ISSN:0975-9646

Implementation of Query Result Caching Using Dynamic Data Cache

M. A. Ramteke[1], Prof. S. S. Dhande[2], Prof. H. R. Vyawahare[3]

Sipna College of Engineering and Technology, Amravati, Maharashtra, India

***Abstract*- Due to query result caching multiple read operations for the same result is avoided. It allows reuse of answers of previous queries, so reducing the delivery time of answers. In this paper we propose a cache management scheme. We present a method for reuse of query results for the future queries using dynamic data cache. Caching is the most important performance optimization technique. A dynamic data cache caches query results. Experimental evaluation shows that our caching scheme using dynamic data cache can reduce the cost of processing query workloads effectively. The performance of the proposed architecture is evaluated using hit rate and response time of query requests. The results obtained show that the proposed scheme is effective.**

***Keywords*— dynamic data cache, caching, query result caching, cache replacement, hit rate.**

I. INTRODUCTION

If you have queries that run over and over query result caching provides performance gains. Query result caching is a popular technique for reducing both server load and user response time. Due to query result caching multiple read operations for the same result is avoided.

The cache as a component improves performance by storing data such that future requests for that data can be served faster. The data that is stored within a cache might be query results that have been computed earlier. If requested data is contained in the cache it means if it is cache hit, that request can be served by simply reading the cache, which is comparably faster. Caching query results is one of the most crucial mechanisms to cope with a demanding load.

A large number of users submit queries to search engines on a regular basis in search of content. As the number of users is large and the volume of data involved in processing a user query is high, it is necessary to design efficient mechanisms that enable engines to respond fast to as many queries as possible. An important mechanism of this kind is caching. Also query result caching is an important technique employed in many information retrieval systems. A cache of query results that is dynamic data cache can serve the previously computed results of queries for a large number of queries. Hence, this technique helps a retrieval system to satisfy the low response time and high query processing throughput requirements, especially under high user query traffic volumes.

It is obvious that the cached query results will provide very high performance benefit over results that are not cached. A key to achieving high performance and scalability in client-server database systems is to effectively carry out query result caching. When there is a high probability of queries being repetitive in use, query result caching will provide better performance. Instead of wasting time in re-evaluating the query, the database can directly fetch the results from already stored cache. The most obvious benefit of query result caching can be seen in systems where data retrieval rate is very high when compared to data manipulation. Hence database gets modified after the long periodic intervals. During these intervals if a particular query is fired 100 times then the result of the query is calculated only once that is for the first time and 99 times the stored result is reused. Data Manipulation can invalidate the cache results because the inserted or modified or deleted data can bring the difference between the cached results and the actual results. Hence regeneration of the cached results will be required to restore the results back again to the useful state. If data manipulation rate is not low, database system will have to spend a considerable amount of time in bringing the invalidated results once again into a valid state, thus forfeiting the advantage of using this technique.

Dynamic caching of query results is more complex than caching static results, because the cached query entries may become invalid as a result of database writes. Dynamic data cache maintains previous query results. When a result is requested from an application, first is searched inside the cache and if found is automatically returned to the client, otherwise is loaded form database, cached and returned to the client. When an update, delete or insert query is received the cache will remove all cached query entries dependent on the affected tables. So that if next time query is fired related to that tables user will get right results.

Caching queries and reusing results of previously computed queries is one important query optimization technique. Search engines receive wide number of queries per day, and for each query, return a result page to the user who submitted the query. The user may request additional result pages for the same query, submit a new query, or quit searching altogether. Thus an efficient mechanism for caching query may enable search engines to lower their response time in obtaining the result pages. Also in web services communities systems that access distributed web services providers, an efficient query processing requires an advanced caching mechanism to reduce the query response time. A query result caching mechanism allows us to effectively use results of prior queries when the source is not readily available.

II. IMPLEMENTATION METHODOLOGY

A cache is an area of local memory that holds a replica that is copy of frequently accessed data that is otherwise expensive to get or compute. Example of such data includes a result of a query to a database. A cache works as the following: A user requests data from cache using query as a key. If the key is not found, the application retrieves the data from a backend database system and puts it into the cache. The next request for a key is serviced from the cache.

The implemented work is divided into three steps as,

1. Query result returned by cache
2. Query result returned by database and result stored in cache.
3. Eviction of query result from cache; update, delete, insert to database.

