is possible to make an evaluation about the quality of the cache management algorithm that was used, applying a set of metrics especially defined for this purpose, namely the *Hit Ratio* or the *Byte Hit Ratio* metrics.

Over the years many proposals presented a large diversity of algorithms for managing caches. On this domain we should reveal the work that led to the emergence of algorithms such as the *First In First Out* (FIFO), the *Least Frequently Used* (LFU) [8], or the *Least Recently Use* (LRU) [9] algorithms. The operating way of these algorithms is very similar. They are regulated through the definition of static criteria that defines the way as data is removed from a cache when the cache gets full. Independently from the different implementations of data structures that exist to support queues (FIFO), determining the frequency of access of each of the elements maintaining in a cache (LFU) or to sustain a specific time label relative to the time at which data was accessed (LRU), data elements are removed from a cache without concerning their utility for the users of the system.

FIFO, LRU and LFU still are today some of the most popular and theoretically important algorithms for caching management as well as the algorithms *Least Recently Used Second-to-Last Request* (LRU-2) [10], an evolution of the LRU algorithm that was developed to be used in database disk buffering. Finally, a brief reference to a last caching management algorithm: ARC (Adaptive Replacement Cache) [11], which has the ability to balance in adaptive manner the workload of a cache in a self-tuning fashion.

All algorithms for managing caches mentioned earlier aim to manage the information that should be added or removed from a cache system. Similarly, with another level of abstraction, we can refer other works in this domain that address many relevant aspects in the implementation of a caching system, namely the problem of the location of the cache. Alternatively to the implementation of a caching system on the client side, we can do it on the server side, as already referred, creating mechanisms that benefit all users of a given community. From the simple to more complex caching systems, this last approach involves, among others, peer-to-peer, active caching, or chunks based caching architectures. Chunks were defined in Deshpande et al. [12] and are a new indivisible unit. This data unit, with a low granularity level, is mapped in the cache in order to be aggregated to satisfy user requests. The mapping occurs in the server and denotes the relationship between a chunk and the basic units stored in the OLAP Server, allowing for the complementary fetching of data from a main data source.

Let us now look at other approaches, starting by the peer-to-peer architecture. In [5] it was said that, a little bit like as all other proposals in the area of distributed caching, if all participants in a network share their personal caches, everyone would benefit, and proposed a network architecture that enables such features – the PeerOLAP. Additionally, they also presented some other policies for renovating and maintaining data in distributed caching system. In [13] and [14] it was approached an active caching system technique. This technique was presented as an effective solution to the problem of creating a caching system for dynamic information manipulated by Web proxies. As we know, performing caching of static HTML pages is a common practice. But the implementation of a caching system for OLAP queries (and correspondent results) in Web proxies is not very usual. This is due to the fact that proxies usually are not prepared to maintain dynamic information, and especially because they do not have the necessary mechanisms to deal with the necessary post-calculations. On the other hand, in [15] there is a proposal to by-pass problems at the level of caching queries, allowing for the use of data in cache to respond partially to a query. The rest of the answer will be obtained directly from the server. This was accomplished by dividing data into chunks stored directly in the cache. When the level of aggregation of a chunk is lower than of the query, chunks could be used to partially calculate the results of the queries presented. Through the use of mapping mechanisms, between the data of the query and corresponding chunk number, it was possible to determine all the chunks needed for the calculation of the solution of a given query.

Later, in [16] were proposed other algorithms to group proxies dynamically in "neighbourhoods", regrouping them whenever necessary according some predefined requisites. This approach can be regarded as second level caching, at which information refers to maintain the best neighbours of a certain proxy. The process is relatively simple. When a proxy discovers that there is another one that is not his neighbour but it can bring greater benefits, it adds it to its list of neighbours, eliminating any other proxy in that list that is less beneficial, if necessary – the complexity associated to this process relies mostly in maintaining accurate statistics about the other proxies behaviour and performance.

In OLAP, selecting views to materialize is an NP-complete problem [17]. Some of well known approaches - e.g., [18], [19] or [20] - propose this selection to be performed statically before each set of queries, being results used to respond to subsequent queries. To avoid the problem that arises with the fact that the usage patterns are dynamic, in [21], [22] and [7] it was developed some other techniques to exploit this kind of situations. An evolution of the proposal of the system Dynamat [21] was the implementation of a mechanism regarding the usage patterns of users as well as its dynamic structure [23]. In the same line of research we find the system PROMISE [22], which has the ability to predict in a more accurate manner what was the structure of a query based on some previous usage patterns, as well as the current query issued.

Any proposal that intends to respond to the fundamental problems of a dynamic selection of views – e.g., what is the amount of information that is required to draw a good user profile, and what is the right time to bring such views to memory (only when requested or trying to predict what is the next view to be required), can be found in [23]. As we can see a lot of proposals to design and implement caching systems were done during the last decade. Here, we just enumerated some of them, trying to enhance some important issues about some techniques to put and manage data in a cache. All these issues can be exploited and adjusted for the implementation of specific caches for OLAP systems.

## III. To Cache or not to Cache

To cache or not to cache is not a simple decision. The implementation of a caching system in an inappropriate time entails additional costs and does not bring any benefits to the entire system. There are many aspects that must be considered before deciding on the implementation of a caching system. Many of these aspects are related, directly or not, with the existence of a performance problem. To detect or prove it, we can use, for example, profiling or logging techniques that reveal us how the system is being exploited and respond to the information requests. With this, we can find what the information that is most often used is and make sure the system presents it expeditiously. If the system is unable, for performance reasons, to quickly deliver this information, we can improve the system's performance by placing this information in a caching system, which will reduce the number of disk accesses, decrease querying processing time and, consequently, decrease the overall time to get querying results. Additionally, through caching, one achieves the basis to have a more scalable and flexible system, with high service availability and better performance.

