Xml Query Processing – Semantic Cache System

M.R.Sumalatha 1[†] , V.Vaidehi[††] and A.Kannan 2[†††]
rsumalatha@yahoo.com,vaidehi@annauniv.edu, kannan@annauniv.edu
Anna University , Chennai, INDIA

Summary
With the advent of XML, great challenges arise from the demand for efficiently retrieving information from remote XML sources across the Internet. Recently, the advance of extensible Markup Language (XML) as the lingua franca of the Web meets the needs for an interoperable data exchange format on the Web. The semantic caching technology can help to improve the efficiency of XML query processing in the Web environment. Unlike from the traditional tuple or page-based caching systems, semantic caching systems exploit the idea of reusing cached query results to answer new queries based on the query containment and rewriting techniques. Fundamental results on the containment of relational queries have been determined. The results show that some of our measures outperform the traditional measures in certain situations.

Key words:
XML query processing, Semantic cache system..

1. Introduction

Caching popular queries and reusing results of previously computed queries are one important query optimization technique, especially in modern distributed environments such as the World Wide Web. Based on the recent proliferation of XML data and the emergence of the XQuery language, developing a query-based caching system for XQuery queries and rewriting strategies to answer incoming user queries based on the cached XQueries, whenever possible, instead of accessing remote XML data sources gives efficient information retrieval. To manage the space of the cache, a straightforward application of replacement strategies would correspond to removing a complete cached query and its derived XML document as a whole when space needs to be freed [1, 4, 5, 6].When semantic cache system is compared with traditional systems, one major difference can be stated, it is that the data cached at the client side of the former is logically organized by queries, instead of physical tuple identifications or page numbers. To achieve effective cache management, the access and management of the cached data in a semantic caching system is typically at the level of query descriptions. For example, the decision of whether the answers of a new query can be retrieved from the local cache is based on the query containment analysis of the new query and the cached ones, rather than by
looking up each and every tuple, or page identification of objects that could possibly answer a current user request [2, 3]. An important responsibility of cache management is to determine which data items should be retained in the cache and which ones should be replaced to make free space for new data, given limited cache space. Most naturally, the data granularity for replacement in a query-based caching system is the query and its associated result [7, 8, 9].

2. Semantic Cache System

The first method is to install it in a separate proxy server as shown in Figure 1 through which all clients connect to the server. This method leads to a substantial decrease in the amount of network traffic but the maintenance of the cache consistency becomes difficult due to the latency in communicating changes in the database from the server.

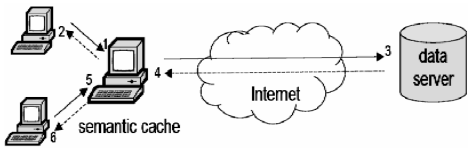

Fig. 1 Cache using Proxy Server

The second method is to implement cache on the server side, that it reduces the load on the web server. The advantage of using this method is that the load on the server decreases but it doesn't have any effect on the amount of network traffic.

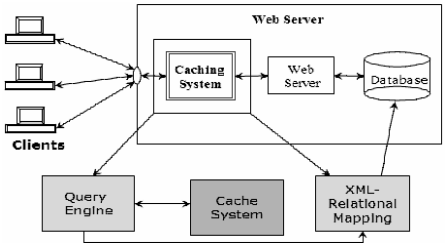

Fig. 2 Cache using Web server

Manuscript received April 5, 2007
Manuscript revised April 25, 2007

The first method may be more suitable since it reduces the response time as well as the network traffic since both the query and the result set is shortened. Our semantic cache can be effectively used in both the environments.

3. Partial Query Matching

In a semantic cache, the partial query matching is given more importance because if the data is partially available, logically it has to be analyzed and the correct result has to be given to the user. The concept of how cached contents are organized by queries is discussed first. That is, a hash-table like data structure is managed by a semantic cache to enable the associative access of cached data via the lookup by queries. For example, Figure 3 depicts an SQL query and its corresponding answer set represented by a spatial region scoped in terms of query predicates.

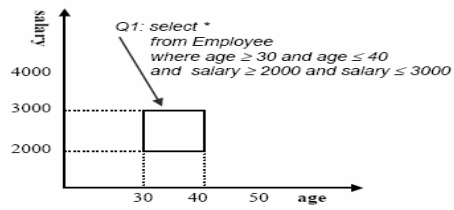

Fig. 3 Query Semantic Region.