The approach is to cache the queries, if same query is requested in future its result can be reused and will improve the performance of application. It consists of techniques for query matching, consistency maintenance, and cache replacement to achieve the desired efficiency. To increase the speed in delivering dynamic web pages, database query result caching is useful. The proposed work is implemented in such a way that it receives SQL queries and will determine if they can be satisfied from dynamic data cache. Dynamic data cache maintains previous query results. The main issue is in retaining data consistency. The cache replacement component of dynamic data cache will consist of a replacement policy, which determines which cached query to replace, and a replacement mechanism, that determines how to replace cached query so that when cache becomes full new incoming query to be stored.

Dynamic caching of query results is more complex than caching static results, because the cached query entries may become invalid as a result of database writes. Dynamic data cache maintains previous query results. When a query result is requested from an application, first it is searched inside the cache and if found result is returned to the client, otherwise is loaded from database, cached and returned to the client. When an update, delete or insert query is received the cache will remove all cached query entries dependent on the affected tables. So that if next time query is fired related to that tables user will get right results. Objective of implemented work is to reduce query processing load, to result in users receiving query result faster using caching and to reduce load on backend database management system.

III. COMPONENTS OF DYNAMIC DATA CACHE

Following are the components of dynamic data cache.

Cache manager-

Cache manager maintains query results with query as a key. Cache is implemented using hash table.

Cache replacement algorithms-

The following cache replacement algorithm is used to replace the cached query results when cache becomes full. The goal of any cache is to maximize the hit ratio. The main difference in strategies is in how cached elements are selected for elimination when the cache becomes full.

a. First In First Out (FIFO):

Result sets are added to the cache as they are generated, when the cache is full, items are ejected in the order they were added.

b. Least Frequently Used (LFU):

The cached query result with least number of hits, the Least Frequently Used cached query result, is evicted. Result sets are added to the cache as they are generated; when the cache is full, the least recently used item is ejected. There is a need to have a replacement algorithm to purge entries from a cache when the boundary conditions are reached. For example, reaching the maximum number of entries allowed. One such algorithm is LFU (Least Frequently Used). The cached query results which were referenced in the frequent past, are not expected to be referenced again in the near future is ejected. LFU is widely used in database and web-based applications. In this algorithm cache entry which has been accessed frequently in past and whose access count is less will be replaced. If we are writing your own cache, one approach is to maintain a timestamp at which the entry was inserted and select the entry with the oldest timestamp also whose access count is less to be removed. This policy replaces the intermediate result that has been requested least frequently. The policy is based on the same principle as page replacement policies in operating systems. Every cached item is associated with a time stamp which is the last time of the item was accessed by a user, since the data server started execution. The item with the minimum time stamp is replaced when a new item must be stored in a cache when cache becomes full.

c. Most Recently Used (MRU):

This cache algorithm removes the most recently used items first. MRU algorithm is most useful in situations where the older an item is the more likely it is to be accessed.

Query matching

- Exact Match Queries
  Incoming query is matched with the query within the cache manager containing query results with query as a key.
- Semantically same written Match Queries
  When no exact match is found then it is checked whether it is semantically equivalent query (written in different way) submitted by the user, it is matched with each query which is stored in cache if match is found then its result is fetched from cache. Otherwise query is sent to backend database system.

Consider the following query select firstname, lastname from persons query is semantically equivalent to select lastname, firstname, from persons. If user fires first query its result will be returned to the user and query will be store in the cache to serve the same future request. Now if user fires second query same steps will be performed. Both the query has same result. Cache management scheme will cache both the queries and unnecessarily cache will get overload. So to avoid the case that even though result of both the queries are same both the queries will be stored in

cache; due to semantic match it will fetch the result of first query and second query is not cached since both the queries are semantically same and hence have same result and thus unnecessarily overloading of cache is avoided. Consider another query select * from employees where first_name='Steven' query which is semantically equal (written differently) to the following query as select * from employees where 'Steven'= first_name. Similarly if user fires first query its result will be returned to the user and query will be store in the cache to serve the same future request. Now if user fires second query same steps will be performed. Both the query has same result. Cache management scheme will cache both the queries and unnecessarily cache will get overload. So to avoid the case that even though result of both the queries are same both the queries will be stored in cache; due to semantic match it will fetch the result of first query and second query is not cached since both the queries are semantically same and hence have same result and thus unnecessarily overloading of cache is avoided.

Consistency manager-

When an update, delete or insert query is fired the cache will remove all cached query entries dependent on the affected tables. So that if next time query is fired related to that tables user will get right results. Read-only queries issued by client are satisfied from the cache whenever possible. Update transactions are always forwarded to the back-end database for execution, without first applying them to the cache.

Query parser-

When no exact match is found query parser performs semantic match with each query in cache. Also when update, delete and insert query is fired then on which tables all these operations is performed is determined by parser so if there is any cached query related to that tables it is removed from cache in order to get proper result when next time query is fired related to that tables.