Usually, a database system can make three types of caching, namely results, execution plans and data objects. Although they are all important, in our case, we only approached the caching of results, by studying the application of some profiling techniques to querying processing of a given OLAP system. However, whatever the specific area of implementation could be, when implementing caching mechanisms one has to remember that the space available for storing the cache is not unlimited. As a direct consequence we need to choose (and evaluate) what data should be kept (or not) in a cache and what data should be removed giving space for new (and hopefully more relevant) data to the users' needs. Keeping this in mind, researchers started to test quite well known algorithms – frequently referred as cache management algorithms – that up to that time had only been used in other types of environments such as for caching HTML pages with great success. As results became known, there was a clear notion that there should be promoted some additional efforts to develop new breads of algorithms that focused OLAP scenarios in particular.

A caching management system is a crucial element in the overall performance of a caching system. Basically, its main function is to decide on which information must be maintained (or removed) from a cache in order to allow the addition of new data when necessary. With the aim of measuring the performance of such a system, there are two basic metrics, the *Hit Ratio* and the *Byte Hit Ratio*. The *Hit Ratio* is one of the most common ways of evaluating the value of any caching algorithm. This metric is the ratio between the number of requests that were in cache and the total number of requests that were made, and can be calculated using the following expression:

$$Hit\ Ratio \leftarrow RequestsSatisfiedByCache\ /\ TotalRequests$$

However, *Hit Ratio* is not a perfect metric. For instance, even with a higher *Hit Ratio*, the number of bytes served directly by the cache could be smaller than a cache with a lower *Hit Ratio*, which led to the creation of another metric: the *Byte Hit Ratio*. This last metric has been vastly used to evaluate how a cache can satisfy its clients' requests. Contrary to the previous metric, this one is intended to take account not only how many requests were satisfied from the cache, but also how much information was served this way. If the scenario is caching small pieces of data, it is natural that the percentage of requests answered directly from it is high, but it is also possible that the number of bytes of information satisfied in this way can be low. Conversely, it is possible that a small portion of large pieces of data results in a reverse scenario. The *Byte Hit Ratio* metric is defined according to the following expression:

$$\textit{Byte Hit Ratio} \leftarrow \textit{BytesSatisfiedByCache} \,/\, \textit{TotalBytes}$$

As a user of any OLAP (or other) system launches his queries, the cache management algorithm has to check if the necessary information is stored in the cache or. If not, it needs to decide whether it should or should not be added to the cache. If the request cannot be satisfied directly from the cache, there are two possible outcomes:

1) the cache still has space to accommodate the new data, and so it is added without further due, or
2) the cache does not have enough space to store the new data.

In the former case, the content is added, and after that time, when it is requested, it will be served from cache instead of being satisfied directly by the OLAP Server. If there is no space available in the cache management system, the algorithm can either discard this information or free some space in cache in order to add this new data. This is the main decision that cache algorithms have to make. As we know, this decision will affect the way a cache behaves in the presence of new information to be added. One of the most basic ways to do this selection is to use a FIFO approach, which means that the oldest record to have been added to cache will be removed in order to create space for a new entry. If this is not enough, the second (the third, and so on) oldest records will be removed as necessary, record by record. The main problem with this technique is the fact that it does not consider the nature of the data. Despite of its size or actuality, data has an intrinsic value that cannot be measured as simplistically as these approaches propose. Other (more sophisticated) decision metrics were developed using a timestamp of the last access to a specific piece of data [9], the frequency of access to the data [8], or other more complex metadata such as the ones used by the Greedy Dual algorithm [24], for instance. All these metrics, in one way or another, take into account the intrinsic value of data and the relevance each piece of data has to the users and, therefore, they are much more suited to do the (caching) job correctly than others that simply look at the characteristics of the data neglecting its nature and its relevance to users.

## IV. OLAP CACHING

Some of the most common operations performed when querying an OLAP server are the well-known drill-down and roll-up operations. The first of these two operations consists of lowering the grain at which the data is being analysed. For instance, we can go down in a hierarchy, detailing systematically, level by level, the grain of the data, from a country-level view to a district-level one, for instance. The roll-up operation is its direct counterpart, allowing viewing data at a higher level following as well a determined hierarchy.

In an OLAP server, the data is stored at the lowest level of granularity and then aggregated to a level required by a specific multidimensional request. In [5] a solution was proposed where this characteristic is explored, mainly by sharing the cache over several cache servers, specifically *OLAP Cache Servers* (OCS). In this approach, each OCS has the capability to apply transformations (aggregations and other operations) to multidimensional structures, and thus combine them to satisfy at least part of a request that has been launched by a user. This way, whenever a user issues a query, the various OCS are asked if they have the needed information and, even if they do not, they are asked again if they can compute it from the data they have at a lower grain than the user requested. This means that an OCS can satisfy not only requests that have been issued before (and cached) but also other issues that involve computations over the data that exists in the OCS.

When configuring an OCS is important to indicate what is the granularity of the data that you want to maintain in a cache, as this will define the type of applications that can satisfy a specific peer. Physically, the data is stored in secondary memory and only brought into primary memory when required or, more accurately, when the fetching algorithms decide the most appropriated time to do that. This operation does not have to necessarily be on-demand and can follow, for instance, some kind of predictive approach [22] or any another technique for fetching data. Another alternative was proposed in [6], where individual caches of users are shared through a peer-to-peer network created between users of a same OLAP System – PeerOLAP. Essentially, this approach was based on the Piazza System [25], and intended to allow a very high level of autonomy in the cache network due to the dynamic nature of Peer-to-Peer networks, where users can connect and disconnect without significantly affecting the overall usability and performance of the system.