Hence, each cached entry consists of a *(key, value, function)* triple for representing a semantic region. The key is a high-level query description extracted from the original syntactical format of a query. The value refers to the corresponding query answers accessible through the key. When a new incoming query is found to be semantically contained or overlapping with a cached query, a probe query is computed for retrieving the available portion of the query answers within the cache, and a remainder query is sent to the remote data server to fetch the remaining part of the query answers. Such a re-use of cached contents based on the semantic query containment relationship is depicted in Figure 4.

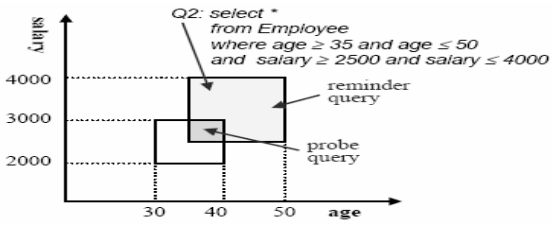

Fig. 4 Probe and Remainder Queries

4. Proposed Work

We have investigated the challenges imposed by the problem in defining architecture for efficient semantic cache management and provided an approach for tackling it using hash tables which implement the probabilistic hashing strategy for collision resolution. The Cache stores cached XML data with a semantic scheme. This semantic scheme consists of a set of patterns, and describes current cached XML data. The cached data is organized as an XML tree, which is a rooted sub tree of the XML tree exported by the XML database server. We have designed a mechanism for partial query matching by forming predicates and rules which perform semantic matching based on these rules which is an innovative method which makes efficient information retrieval.

Since the query joining issue plays a vital role in semantic caching we have analyzed how it can be joined effectively by proposing a dynamic hash mapping technique. The complex query given from the user is to be evaluated. This query is split in to simple sub queries by using the user defined functional modules, query decomposer, query marker and query trimmer. The query decomposer does the work of formation of sub query tree. The actual place where the division in to a simple sub queries is to be made is done by the query marker. The actual trimming in to sub query is done by the query trimmer. Now the Sub query is matched for its presence in the fact table which is a knowledge based information about the cache contents. If it is present and on obtaining a successful exact match then we draw up the conclusion the actual data is present in the cache. Actual work to be done is tuning the effective organization of the fact table entries or rules, that would result in obtaining faster matches with the incoming sub query. The rules of the fact table are the organization of the attributes of the query (ie) the name of the database, the relation R and the attribute of the relation of the document contents. We present this as an N-ary tree with the first level nodes as the resolving what database the incoming query requires. It proceeds to the next level if the database name matches with the one present in the fact table. Then a match or presence of the relation is checked in the fact table. This process is repeated until a failure or complete exploration of the N-ary tree. On the point of failure or success this fact table mapping technique returns the result depicting the presence in cache or not. On inferring the portion of the query present, hash function is applied on the attributes obtained from N-ary tree which maps directly to the actual data on the cache. The other part that is not present in the fact table is rewritten as "probe" and "reminder" part. The probe is the part where it's actual data is present in the cache and the reminder part

of the sub query is the portion that has to be sent to the server where the answer for the query is to be obtained from the database. This is the way in which the dynamic hash mapping reduces enormous amount of computation time of the complex subquery. We have the organization of the fact table in the form of an exhaustive dynamic n-nary tree where the incoming query's presence is checked. If the part of the query is not executed before then it is treated as "remainder" and it has to be sent to the server for evaluation. If the query or its part has been executed before then the path determining its database, table, attribute is returned. This returned value is employed to extract the actual value from the cache.

5. Semantic Cache Architecture

The Architecture of the entire cache system shows how the cache system works. As the XQuery comes, it is checked up with the cache system, if found return the results from the xml document stored. Then it will be send to the Tokenzier to split the Xquery into tokens and send to the partial matching module. If it doesn't match in the Naïve cache then it is partially matched up in the Rule set. If not found in the cache and the rule set means it will reported to the Query engine. If the system get a command from the server to update its cache with a query .The system will be then ready to receive a XML document. Then the server sends the XML document with some intermediate form between the cache system and server. The Semantic Cache will then interpret the XML document and store its values in the Cache table by performing all the operations like the hashing, query lookup, and replacement and store the results in a document.