Resource manager-

Here resource manager maintains statistics that is access count of query, last access time of query and store time of query. All these information is used by cache replacement algorithms in order to evict the cached query result when cache becomes full.

IV. GENERALIZED ALGORITHM

Following steps are performed for caching.

Input -user incoming query.

Output -result.

For each query perform the following steps

Step 1: Read query exists in cache, retrieve result from cache.

Step 2: Read query not available in cache (ie if no exact match found) then perform semantic match. If semantic match is found then return the result from cache.

Step 3: Read query not available in cache then

Invoke database server table; Query the corresponding database; Return the result; Store result in cache.

Step 4: If query is update, delete or insert then remove cached query result from cache.

V. RESULT ANALYSIS

The goal of any cache is to maximize the hit ratio. The performance of application demo that is simulator performance with caching was evaluated. The performance metrics used in the presented approach focus on the cache hit rate and response time. To evaluate the efficiency of policies, we use the hit rate of the cache and response time. In this experiment the performance of caching query results is evaluated, by comparing the effectiveness of several cache replacement algorithms: FIFO, MRU and LFU. Furthermore, the performance of the cache replacement algorithms is studied by estimating the strength of the cache content. This strength is evaluated by the consideration of the cached result retrieval rates as well as their frequency of access.

The database query result sets can be further categorized into two groups by their access and invalidation patterns: (1) high request rate and no invalidation; (2) high request rate and low invalidation rate.

Caching query results provide more gains in terms of efficiency, especially when the network communication dominates the query processing costs.

(1) High request rate and no invalidation

A set of queries was fired to test the performance gained due to caching under different cache replacement policies in which maximum number of queries was repeated and there was no updation, insertion and deletion.

Below figure shows performance gain in terms of average response time under different cache replacement algorithms, high request rate and no invalidation.

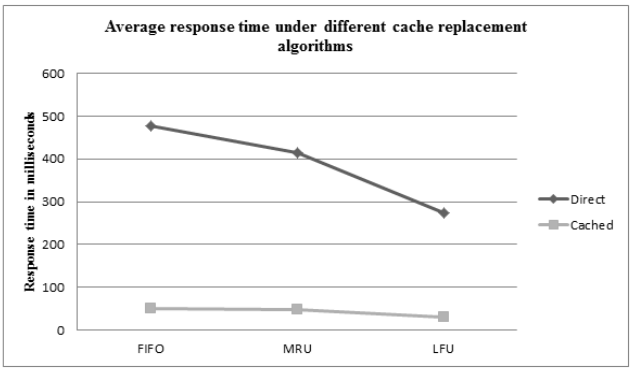

Figure1: Performance gain in terms of average response time under different cache replacement algorithms, high request rate and no invalidation.

Figure 2 shows hit rate of cache in form of percentage under different cache replacement policies in which maximum number of queries was repeated and there was no updation, insertion and deletion.

Cache hit rate represents the percentage of all requests being serviced by a cache copy of the requested result, instead of contacting the original database server. The higher the hit ratio, the better the response time in general.

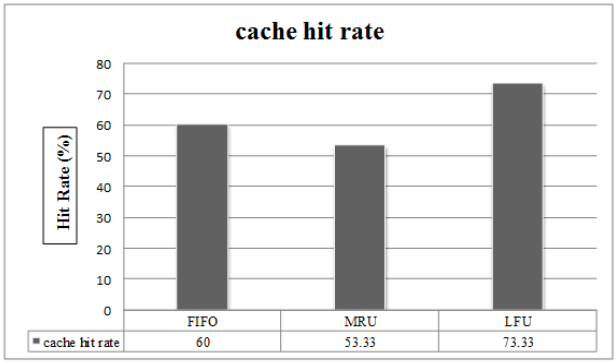

Figure 2: Hit rate in form of percentage under different cache replacement algorithms, high request rate and no invalidation.

(2) High request rate and low invalidation rate.

A set of queries was fired to test the performance gained due to caching under different cache replacement policies in which maximum number of queries was repeated and there was less updation.

It is obvious that benefit of query result caching can be seen when data retrieval rate is very high when compared to data manipulation.