As in other proposals following a decentralized architecture, a challenge that this system often faces is the need to establish a mechanism to avoid the uncontrolled spread of messages, which can, as we know, create congestion in the network, deteriorating the overall system performance. A possible solution to overcome this is the definition of a maximum number of "jumps" that a message could give before lose their validity. This is quite intuitive. A message after a given number of “jumps”, even if it can find a peer that has an adequate response, will be hardly provided in the best possible time. Another problem that arises here is when a message is being resent to other users, even before reaching the maximum number of jumps, and the only place where this can be resubmitted it is the data warehouse itself. In this case, the message is not relayed to the central peer such as this could lead to the repetition of messages sent to the data warehouse, which breaks any kind of goal of a caching system.

As mentioned before, OLAP data is quite dynamic by nature, which means that it is very difficult to predict when the cached data will become out-dated. To deal with this problem, an active caching technique was created [25]. It consists of keeping in the cache server a Java applet that is invoked every time a cache hit occurs. This applet has the role to check with the OLAP Server if the cache information stills valid or if it has changed since the last time it was requested by one or more users. If data stills valid, it will be returned to the user who requested it. If not, the full request will be redirected to the OLAP Server.

One other question that was frequently placed by researchers, was focused on what would be the optimal level of granularity to store data in a cache, in order to not only be able to aggregate it as needed, but also to be able to do that in a timely fashion manner. On such units is the previously presented chunk, defined by Deshpande et al. [12]. When a cache server receives a request from a user, it calculates the parts of that request that it can be satisfied accessing directly the cache, and the information that it need to be requested to the OLAP Server (at a low level of granularity). When all the required data is located in the cache server, it combines it and sends the results to the user, without him ever knowing if the information came from the central server or the cache server. As a last reference we selected the work presented by Sapia [22], which is an approach particularly interesting to us. In that work, the author proposed a predictive system for user behaviour in multidimensional information system environments that explore characteristic patterns users use to show when explore multidimensional data structures. It is an OLAP caching approach that complements other techniques, such as the ones presented in [26] or [15].

Finally, we should say that the maintenance of caches is something that must be included in the routine of any OLAP system administrator. One cannot optimize performance of such a system simply deciding, from one day to another, the implementation of some kind of caching mechanisms as a solution for a current optimization problem. Generally speaking, implementing a caching system by itself cannot solve any optimization problem. Frequently such problems are treated as early as possible, just starting in the querying design phase and evolving their treatment throughout the design chain of a query. In some sense, caches help to solve (or mitigate) such situations. In our case, we were concerned researching some of the most relevant aspects in the maintenance of a caching system for analytical servers, seeing how we could establish a way to "guess" the various forms of data querying that a specific user community practiced. Basically, the idea was only to find a way to characterize their querying patterns, establishing the most used sequences of queries used, and based on that knowledge materialize their results (when possible) in a caching system. And that is what we will explore in the next section.

## V. A New OLAP Caching Approach

By their nature, OLAP systems allow for identifying, for each user or group of users, how they access and explore data cubes. If we take the example of a high-level decision-maker, most likely he will access only information regarding to the sales of a specific store or a particular region, avoiding specific and detailed information relating to the sales of all company's products, for example. Exploring features like this, we can define not only which areas of impact in terms of data analysis a user usually does, but also which specific sequence of searches usually he uses to follow. Thus, it is possible to make predictions about what will be the next query sent to the OLAP server by a given user, simply knowing which of the queries he released in the past when doing some data exploration over a set of data cubes. One way to acquire knowledge about the behaviour of a user passes is, for instance, analyzing the log files of previous sessions of data querying. Through the data stored in these files and with the application of specific domain-oriented data mining techniques, it is possible to extract some useful and accurate knowledge about the user usage profile.

One of the issues related to the use of this type of knowledge is his assertiveness and the advantage that comes from its use. Unlike other caching techniques, this approach does not aim to reduce the workload of a data server but to improve the response time satisfying user requests. This type of technique uses the last query launched by a user, and based on it (and other historical data) tries to infer which will be requested next by the user. If it is possible to carry out this prediction, with a sufficiently high degree of certainty, the query and the answer will be immediately placed in the cache so that when the user presents the query its answer will be already stored in memory and the system only needs to provide the results to the user, almost immediately. Another approach involves not only the last query performed, but also a certain number of queries before that. Thus, keeping information about the sequence of requests (queries) made by users it is possible to make predictions with greater certainty. However, you must also have a larger amount of information related to the past behaviour of a user in order to enable the accomplishment of such predictions.

This work was based on the assumption that OLAP system's users have predictable patterns of data that they use to consult on their regular OLAP sessions. The nature of most OLAP users in a company – decision makers – usually means they are focused in a relatively small subset of the data stored in a data warehouse. The day-to-day activity of a decision maker may begin with an analysis of a pre-defined dashboard or an interactive report, and based on the information gathered from the analysis of the data, he will continue his exploration in a lower level view of the same data – probably appealing to a typical drill-down operation. This shows us that for any given user his behaviour will be repeated during a certain period of time, revealing then a regular usage pattern.

One possible way of extracting these patterns is by analyzing OLAP Server's logs that contain information about what multidimensional queries users had submitted and when they happened. It is also possible to know, for a given user, the sequence of queries he launched between his login and his logout in a specific OLAP session. From the analysis of this kind of information, given a certain period of an OLAP system exploration, another problem arose: how far back in the logs should we go to make sure that the retrieved rules are truly representative of the user's exploration patterns? On one hand, if we analyse the OLAP exploration habits (and tendencies) for a short period of time, we may get rules that represent the most recent patterns and not what the user usually does in the "long run". However, on the other hand, if we analyse a larger period, we may extract rules that represent older OLAP exploration patterns that do not represent what users are doing currently (users may change their exploration habits due to a large variety of reasons, demanding that the algorithm should be able to adapt to such changes).

