ISSN:0975-9646

A Review on Query Result Caching using dynamic data cache

M. A. Ramteke, Prof. S. S. Dhande, Prof. H. R. Vyawahare

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

***Abstract*— Backend database system is often the performance bottleneck when running applications. The need for speed is increasing rapidly so we must be able to handle large amount of data in small time units like seconds or even milliseconds. The main problem that affects this to be fulfilled is the intensive usage of database through unnecessary, repetitive calls. A common approach to scale the database component is query result caching. Caching is a technique that can drastically improve the performance of any database application. Edge application directs its database requests to a database cache, rather than the backend DBMS. A solution for our performance problem and in gaining more speed in our application is to cache query results and where is possible our dynamic content.**

***Keywords*— data cache, caching, cached query result, cache replacement.**

I. INTRODUCTION

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 (cache hit), this request can be served by simply reading the cache, which is comparably faster. Web search engines serve millions of query requests per day. Caching query results is one of the most crucial mechanisms to cope with such a demanding load. An efficient storage model can be used to cache query results.

The performance of a caching system depends on the underlying caching data structure, cache eviction strategy, and cache utilization policy.

Data cache is a standalone database engine that maintains previous query results. When a piece of data 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.

Caching popular queries and reusing results of previously computed queries is one important query optimization technique. Popular search engines receive millions 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. An efficient scheme for caching query result pages may enable search engines to lower their response time.

II. LITERATURE REVIEW AND RELATED WORK

Many applications fail to be efficient due to a huge number of unnecessary database calls, network traffic between the application itself and the database server, retrieving and loading the results for same request from server. Qiong Luo, Jeffrey F. Naughton, Rajasekar Krishnamurthy, Pei Cao and Yunrui Li, proposed a new collaboration scheme between an active proxy and a database web server [1].

The backend database system is often the performance bottleneck when running applications. A common approach to scale the database component is query result caching, but it faces the challenge of maintaining a high cache hit rate while efficiently ensuring cache consistency as the database is updated.

Many online business applications today are being developed and deployed on multi-tier environments involving browser-based clients, web application servers and backend databases. The dynamic nature of these applications necessitates generating web pages on-demand, making middle-tier database caching an effective approach to achieve high scalability and performance. [2]. Caching is critical for improving the performance of many middleware applications. In order for an application to benefit from caching, it must repeatedly use data which is expensive to calculate or retrieve.

This limitation can be overcome by dynamic data cache. It saves the results of queries that are submitted to the database system. A cache hit is recognized and serviced from the cache if a query is an identical match to a previously submitted query. The advantage of such caching is that it is simple and it caters to access scenarios where the same query is likely to be submitted over and over.

A database cache feature is incorporated in DB2 UDB by modifying the engine code and leveraging existing database functionality. This allows us to take advantage of DB2's sophisticated query processing power for database caching [3]. As a result, the user queries can be executed at either the local database cache or the remote backend server, or more importantly, the query can be partitioned and then distributed to both databases for cost optimum execution.

By caching data, the application only needs to retrieve the data once. Whenever the data is needed after it has been cached, the application can retrieve it from a remote location. Database caching at proxy servers enables dynamic content to be generated at the edge of the network,

thereby improving the scalability and response time of web applications.

The scale of deployment of edge servers coupled with the rising costs of their administration demand that such caching middleware be adaptive and self managing [4]. To achieve this, a cache must be dynamically populated and pruned based on the application query for such a cache maintains a large number previous query results.

K. Amiri, R. Tewari, S. Sprenkle, and S. Padmanabhan proposed a cache that often rely on update propagation protocols to maintain consistency with back end database system. They focused on update propagation form backend database to the edge server cache [5].

Per-Ake Larson, Jonathan Goldstein, Jingren Zhou prototyped MTCache, a mid-tier database cache solution for Microsoft SQL Server in which mid-tier database caching is carried out, that is, running a local database server on each application server that caches data from the backend database. This allows queries to be computed locally [6].

A persistent and self-managing edge-of-network data cache is dynamically populated based on the application query stream and stored locally in a persistent database [7]. DBProxy, an edge-of-network semantic data cache for web applications adapt to changes in the workload [8].

C. Mohan presented an overview of caching technologies for web applications [9].

Ferdinand, the first proxy-based cooperative query result cache with fully distributed consistency management. To maintain a high cache hit rate, Ferdinand uses both a local query result cache on each proxy server and a distributed cache [10].

III. Cache Replacement

The following cache replacement algorithm can be used to replace the cached query results.

a. FIFO (First In First Out):

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 Recently Used (LRU):

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 LRU (Least Recently Used). In this algorithm cache entries which have not been accessed recently 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 to be removed. This policy replaces the intermediate result that has been requested least recently. The policy is based on the same principle as page replacement policies in operating systems. Every cached item is associated with a time stamp that stores the last time the item was accessed by a query, since the data server started execution. The item with the minimum time stamp is replaced when a new item must be stored in a full cache.

c. Most Recently Used (MRU):

It discards, in contrast to LRU, the most recently used items first. Many times during the usage of an algorithm, a list of the last n most recently used results comes to be useful. Sometimes, this is referred to the least recently used (LRU) cache, but this simply implies elements that fall out of the list (i.e. the least recently used ones). MRU algorithm is most useful in situations where the older an item is the more likely it is to be accessed.

IV. Applications

- To improve web search engines performance. Performance and scalability are critical to web search engines. It must be ensured not only that our web search engines always performs well, but that it will continue to do so as the user load increases.
- Caching technology can play in providing real-time data access, distributed applications.
- Popular search engines receive millions of queries per day, and for each query, it returns 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. An efficient scheme for caching query result pages may enable search engines to lower their response time.

V. Conclusion

Caching query results increases the scalability of the back-end database by serving a large part of the queries at the dynamic data cache. This reduces average response time when the back-end 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] L. Degenaro, A. Iyengar, I. Lipkind, and I. Rouvellou, “A middleware systemwhich intelligently caches query results”, In Middleware Conference, pages 24–44, 2000.

[3] 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.

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

[5] K. Amiri, R. Tewari, S. Sprenkle, and S. Padmanabhan, “Scalable consistency maintenance for edge query caches”, F.Douglis and B.D. Davison (eds.),Web Content Caching and Distribution, 79-70, 2004.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] Laurentiu CIOVICA Academy of Economic Studies, Bucharest, Romania, "Open Source Caching Solutions", Open Source Science Journal Vol. 2, No.3, 2010.

[13] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan, "Scalable template-based query containment checking for web semantic caches", In ICDE Conference, 2003.

[14] Hemant Kumar Mehta, Priyesh Kanungo, Manohar Chandwani **"**Dependency Free Distributed Database Caching for Web Applications and Web Services" 2nd International Conference and workshop on Emerging Trends in Technology (ICWET) 2011.

[15] Swaminathan Sivasubramanian, Guillaume Pierre, and Maarten van Steen, Gustavo the Alonso, "Analysis of Caching and Replication Strategies for Web Applications" Published by IEEE Computer Society IEEE 2007.