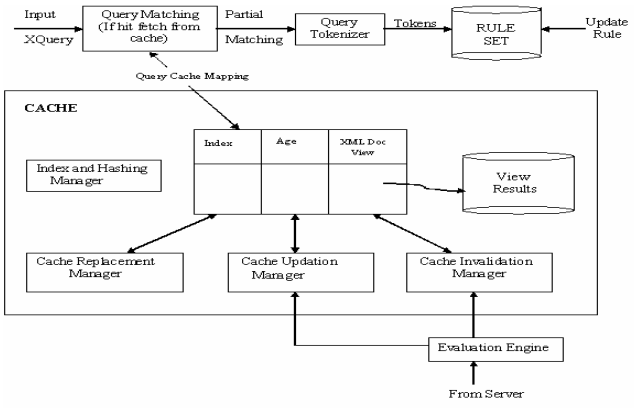

Fig. 5 Semantic cache Architecture

The partial matching structure in Figure 6 shows the working of the semantic partial matching when the entire query is not available in the cache and only partial results are available. For this purpose a partial match hash table is used to store semantic information about the queries in the cache. For queries that couldn't be answered from the naïve will be checked up with the partial matching cache to find any relation with the incoming queries that has some relation with the queries that present in the cache. If some relation exists then it will return back with the results and inform the query engine about the relation and remaining query formulation.

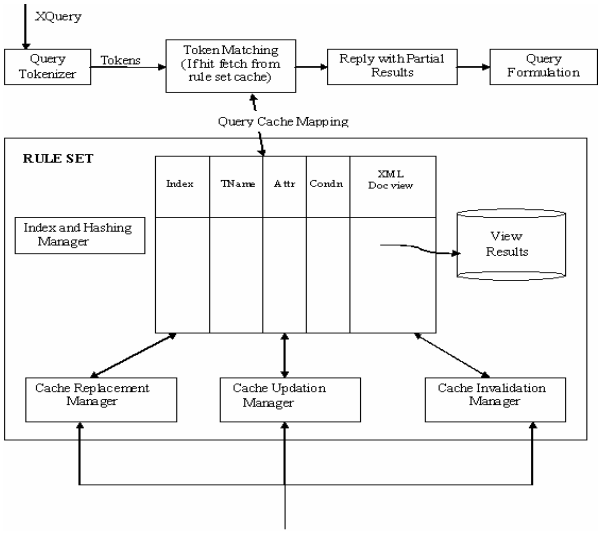

Fig. 6 Partial Matching structure

6. Implementation Issues

1.1.1

A. Steps in storing a query and its results in the cache

The incoming Query and its details will be cached and stored to the cache table,

Query : The incoming query

Present Bit: To signify the presence of the entry in a particular position.

Age : Number of times it has been accessed.

Query No :The unique number to identify each query.

The query is then hashed and the hash value obtained is used to store the above given fields are inserted into the hash table. The result of the query is appended to the

XML document with appropriate query number for the query node using LIBXML.

B. Partial Matching Table

The queries that were not matched in the cache will be partially matched up with the rule set. The rules are formed while the entries are added up in the cache. The incoming Query is split into the following parts based on FLWR expressions.

Tablename : Name of the table the query refers
Attribute : Name of the attribute in the table the query refers
Condition : The condition to be satisfied.
Value : The value of the attribute.
Query No : The unique number to identify the query.
Attr+Tab : This combination is used to index the data in the hash table.

The Attribute + Table value is then hashed and the hash value obtained is used to store the above given fields are inserted
into the hash table. Partial values will be computed by matching the operators, value and conditions. Partial return can also be answered from the cache.

2.

C. Probabilistic hashing & Cache Replacement Strategy

The cache is implemented using probabilistic hashing method this technique is particularly useful for implementing cache since it has more advantages compared to the Chaining concept in both time and space complexity. Since the probabilistic hashing is used, when the cache is full and a new entry must be added, and to make sure so that the query which is deleted should have its results removed from the linked documents and must be replaced with the result of the newly added query. Based on Age factor, replacement is carried out. In the cache table a field called 'age' is maintained for the no of hits for every entry for this purpose. Since probabilistic hashing is used there is no need for searching, and will directly replace the corresponding index which will result in efficient time complexity.

7. Result Analysis

The experimental and analysis of the result serve two main purposes:

- The first purpose is to validate the partial matching ideas for XQuery. For this, some example queries were picked from the W3C working draft "XML Query Use Case" [10] and run them through the XQuery engine with and with the semantic caching mechanism provided by the caching system. The correctness of the semantically matched queries by comparing their answers with those produced by directly executing the corresponding original queries is also checked.
- The second purpose is to determine whether or not the system can help to improve query performance. If yes, how much is the performance gain achieved, and what is the response time when the size of the cache is varied.