Taking these constraints into consideration, we began our approach by retrieving the OLAP server's log files, preparing them to be analysed latter by a specific data mining algorithm with the ability to generate a set of association rules that represent the most relevant exploration user patterns – we designate a set of usage patterns by an OLAP profile. From the OLAP server's log files we extract all the MDX queries that were launched during a certain period by a community of users that we want to establish the correspondent data exploration profiles. Each MDX query is fragmented accordingly several dimensions of analysis, such as OLAP session, cube, query, data and time, dimensions, measures, and users. Then, this information is stored in a specific relational data mart that will provide on the next phase the data to data mining association algorithms.

To establish the association rules we used the well-known *Apriori* algorithm [27]. This is one of the most used algorithm for mining frequent item sets, having prove its effectiveness so many times analysing a set of transactions and surfaces the relationships between them, given a minimum value for support and confidence. As it is well known, association rules are usually represented in the format: $A \rightarrow B$ *(sup=α; conf=β)*, where *sup* and *conf* represent, respectively, the support and the confidence values of a rule. From an association rule (and from its support and confidence values) we can retrieve two important things, namely the:

– *support* (*sup*), that represents the ratio between the number of times that a sequence of queries A followed by a sequence of queries B was found in the dataset and the total number of queries in that dataset:

$$sup(A \rightarrow B) = \frac{\#(A\ followed\ by\ B\ in\ the\ dataset)}{\#(queries\ \text{in the dataset})}$$

– *confidence* (*conf*), that represents the number of times a sequence of queries $A$ is followed by a sequence of queries $B$ in the dataset, divided by the number of times a query $A$ (independently of what query followed it) was found in the same dataset:

$$\text{conf}(A \rightarrow B) = \frac{\#(A\ followed\ by\ B\ in\ the\ dataset)}{\#(A\ in\ the\ dataset)}$$

If we take the association rule $A \rightarrow B$ *(sup=0.3; conf=0.8)*, as a working example, we can say that for every time a user issues the query $A$ he will, in 80% of the cases, issue the query $B$ right after that. On the other hand, we can say that for the analysed dataset, a sequence of queries $A$ followed by a sequence of queries $B$ occurred in 30% of all cases. However, the antecedent (A) of such rules does not correspond necessarily to a single event, which means that A can also represent a set of queries. In our scenario, the prediction process will be supported not only by a single query, but also by a sequence of queries that a user triggered from the beginning of its working session. If the association rule is something like $A1, A2, A3 \rightarrow B$, it means that the consequent of the rule ($B$) can be predicted as a consequence of the occurrence of the events A1, A2 and A3, with a given confidence and support. These rules allow for more than a simple prediction, "step by step", which may happen when we want to predict what the next query to be executed is. Thus, it is possible to predict *a priori*, with greater anticipation, which queries will be launched by users until the end of their working sessions, as well as to know the sequence those queries.

It can also happen that the antecedents of the rules are not necessarily sets of queries that were issued, but other types of querying conditions environments, like periods of a day, periods of a week, or even business data like sales or stock information. If we explore all these possibilities, associating them to a specific OLAP environment, it is possible to define some specific rules that indicate the frequency of a particular user performing a query, which predictably will be executed at the same time on a given day or week. For example, we can think in an application scenario where a decision agent every Friday afternoon, before leaving its workplace, check systematically the most relevant management indicators in order to get a last view of the business status of the company. On such scenario and time frame, the indicators he analysed were always the same. Nevertheless, the consequences of the analysis will be clarified by a particular set of query that he will launch to verify a particular business case brought to his attention.

Using this type of prediction system reveals some interesting capabilities that can be very useful improving the performance of an OLAP system. However, it also raises some pertinent questions that should be answered according to the application context of each specific case, namely:

- What is the number of searches of a user that should be taken into account during a prediction process within the same session?
- All rules should be accepted as valid or we need to define some minimum values for support and confidence, from which the materialization of the corresponding views will be rewarded?
- Shall we materialize immediately all queries that we predict will become necessary or only a part of them?

This technique allows us to establish probabilities for the sequence of queries that a user will issue between the beginning and the end of an OLAP session. With this information some actions may be taken to improve the OLAP server's response time to queries. Our approach was to simulate the user's interaction and place in cache the views our algorithm predicted would be used. The main problem with this is the high value of rules that are going to be generated. This could easily produce untreatable results.

Keeping this problem in mind, our work focused on reducing the number of queries that should be included in the prediction phase, without affecting results significantly. To do this, we chose to map all the sequences of queries predicted by the mining algorithm, representing them in a Markov chain [28] as a way to provide a better visual insight of the entire set of generated rules. Next, we defined the minimum value for the confidence associated with the rules that should be used in the prediction phase (*minconf*). Shortly, we discovered that this action would not be enough if we wanted to effectively reduce the number of predicted queries. We needed to optimize the process.

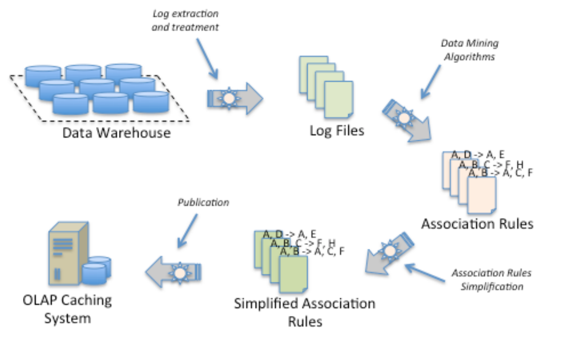


Figure 2. A query sequence prediction for the first dataset.

