Answering Queries Using Cooperative Semantic Caching

Andrei Vancea, Burkhard Stiller

Department of Informatics IFI, University of Zurich
Binzmühlestrasse 14, CH—8050 Zürich, Switzerland
[vancea|stiller]@ifi.uzh.ch

Abstract. Semantic caching is a technique used for optimizing the evaluation of database queries by caching results of previous answered queries at the client side and using the cached results when trying to answer new queries. Before sending a query to the database server, the client first checks, if there are any cached query results that semantically contain the new query or parts of the query. If such cached results are found, they can be used when answering the new query. Otherwise, the query will be answered by the database management server.

This paper proposes to extend the general semantic caching mechanism by enabling clients to share their local semantic caches in a cooperative matter. If a particular query cannot be answered using the local cache, the system will verify, if there are other clients, located across the Internet, that are able to answer the query using the data stored in their caches. Such an approach will increase the throughput of database servers, because servers will only receive queries that cannot be answered using the cooperative cache concept.

1 Introduction

Database managements systems are, most of the time, build using the classical client/ server architecture [4]. A client sends a query (usually expressed in the Structured Query Language, SQL) to the database server and waits for the result of the execution. Caching data on the client side represents an important technique used both, for reducing the execution time of queries and also for increasing the throughput of the server [6].

In case of the application of the semantic caching mechanism [5] clients cache the result of the execution of old queries. In some cases, a subsequently query can be executed only using the cached data. Consider the following example: a client sends to the server a query, asking for all persons older then 15 (Q_1 : *select * from persons where age* > *15*). The server returns the result set, and the client stores it in the local cache. Later, the client requires all persons older then 18 (Q_2 : *select * from persons where age* > *18*). It can be clearly seen that the answer for Q_2 is totally subsumed by Q_1. Thus, Q_2 can be answered locally using the cached result set of Q_1. In the general case, new queries are, of course, not always totally contained in the cached queries. When there is just an overlapping between the new and the cached queries, the query is split into two disjoints parts: one that can be answered using the data contained in the cache (which is called the probe query) and a remainder query, that must be executed in the server [3]. Thus, a subsequent query that asks for all persons older then 10 (Q_4 : *select * from persons where age > 10)* will be split in a probe that asks for all persons older then 15 and a remainder that asks for all persons between 10 and 15. This split is done automatically, by analyzing the semantics of the queries.

Peer-to-peer networks have been applied successfully for enhancing beyond the traditional client-server communication, thus they are applicable to the distribution problem outlined. [8] presents CoopNet, a cooperative network architecture, where clients cooperate in order to improve the overall network performance. It is described how CoopNet is used for solving Web flash crow scalability problems. In this approach, clients that have already downloaded web content start serving the content to other clients, relieving the server of this task. The redirection of requests from the server to other clients is handled by a centralized component running at the server side. Thus, this approach does not integrate the distribution aspect.

Therefore, this paper develops a distributed and cooperative approach for reducing the load of database servers. Using a semantic caching approach, clients will store in their local caches the result of queries they requested. Local caches will be shared between clients in an cooperative matter. Before sending a query to the server, it will first be checked, if there are any other clients that have entries in their cache that can be used for answering the requested query. If such clients are found, their cache entries will be used when answering the query.

2 Approach

In the new approach proposed, clients are allowed to share cached query results in a cooperative matter. In order for this to be accomplished, a system named CoopSC (Cooperative Semantic Cache) is designed, which allows clients to register queries for which they have cached results and also to search for queries stored in the collaborative cache that subsume or overlap new queries for which they want the result. Two solutions for this system (Fig. 1) are foreseen: a *centralized* approach and a *fully distributed* one.

