Adaptive OLAP Caching

Towards a better quality of service in analytical systems

Pedro Marques
ALGORITI R&D Centre
University of Minho
PORTUGAL
pcmarkes@gmail.com

Orlando Belo
ALGORITI R&D Centre
University of Minho
PORTUGAL
obelo@di.uminho.pt

Abstract — **Nowadays, the use of Multidimensional Data Systems has become a part of everyday actions in medium and large companies. These systems, which concern mainly in aiding their users in the process of decision-making, have a large flexibility in data exploration and high performance response levels to queries. Despite all the existing techniques, it is sometimes very hard to maintain such levels of performance that the user demands. With the purpose of tackling eventual performance losses, other techniques where developed trying to reduce data servers load. One of such mechanisms is the creation of OLAP caches maintaining previous queries and serving them upon subsequent requests without having to ask to the server. Due to OLAP systems organization, it is possible to identify the characteristics of its users and its exploration patterns – what queries will a user submit during a session, their frequency and resources involved. However, it is possible to go one step further, and to predict exactly what data will be requested by a specific user and, especially, the sequence of those requests. This is called the prediction phase and is followed by the pre-materialization of views that correspond to the user's requests in the future. These views are then stored in the cache and served to the user in the appropriate time. The technique we propose here consists in maintaining 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.**

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

I. INTRODUCTION

Due to the amazing increase of companies' data repositories in the last decade, attentions are now turned to the implementation of more powerful ways of analyzing data. As a consequence, Decision Support Systems, and more specifically, *Online Analytical Processing* (OLAP) [2] [18] systems are being implemented in a large scale when compared to few years ago scenarios. As we know, one of the greater 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 into company's data. OLAP systems were so well accepted by decision makers that soon they 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 deal 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 [10] [11] [16], revealing benefits good enough to keep the focus on improving caching techniques in order to integrate them effectively on OLAP server systems.

The work 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. 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 the user, put in cache the results that support responses to a specific user query, providing it faster than if no cache was available in the OLAP platform.

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

 ISBN: 978-1-61208-223-3

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 into more five sections, namely: Section II shows a detailed overview about OLAP caching, its advantages and disadvantages; Section III presents some related work and discuss major characteristics of the problems we could face when dealing with high specialized caches, as are OLAP caches; Section IV, it's where we present our approach for a new model of OLAP caching; Section V reveals and discusses the results of the performed tests to validate the OLAP caching model proposed; and finally, section VI, presents some final remarks and conclusions, as well as some future research lines.

II. TO CACHE OR NOT TO CACHE

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 scenario in particular. One of the most common ways of evaluating the value of any cache algorithm is using a metric called *Hit Ratio*. This metric is the ratio between the number of requests that were in cache and the total number of requests that were made. 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 lead to the creation of another metric: *Byte Hit Ratio*. The latter metric has been vastly used to evaluate how a cache can satisfy its clients' requests.

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 it is not, to decide whether it should or shouldn't 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 hasn't 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 record will be removed as necessary, record by record. The main problem with this technique is the fact that it doesn't take into account 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 taking into account the timestamp of last access to a specific piece of data [15], the frequency of access to the data [6], or other more complex information such as the ones used by the Greedy Dual algorithm [5], 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 to the characteristics of the data neglecting its nature and its relevance to users.

III. OLAP CACHING

One 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. The roll-up operation is its direct counterpart, allowing viewing data at a higher level following 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. Kalnis and Papadias [10] proposed a solution 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 don't, 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.

Kalnis et al. [11] proposed another alternative 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 [8], and intended to allow a very high level of

 ISBN: 978-1-61208-223-3

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 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 [4]. 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 placed often by researchers, it 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. Deshpande et al. [7] used a new indivisible unit called chunk. This data unit, with a low granularity level was mapped in the cache in order to be aggregated to satisfy user's 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 central server. 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 he received came from the central server or the cache server.

As a last reference we selected the work presented by Sapia [16], 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 by Albrecht et al. [1] or Deshpande et al. [7].

IV. A New OLAP Caching Approach

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 – use to induce them to be focus on a relatively small subset of 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 he gathers from the analysis of the data, 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 de 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, giving 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 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. We used the well-known *Apriori* algorithm [3], which is one of the most used algorithm for mining frequent itemsets, 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:

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

- *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:

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

If we take $A \rightarrow B$ *(sup=0.3; conf=0.8)*, as an example, we can say that for every time a user issues the query *A* he will, in

 ISBN: 978-1-61208-223-3

80% of the cases, issue the query *B*, right after that. On the other hand, generally speaking, 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. 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. We could then simulate the user's interaction and place in cache the views he used mostly. 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 taken into account in the prediction phase, without affecting significantly the results. To do this, we choose to map all the sequences of queries predicted by the mining algorithm, representing them in a Markov chain [9] 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 taken into account in the prediction phase (*minconf*). Soon, we discovered that this action would not be enough if we wanted to reduce effectively the number of predicted queries. We needed to optimize a little bit more the process. 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.