When removing the rules with a confidence value smaller than *minconf*, we realized that some rules remained without the possibility to be predicted as a sequence of any other query. If we think of the sequence of queries as a graph, and we start removing some of the nodes, there are some of them that lose their entrance arches. Those "nodes" represent the queries that were removed in this second optimization step. This way we also risk an increased number of cache misses, but provide us an alternative way of reducing the number of views to be pre-materialized in the cache. The process followed to establish the set of association rules for a particular *minconf* is depicted in Figure 2.

## VI. Validating The Proposed Technique

In order to test the technique proposed here, we decided to promote two different test cases, considering the number of query hits achieved before and after the proposed optimization scenarios, for a given set of artificial queries (generated by artificial processing algorithms, not representing the actual usage of an OLAP Server). In Figure 3, we can see the sequence of queries in a Markov chain, which were predicted by the mining algorithm that was used – $S_0$ and $S_8$ represent, respectively, the beginning of the session provoked by the user's login and the end of that session. The edges' values represent the transition probabilities between two different states (or queries). Based on the Markov chain presented in Figure 3, we can see that, for example, the query S1 is the first query being made in 40% of the treated cases (this value is the label of the transition $S_0 \rightarrow S_1$) and queries S3 and S2 will be executed then, respectively, in 90% and 10% of all queries executed. The rule that support $S_1 \rightarrow S_3$ could be something like: $S_1 \rightarrow S_3$ *(sup= …; conf=0.9).*

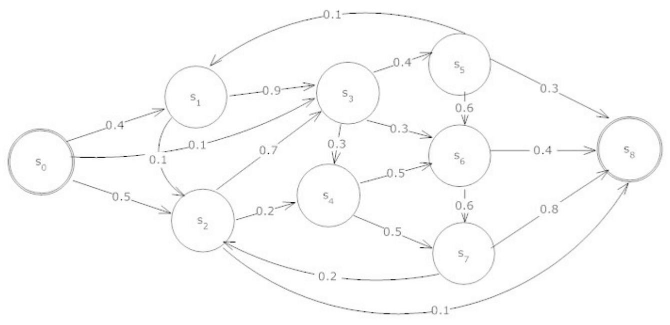


Figure 3. A query sequence prediction for the first dataset.

One way to reduce the number of queries to be materialized prior to the consultation (and cached) of a given user is by observing the probability of occurrence of all queries. Thus, from the start of a new working session, it is possible to identify which set of queries allows for reaching the final state (S8) with a greater probability of success. However, to do this, we have to look in each state of the Markov chain, which is the state most likely to be the next state to be reached until the final state is reached. All the tests conducted over this dataset basically used various values for *minconf* simplifying the rules accordingly. The chosen values for *minconf* were, respectively, 0.3, 0.4, and 0.5 (Table I). One other simplification was introduced, and named as "main route", simplistically put in cache the sequence of queries that a user will most likely follow in a future data exploration process, from login to logout.

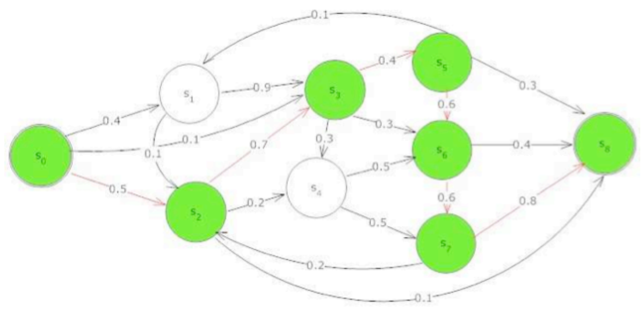


Figure 4. A candidate sequence of queries that an user most likely follow in a future data exploration process.

In Figure 4, we can clearly identify such candidate sequence of queries. It will be the sequence represented by the path:

$S_0 \rightarrow S_2 \rightarrow S_3 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7 \rightarrow S_8$.

It can be easily found by following the higher transition probabilities between the $S_0$ and $S_8$ nodes. In the specific case of the Markov chain presented in Figure 4, the identification of the "main route" will be a result of the materialization of the nodes S2, S3, S5, S6, and S7, since nodes S0 and S8 correspond respectively to the begin and end session actions. The results of the tests, for the different values of *minconf* and for the "main route" simplification models, can be found in Figure 5. All the results of the tests were compared with each other - for a fairer comparison, we present the percentages for the values attained in each test.

TABLE I. TEST RESULTS FOR THE FIRST DATASET

| ***Minconf*** | **0.3** | **0.4** | **0.5** | ***"main route"*** |
|---|---|---|---|---|
| Pre-materialized views (%) | 100 | 86 | 28 | 71 |
| Cache Hits (%) | 100 | 89.8 | 38.3 | 79.78 |

As a comparison value, if we add 50% of all queries to the cache, intuitively we think we would achieve almost 50% cache hits for any given user (Figure 5). However, this value is merely meant to provide us with a reference value, and should not be considered in terms of absolute values. Figure 6 leads us to note two key values of *minconf* values if 0.3 and 0.5, which show the most relevant (best and worst) test results. As for the value 0.5, it means that only 28% of all possible views were pre-materialized and, even in that case, the cache hits came around 38.3%, which represents a 10% increase in system performance when compared to our reference values. The usage of 0.3 for *minconf* resulted in no view being simplified and, consequently, the values of cache hits were measured at 100%. In Figure 7 and Figure 8 we can see the simplified Markov chains that resulted, respectively, from the application of a *minconf* = 0.3, and a *minconf* = 0.5.