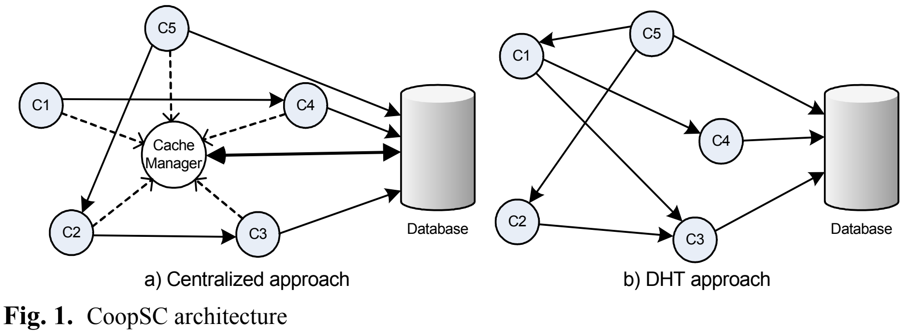

Fig. 1. CoopSC architecture

When using the centralized approach, clients register and look for queries in a centralized *cache manager*. The cache manager keeps, for each client, queries for which they have cached result sets. When a client decides to cache a new query or to drop an existing query, the cache manager must be notified. Before sending a query to the database server, the client will connect to the cache manager and asks for a query cached by another client that subsumes or overlaps its original query. If the cache manager finds one, it will return the identify of the client, the probe query and the remainder. The initial client will connect in turn to the client returned by the cache manager and asks the probe query. The remainder query can be executed by the database server or by using the semantic cache of a different client. This solution is similar with the one used by the CoopNet [8] system. The web server knows what data end-hosts contain, and redirects the requests from the server to clients when needed. In the CoopNet system, the mechanism for selecting the client to which to redirect a request is fairly simple. The web server only has to know the list of clients that keep the latest version of a particular web page. However, in CoopSC, deciding which clients can be used when answering a query, is much more difficult. It must determined which clients have cached queries that subsume or overlap the new query. In order for this to be determined, the semantics of these queries — stored in the cache and of the new query — must be analyzed. Since for this approach the description of all queries cached by different clients are stored in a central point, query containment verification and query rewriting are simplified. Unfortunately, this approach could have scalability problems. Because the cache manager must be contacted before every query execution it could become a bottleneck of the system.

A different approach is possible by applying a Distributed Hash Table (DHT) for indexing queries cached by different clients. After deciding to cache a query, a client will store the description (SQL) of the query into the DHT. The DHT will also be used, when looking for a query cached by another client that contains or overlaps a given query. On one hand, the advantage of this approach is that the cache content is stored in a fully distributed way. Thus, this solution will be much more scalable. On the other hand, looking for a query from the DHT that subsumes or overlaps a given query becomes much more difficult. Special consideration must be taken about the way queries are indexed in the DHT.

CoopSC has similarities with the Wigan [2] system. The purpose of both systems is to cache old result of database queries in order to answer new queries. The main difference is the way in which cache results are chosen for answering new queries. In Wigan, a cached query Q_1 can be used for answering a query Q_2 only, if Q_2 is strictly subsumed by Q_1. In real world applications, the number of cases in which this happens is limited. CoopSC does also support cases in which there is only an overlapping between Q_1 and Q_2.

Query containment and rewriting determine a fundamental concept related to semantic caching. A query Q_2 is set to be contained is a query Q_1 if Q_2 produces a subset of the answers of Q_1. If it has been decided that Q_2 is contained in or overlaps Q_1, Q_2 must be reformulate with the respect to the structure of Q_1. There has been an intensive research in the database community related to the query containment and rewriting problems [6]. It has been proven that the query containment problem is undecidable for relational algebra and SQL but, there are efficient algorithms for queries that have particular constraints [9], [7]. It is planned to investigate how these algorithms can be adapted and used in the cooperative cache architecture proposed.

Cache consistency is another important issue that must be handled by the CoopSC approach. After the execution of a modification in the server, some cache entry can become outdated. CoopSC must contain a mechanism for invalidating cache entries stored by clients that are no longer up-to-date.

In order to evaluate all benefits of the CoopSC architecture, it must be shown that, under heavy load, a database server performs better, when using the cooperative cache. For this to be proven, it is planned to test the functionality of the CoopSC by using a test-bed consisting of a database server and a number of clients machines that execute, in parallel, queries on the database. The same set of queries will be executed under three different scenarios: (a) without using the cache; (b) using only the local semantic cache; and (c) using the cooperative semantic cache. In each scenario the average query response time will be measured. It is expected that the average response time will be lower when using the distributed cache approach.

3 Conclusions

This paper skteches a new approach for answering database queries using a cooperative semantic caching mechanism. This solution proposed and partially outlined in terms of key aspects will increase the throughput of database management servers. The respective and general architecture of the new system termed CoopSC was described. The main issues concerning the implementation of the system were discussed. Furthermore, for the upcoming fine design and implementation of CoopSC, this paper also presents a possible evaluation.

References

1. M. J. Carey, M. J. Franklin, M. Livny, E. J. Shekita, Data caching tradeoffs in client-server DBMS architectures, ACM SIGMOD Record, Vol. 20, Issue 2, pp 357-366, May 1991.
2. J. Colquhoun, P. Watson, A Peer-to-Peer Server based on BitTorrent, Technical Report No. 1089, School of Computing Science, Newcastle University, April 2008.
3. S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, M. Tan, Semantic Data Caching and Replacement, 22th International Conference on VLDB, Bombay, India, pp 330-341, September 1996.
4. H. Garcia-Molina, J.D. Ullman, J.D. Widom, Database Systems: The Complete Book, Prentice Hall, June 2008.
5. P. Godfrey, J. Gryz, Answering Queries by Semantic Caches, Database and Expert Systems Applications, Florence, Italy, pp 485–498, September 1999.
6. A. Levy, Answering Queries Using Views: A Survey, The VLDB Journal, Vol. 10, No. 4, pp 270-294, December 2001.
7. A. Levy, A. Rajaraman, J.J. Ordille, Querying Heterogeneous Information Sources Using Source Descriptions, 22nd International Conference on VLDB, Bombay, India, pp 251-262, September 1996.
8. V. Padmanabhan, K. Sripanidkulchai, The case for cooperative networking, International Peer-To-Peer Workshop, Cambridge, MA, USA, pp 178-190, March 2002.
9. R. Pottinger, A. Levy, A Scalable Algorithm for Answering Queries Using Views, The VLDB Journal, Vol. 9, No. 1, pp 484-495, 2000.