Without the aid of the cache, a distributed querying process can be broken down into three distinct stages. The cost distribution is analyzed in these three phases and aim to identify the factors that most strongly influence the user perceivable response time.

- The first stage is sending the query from the client to the server. There is a query shipping cost associated with this stage, although it should not fluctuate between queries as the size of queries themselves are very small compared to that of an XML document.
- The second stage is the execution of the query on the remote server against the originally specified XML document. This time component should greatly depend on both the query and the size of the XML document being queried against.
- The last stage of the process is sending the query results back to the client.

The result data shipping time associated with this stage is dependent on the size of the results and therefore also dependent on the query and the size of the source XML document. Roughly, the formula below can be used to indicate how the user perceivable response time is summed up.

$$\mathbf{T_{total} = T_{queryshipping} + T_{queryexecution} + T_{resultshipping}}$$

It can be seen from the above that the cost spent in the first and third stages involves the query shipping and the result data shipping across the network respectively. Hence, the network delay is a major factor besides the local query processing time.

The former is related to the network distance, the available bandwidth, the quality of network transmission, as well as the result data size. The latter is dependent on the efficiency of the query engine being used, the source document size and the query size. The time saving can be achieved in possibly all three stages by deploying the cache system as a proxy server in the LAN to assist the

query engine. That is, if the new query is totally contained within a cached query, then neither $T_{queryshipping}$ nor $T_{resultshipping}$ necessary any more, even $T_{queryexecution}$ will be scaled down since the XML document being queried against is not the original XML document but only the view document of the containing query, which can be reasonably assumed to be a small percentage of the original one. The caching system can be effectively designed by probabilistic hashing, which can be used for collision resolution due to its low response times. Also even if a query is overwritten due to collision, no loss is incurred since the original query can always be retrieved from the server.

The performance of the cache system is analyzed by plotting the average response time for various sizes of the cache. The results analyzed are shown in the figures below.

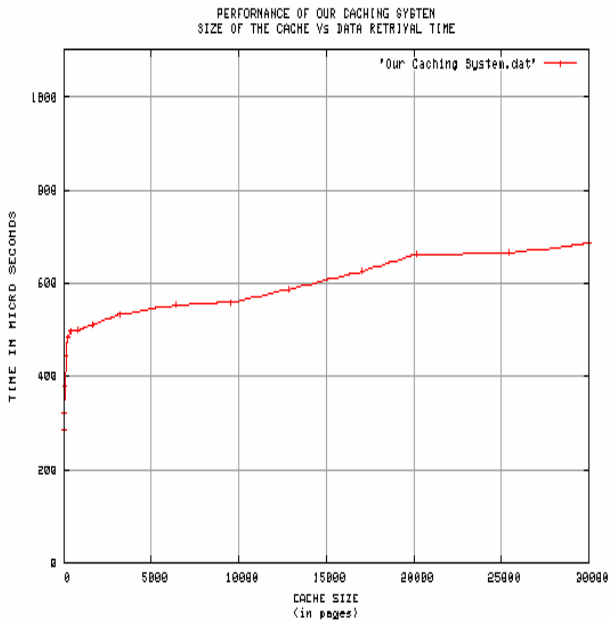

Fig. 7 Performance of Semantic cache systems

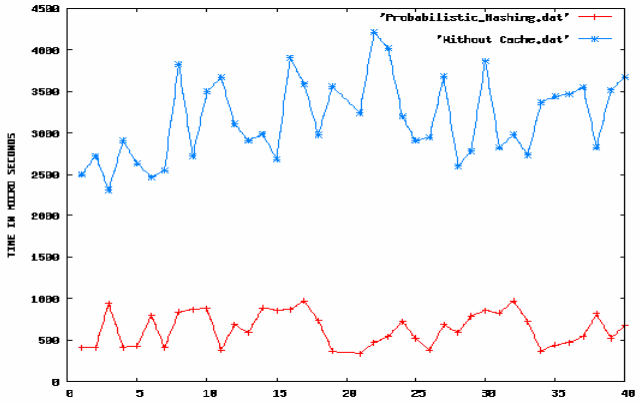

Fig. 8 Response Times with and without cache

The results show that the response time is almost the same even when the size of the cache is varied. This is due to the use of probabilistic hashing technique. The average response time of the system is around 600µs which is much lesser than the network latency incurred in answering a query from the database at the server. Also when partial results are retrieved using this method, there is significant reduction in the amount of data transmitted over the network and also reduces the load on the server and DBMS.