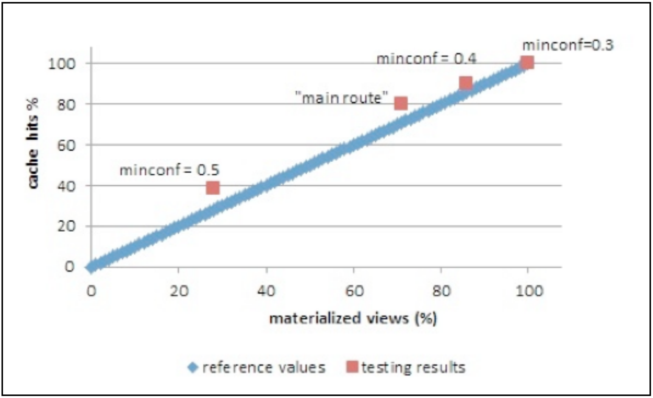
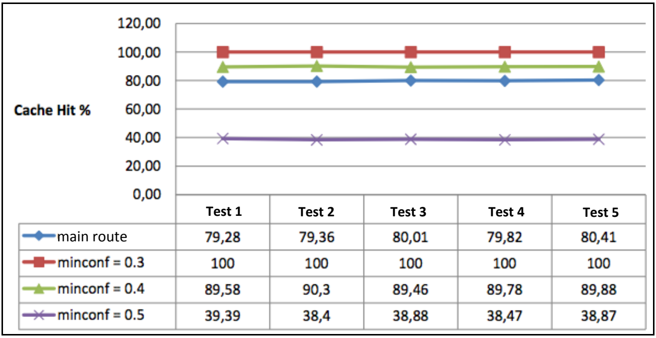


Figure 5. Test results graph for the first dataset.



| | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 |
|---|---|---|---|---|---|
| main route | 79,28 | 79,36 | 80,01 | 79,82 | 80,41 |
| minconf = 0.3 | 100 | 100 | 100 | 100 | 100 |
| minconf = 0.4 | 89,58 | 90,3 | 89,46 | 89,78 | 89,88 |
| minconf = 0.5 | 39,39 | 38,4 | 38,88 | 38,47 | 38,87 |

Figure 6. Comparison of the results of the tests.

With the application of a confidence restriction of 0.3 it is possible to remove arcs that possess a lower probability than the value of confidence defined (Figure 7). Thus, it is possible to see that, for example, the arc corresponding to the transition between nodes S2 and S8 was not accepted. On the other hand, if the strategy is to materialize all views remaining in the Markov chain, then, with this confidence value, it is not possible to remove any of the views present in the chain. Due to this fact, it can be considered that although the definition of a more restrictive value of the minimum confidence (*minconf* = 0.3) the benefit in this case would be non-existent, since the set of views to materialize would be precisely the same.

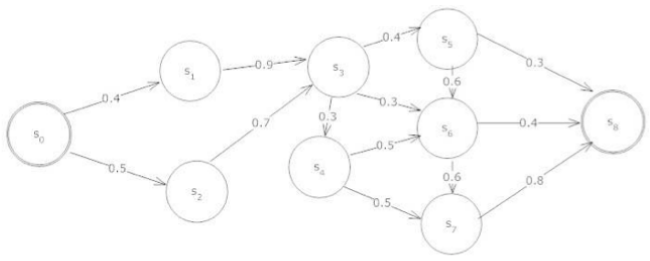


Figure 7. A simplified Markov chain for minconf = 0.3.

In the case presented in Figure 8 it is shown another simplification of the prediction model generated before, but now applying a *minconf* = 0.5. Following the logic previously exposed for this case were removed from the model nodes S1, S4 and S5 since they did not have, after the first step of simplification, any incoming arc.

.

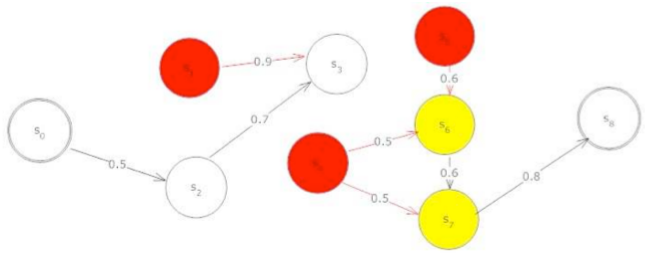


Figure 8. A simplified Markov chain for minconf = 0.5.

In a second stage of analysis, we observed that the node S6, after the removal of node S5, lost the only link it owned that could support the prediction of its occurrence. Because of this, it was necessary to remove as well the node S6. Then, with the same rationale, we removed the node S7.

TABLE II. TEST RESULTS FOR THE SECOND DATASET

| ***Minconf*** | **0.02** | **0.3** | **0.4** | **0.6** |
|---|---|---|---|---|
| Pre-materialized views (%) | 54 | 52 | 50 | 46 |
| Cache Hits (%) | 89 | 88 | 87 | 86 |

As can be seen from Figure 8 the restriction of a *minconf* = 0.5 has resulted in a quite considerable simplification of the prediction model previously generated. Thus, for this case, we will materialize only the views S0 (the initial state), S2, S3 and S8 (the final state). Later, other tests were conducted with another data set retrieved from several OLAP sessions we made on a specific OLAP server. This second dataset contains a total of 59 queries being issued to the server, and the values of *minconf* used to simplify the generated rules were 0.02, 0.03, 0.4, and 0.6. The results of this second experience can be found in Table II and Figure 9. The results obtained in this second round of tests shows us that, even though the differences between the different values of *minconf*, they do not yield great differences in the percentage of cache hits – nor in the percentage of materialized views. The gains relative to the reference values were quite relevant, staying approximately between 35% and 40% (for values of *minconf* equal to 0.02 and 0.6, respectively). Despite the lack of real caching data, all data sets prepared for testing and using in the several application scenarios we conceptualized provided the necessary means to prove the utility of our approach.