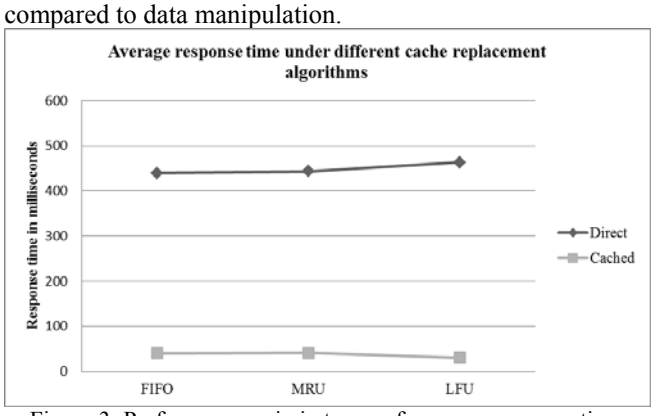

Figure 3: Performance gain in terms of average response time under different cache replacement algorithms, high request rate and low invalidation rate.

Figure 4 shows hit rate of cache in form of percentage under different cache replacement policies in which maximum number of queries was repeated and there was less updation.

Cache hit rate represents the percentage of all requests being serviced by a cache copy of the requested result, instead of contacting the original database server.

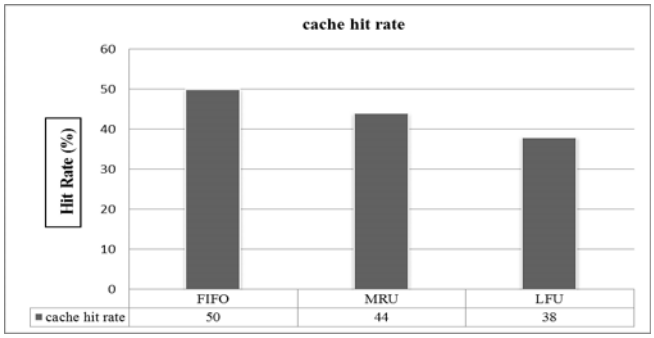

Figure 4: Hit rate in form of percentage under different cache replacement algorithms, high request rate and low invalidation rate.

VI. CONCLUSION

Our demo application shows effective use of query result caching. With increasing complexity of the query the benefits of caching keeps increasing. This is because complexity has a role to play when the result of the query is calculated and not when the calculated result of the query previously saved is brought from the cache. Time required to calculate the result of the query keeps increasing with the increasing complexity of the query while time taken to make reuse of previously calculated result remains almost constant. Thus caching query results increases the scalability of the backend database by serving number of queries at the dynamic data cache. This reduces average response time when the backend server is experiencing high load. It offloads origin backend system and provides better client response time.

REFERENCES

[1] Qiong Luo, Jeffrey F. Naughton, Rajasekar Krishnamurthy, Pei Cao and Yunrui Li. "Active Query Caching for Database Web Servers", D. Suciu and G. Vossen (Eds.): WebDB 2000, LNCS 1997, pp. 92-104, 2001. Springer - Verlag Berlin Heidelberg 2001.

[2] M. Altinet, Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, B. G. Lindsay, H. Woo, L. Brown, "DBCache: Database Caching for Web Application Servers", 612, SIGMOD 2002.

[3] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan, "DBProxy: A self managing edge-of-network data cache". Technical Report RC22419, IBM Research, 2002.

[4] C. Bornhovd, M. Altinel, C. Mohan, H. Pirahesh, and B. Reinwald, "Adaptive Database Caching with DBCache", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2004.

[5] Per-Ake Larson, Jonathan Goldstein, Jingren Zhou, "Transparent Mid-Tier Database Caching in SQL Server", June 9-12, 2003, San Diego, CA. 2003 ACM 1-58113-634-X/03/06 SIGMOD 2003.

[6] K. Amiri, R. Tewari, S. Park, and S. Padmanabhan, "On space management in a dynamic edge data cache". In WebDB Conference (Informal Proceedings), 2002.

[7] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan, "DBProxy: A dynamic data cache for Web applications", In Proc. International Conference on Data Engineering, IEEE Computer Society 2003.

[8] C. Mohan, "Caching Technologies for Web Applications" Available at almanden.ibm.com/u/mohan/Caching_VLDB2001.pdf , Rome ,VLDB 2001.

[9] Charles Garrod, Amit Manjhi, Anastasia Ailamaki, Bruce Maggs, Todd Mowry, Christopher Olston, Anthony Tomasic, "Scalable Query Result Caching for Web Applications", VLDB Endowment, ACM. VLDB `08, August 2430, 2008.

[10] Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H. Woo, B. G. Lindsay, J. F. Naughton, "Middle- Tier Database Caching for e-Business", 600- 611,SIGMOD 2002.

[11] Per-Ake Larson, Jonathan Goldstein, Hongfei Guo, Jingren Zhou, "MTCache: Mid-Tier Database Caching for SQL Server", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering.

[12] Laurentiu CIOVICĂ ,"Open Source Caching Solutions", Open Source Science Journal Vol. 2, No. 3, 2010.