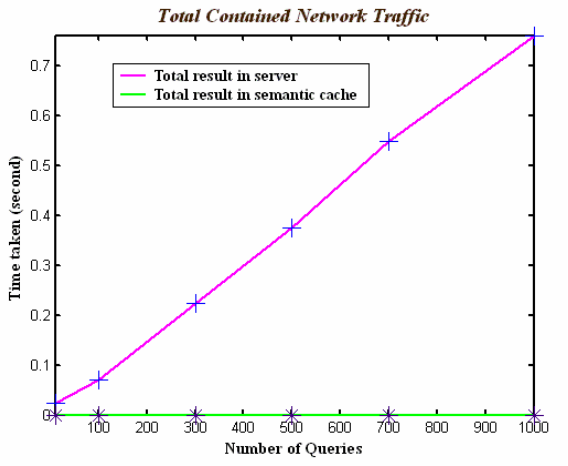

Fig 9. Network Traffic (Totally Contained)

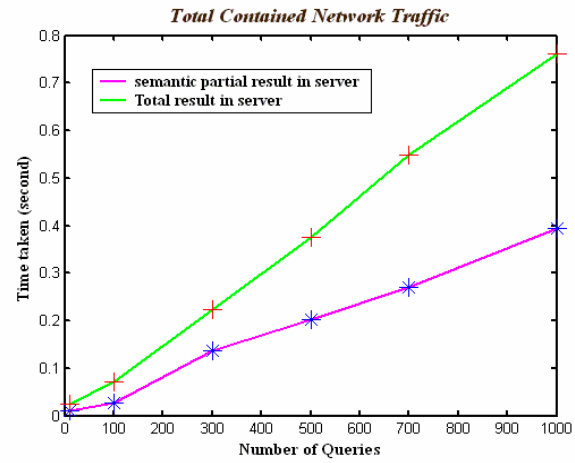

Fig 10. Network Traffic (Partially Contained)

8. Conclusions and Future Work

In order to improve the efficiency of XML query processing in the Web environment it is enhanced using the semantic cache system. Future research issues can give better solution for this system, provided it overcomes the

problems like replacement strategy that can better adapt to the user query access pattern and improve the cache space utilization, security issues. The system can also be made multithreaded so that it can process multiple queries simultaneously for better performance and parallelism.

References

[1]. Wanhong Xu, "The Framework of an XML Semantic Caching System", *WebDB* 2005, ACM, June 2005.
[2]. Bhushan Mandhani and Dan Suciu, "Query Caching and View Selection for XML Databases", *VLDB Journal* (2005).
[3]. Qun Ren, Margaret H. Dunham, and Vijay Kumar, "Semantic Caching and Query Processing", *IEEE transactions on knowledge and data engineering*, Feb 2003.
[4]. Li Chen and Elke A. Rundensteiner, "XQuery Containment in Presence of Variable Binding Dependencies", *ACM*, May 2005.
[5]. Li Chen, Song Wang, Elizabeth Cash, Burke Ryder and Ian Hobbs, "A Fine-Grained Replacement Strategy for XML Query Cache",
ACM (Nov -2002).
[6]. Athena Vakali, Barbara Catania and Anna Maddalena, "XML Data Stores: Emerging Practices", *IEEE Internet Computing*, April 2005.
[7]. Irini Fundulaki and Maarten Marx, "Specifying Access Control Policies for XML Documents with XPath", *ACM*, Jun 2004.
[8]. Jaap Kamps, Maarten Marx, Maarten de Rijke and Borkur Sigurbjornsson, "BestMatch Querying from DocumentCentric XML".
ACM , Jun 2004.
[9]. Li Chen, Elke A. Rundensteiner and Song Wang, "XCache – A Semantic Caching System for XML Queries", *ACM SIGMOD*, June 2002.
[10]. W3C. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, May 2003.

M.R.Sumalatha received the B.E. degree in Computer Science from Madras University and M.E. degree in Electronics Engineering from Madras Institute of Technology, Anna University in 1999 and 2001, respectively. During 2001-2007, she is in the Database area of Research. She is currently working in the area of Information retrieval in Database management systems and web services.