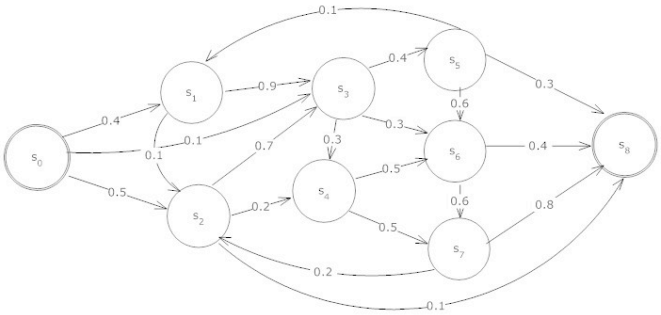

Figure 1. A query sequence prediction for the first dataset

V. 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 1, we can see the sequence of queries (in the form of a Markov chain) that were predicted by the mining algorithm that was used – S_0 and S_8 represent, respectively, the begin of the session provoked by the user's login and the end of that session. The vertices' values represent the transition probabilities between two different states (or queries in this case).

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 that was introduced, and named as "main route", simplistically put in the cache the sequence of queries that an user most likely follow in a future data exploration process. In Figure 1, easily we can 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. The results of the tests, for the different values of *minconf* and for the "main route" simplification models, can be found in Figure 2.

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 2). However, this value is merely meant to provide us with a reference value, and should not be taken into account in terms of absolute values.

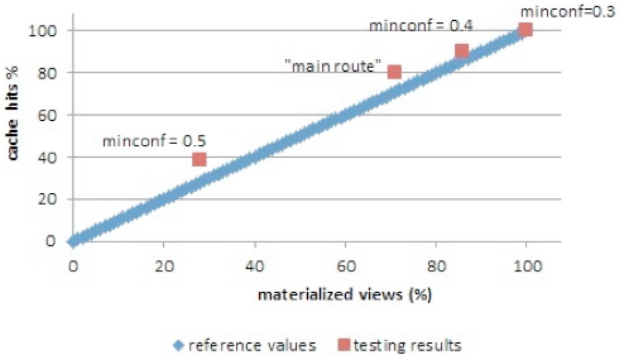

Figure 2. Test results graph for the first dataset

Observing Figure 2, it will lead us to note two key values of *minconf* – 0.3 and 0.5 –, which shows 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. On the other hand, the usage of 0.3 for *minconf* resulted in no view being simplified and, consequently, the values of cache hits were measured at 100%.

Later, other tests were conducted with a real set of data retrieved after a simulation of an OLAP Server usage. 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

 ISBN: 978-1-61208-223-3

second experience can be found in Table 2 and Figure 3. The results obtained in this second round of tests shows us that, even though the differences between the different values of *minconf*, they don't yield great differences in the percentage of cache hits – nor in the percentage of materialized views.

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

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).

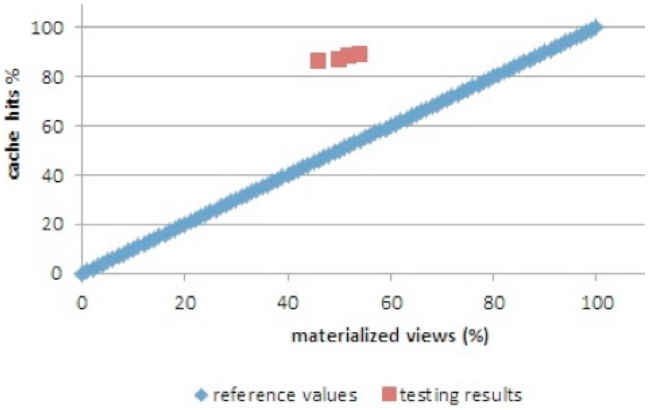

Figure 3. Test results graph for the second dataset

VI. CONCLUSIONS AND FUTURE WORK

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. [4] [7] [10] [11] [12] [13] [14] [16] [17], trying to establish the basis to propose a different form 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. 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. 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.

REFERENCES

[1] Albrecht, J., Bauer, A., Deyerling, O., Günzel, H., Hümmer, W., Lehner, W. and Schlesinger, L. (1999). 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, 156-,.

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

[3] Agrawal, R. and Srikant, R. (1994), Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

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

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

[6] Chrobak, M. and Noga, J. (1999). LRU is Better than FIFO. Algorithmica, Springer New York 23: 180-185.

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

[8] Gribble, S., Halevy, A., Ives, Z., Rodrig, M. and Suciu, D. (2001). What can databases do for Peer-to-Peer. In Proc. of WebDB, 2001.

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

[10] Kalnis, P. and Papadias, D. (2001). 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, 367-378.

[11] Kalnis, P., Ng, W., Ooi, B., Papadias, D. and Tan, K. (2002). 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, 25-36.

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

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

[14] Loukopoulos, T., Kalnis, P., Ahmad, I. and Papadias, D. (2001). 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, 419-.

[15] Mookerjee, V. and Tan, Y. (2002). Analysis of a least recently used cache management policy for Web browsers. Operations Research, 50, 2 (March 2002), 345-357.

[16] Sapia, C., PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems, In Proceedings of the Second International Conference on Data warehousing and Knowledge

 ISBN: 978-1-61208-223-3

Discovery (DAWAK 2000), Greewich, UK, Septeber 2000, Springer LNCS, 2000.

[17] Yao, Q. and An, A. (2003). Using user access patterns for semantic query caching. Database and Expert Systems Applications, 14th International Conference. Prague, Czech Republic.

[18] Zhenyuan, W. and Haiyan, H. (2010). OLAP Technology and its Business Application, Intelligent Systems, WRI Global Congress on, pp. 92-95, 2010 Second WRI Global Congress on Intelligent Systems.

 ISBN: 978-1-61208-223-3