However, the results were not a surprise. In fact, based on the experience we have from other studies using association rules, we expected that the most frequent queries, as well as their more frequent sequences, were revealed naturally. This would allow for solid knowledge about analytical usage trends of a user community, and hence determine which query should be materialized in the cache at any given time. However, materialize only the queries indicated by association rules did not establish effectively a querying materialization plan for caching in the medium term. To support this in a more effective way we represented association rules in Markov chains. This allowed us to get a larger "horizon" for query materialization. Thus, this combination of techniques that provided us a very practical, not complex, way to establish *a priori* very practical querying materialization plans for an OLAP engine caching system, reducing consequently the workload of an OLAP server and improving its querying response time.

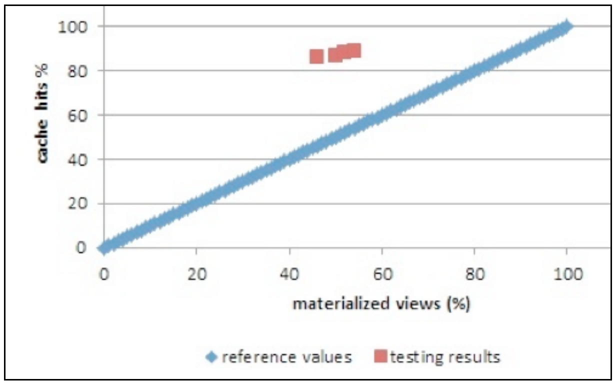


Figure 9. Test results graph for the second dataset.

Until the time we finished this work, we did not find any other caching solution for analytical systems such as the one we developed and exposed here. As such, we were unable to evaluate our proposal based on other alternative solutions that adopt or follow the same kind of strategy and categorization of patterns of analytic exploration we did. Even taking into consideration the works [14], [22], and [7], which revealed very curious and interesting aspects of predictive caching, we cannot make such comparative assessment.

## VII. CONCLUSIONS AND FUTURE WORK

In recent years, the large increase using multidimensional database systems led to a greater interest in research of new techniques for the improvement of OLAP systems' functionalities and services. As widely used in other areas, these techniques can highlight the implementation and operation of caching mechanisms in OLAP systems, decreasing analytical server working load. As for Web systems, initially on OLAP systems the implementation of such mechanisms was only on the client side. This meant that only a certain user benefit of previous orders. Consequently, and as a manner to increase the advantage achieved by using caches, such mechanisms has focused on sharing benefits for all users of a given OLAP server. Thus, through the distribution of caches or by means of shared caches among individual users, either through the creation of systems specifically designed to serve as cache servers, significant improvements were achieved in terms of performance [16][13][5][6][14].

The main goal of this study was to investigate in what conditions a predictive caching system could be used in a typical OLAP environment. In order to reach such goal, we studied several known cache techniques, e.g., [15][29][5][6][25][12][14][7][4], trying to establish the basis to propose a different manner to know *a priori* the contents of an OLAP cache in a near future. All those techniques were crucial to the development of our work, for both the ideas of exploring the log files present in the OLAP Server and the simplification of the rules generated after the application of mining algorithms to that information. All the tests performed showed satisfactory improvements in the ratio between materialized views and cache hits. In our perspective, they also showed that this approach has the necessary pre-requisites to be applied to a more real scenario with advantages for the overall system's global performance.

The results of all tests demonstrated that the simplification of a certain percentage of rules to be pre-materialized does not mean that the same percentage of requests cannot be served from the cache. The doubt that remains is that if this number of cache hits is small in terms of a percentage higher or lower than the simplification of rules. Analyzing the data generated by tests, both using dummy data (the first data set), as the data retrieved from a specific data repository (the second data set), the technique for simplifying cache maintenance used proved to be very beneficial in all the tested scenarios. Even though some important questions remain, both for the period of logs that should be analysed and for the values of *minconf* to be used. This last, is an issue that should be addressed on a case-by-case approach, and should be included in a typical tuning-phase after finishing system implementation.

Finally, we think that with larger datasets feeding the mining algorithm, results should be even better. With a greater number of test cases, preferably from real application scenarios, it is possible to define with greater precision which are the access patterns of users as well as what benefits arise from the application of the several techniques. For that reason we plan in a near future to extend the current study, comparing it with other similar approaches and including some work concerning the exploration of multidimensional queries. We will give particular attention to the less busy periods of an OLAP server, in order to pre-materialize some specific multidimensional views that can be used latter when a user logs in – the log in periods can, as well, be subject of prediction. Yet because of the scarcity of the available data, and because it was decided not to integrate in this work the process where OLAP server logs are analysed and where are acquired all the association rules that serve as input to the technique developed – we focused on the simplification of the already generated rules –, we plan to, in a near future integrate these two phases in the process described in this paper. In the short term, we need to evaluate in a more effective way the practical utility of the predictive caching technique developed, extending the analysis periods along with the process of making a comparative study with other similar and concurrent techniques.

## REFERENCES

[1] P. Marques and O. Belo, "Adaptive OLAP Caching, Towards a better quality of service in analytical systems," in Proceedings of The Second International Conference on Business Intelligence and Technology (BUSTECH'2012), pp. 42-47, Nice, France, July 22-27, 2012.

[2] A. Abelló and O. Romero, "On-Line Analytical Processing (OLAP)," in Encyclopedia of Database Systems (editors-in-chief: Tamer Ozsu & Ling Liu), Springer, pp. 1949-1954, 2009.

[3] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos, "Cubetree: Organization of and Bulk Incremental Updates on the Data Cube," in proceedings of the 1997 ACM SIGMOD international conference on Management of data (SIGMOD '97), J. Peckman, S. Ram, and M. Franklin (Eds.). ACM, New York, NY, USA, pp. 89-99, 1997.

[4] W. Zhenyuan and H. Haiyan, "OLAP Technology and its Business Application, Intelligent Systems," in proceedings of the Second WRI Global Congress on Intelligent Systems, pp. 92-95, 16-17 Dec, Wuhan, 2010.

[5] P. Kalnis and D. Papadias, "Proxy-server architectures for OLAP," in proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD '01), Timos Sellis (Ed.). ACM, New York, NY, USA, pp. 367-378, 2001.

[6] P. Kalnis, W.Ng, B. Ooi, D. Papadias, and K. Tan, "An adaptive peer-to-peer network for distributed caching of OLAP results," in Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD '02). ACM, New York, NY, USA, pp. 25-36, 2002.

[7] Q. Yao and A. An, "Using user access patterns for semantic query caching," in proceedings of Database and Expert Systems Applications, 14th International Conference. Prague, Czech Republic, 2003.

[8] M. Chrobak and J. Noga, "LRU is Better than FIFO," Algorithmica, Springer New York 23, pp. 180-185, 1999.

[9] V. Mookerjee and Y. Tan, "Analysis of a least recently used cache management policy for Web browsers", Operations Research, vol. 50, 2, pp. 345-357, Mar 2002.

[10] J. Boyar, M. Ehmsen, J. Kohrt, and K. Larsen, "A Theoretical Comparison of LRU and LRU-2," in Proceedings of the 4th International Workshop on Approximation and Online Algorithms, volume 4368 of Lecture Notes in Computer Science, pp. 95–107, Springer-Verlag, 2006.

[11] N. Megiddo and D. Modha, "Arc: a Self-Tuning, Lowoverhead Replacement Cache," in Proceedings of FAST '03: 2nd USENIX Conference on File and Storage Technologies San Francisco, CA, USA, March 31–April 2, 2003.

[12] W. Lehner, J. Albrecht, and W. Hümer, "Divide and Aggregate: caching multidimensional objects," in proceedings of the Second Intl. Workshop on Design and Management of Data Warehouses (DMDW 2000), Stockholm, Sweden, 2000.

[13] P. Cao, J. Zhang, and K. Beach, "Active Cache: Caching Dynamic Contents on the Web," Distributed Systems Engineering, vol. 6, 1, pp. 43-50, 1999.

[14] T. Loukopoulos, P. Kalnis, I. Ahmad, and D. Papadias, "Active Caching of On-Line-Analytical-Processing Queries in WWW Proxies," in Proceedings of the International Conference on Parallel Processing (ICPP '01). IEEE Computer Society, Washington, DC, USA, pp. 419-426, 2001.

[15] P. Deshpande, K. Ramasamy, A. Shukla, and J. Naughton, "Caching multidimensional queries using chunks," ACM.SIGMOD Rec. 27, 2, pp. 259-270, June, 1998.

[16] S. Bakiras, T. Loukopoulos, and I. Ahmad, "Dynamic Organization Schemes for Cooperative Proxy Caching," in IPDPS'03 (International Parallel and Distributed Processing Symposium), 2003.

[17] H. Gupta, "Selection of Views to Materialize in a Data Warehouse," in Proceedings of the 6th International Conference on Database Theory, Springer-Verlag, London, UK, 1997.

[18] V. Harinarayan, A. Rajaraman, and J. Ullman, "Implementing data cubes efficiently," ACM SIGMOD Record, vol. 25, 2, pp. 205-216, 1996.

[19] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized Views Seleccion in a Multidimensional Database," in Proceedings of the 23rd International conference on Very Large Databases. Morgan Kaufmann Publishers Inc: San Francisco, CA,USA, 1997.

[20] A. Bauer and W. Lehner, "On solving the view selection problem," in Proceedings of the 15th International Conference on Scientific and Statistical Database Management. IEEE Computer Society, Washington DC, USA, 2003.

[21] Y. Kotidis and N. Roussopoulos, "A case for dynamic view management," in ACM Transactions on Database Systems. ACM: New York, USA, 2001.

[22] C. Sapia, "PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems," in Proceedings of the Second International Conference on Data warehousing and Knowledge Discovery (DAWAK 2000), Greewich, UK, Septeber 2000, Springer LNCS, 2000.

[23] K. Ramachandran, B. Shah, and V. Raghavan, "Dynamic pre-fetching of views based on user-access patterns in an OLAP system," in ACM SIGMOD, 2005.

[24] L. Cherkasova, "Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy," HP Laboratories Technical Report HPL, 1998.

[25] M. Lawrence, F. Dehne, and A. Rau-Chaplin, "Implementing OLAP Query Fragment Aggregation and Recombination for the OLAP Enabled Grid," in proceedings of Parallel and Distributed Processing Symposium (IPDPS 2007), IEEE International, pp. 26-30 March 2007.

[26] J. Albrecht, A. Bauer, O. Deyerling, H. Günzel, W. Hümmer, W. Lehner, and L. Schlesinger, "Management of Multidimensional Aggregates for Efficient Online Analytical Processing," in Proceedings of the 1999 International Symposium on Database Engineering & Applications (IDEAS '99). IEEE Computer Society, Washington, DC, USA, pp. 156-164, 1999.

[27] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in J. Bocca, M. Jarke, and C. Zaniolo, editors, Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pp. 487-499, Santiago, Chile, September 1994.

[28] R. Howard, "Dynamic Programming and Markov Processes," MIT Press, June, 1960.

[29] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu, "What can databases do for Peer-to-Peer," in proceedings of WebDB, 2001.