8-15-1997

ENHANCING DATABASE PERFORMANCE IN A DSS ENVIRONMENT VIA QUERY CACHING

Aditya N. Saharia
University of Illinois at Chicago, saharia@uic.edu

Yair M. Babad

ENHANCING DATABASE PERFORMANCE IN A DSS ENVIRONMENT VIA QUERY CACHING

Yair M. Babad & Aditya N. Saharia
IDS Dept. (M/C 294), College of Business Administration
University of Illinois at Chicago
601 S. Morgan, Chicago, IL 60607
Contact: E-mail: Saharia@UIC.EDU, Phone 312-996-2370, fax 312-413-0385

ABSTRACT

A key element in all decision support systems is availability of sufficiently good and timely data to support the decision making process. Much research was, and is, devoted to data and information quality: attributes, assurance that quality data is used in the decision process, etc. In this paper we concentrate on a particular dimension of data availability and usage - the retrieval of data in a timely and decision enhancing manner. We propose to augment the decision support databases by an adaptive and efficient query cache. The cache contains snapshots of the decision support database, each being the answer to a recently invoked query. A snapshot can be reused by the originating user, or a different user, at a later time -- provided the use of cached data leads to savings over the use of a new query, and these savings exceed the cost of using stale date. The proposed scheme is conceptually different from conventional data replication schemes. In data replication schemes the data items to be replicated and the protocols for concurrency control are defined at the system level. In our scheme the cache is populated dynamically and the snapshots it contains are refreshed only if the cost of using stale information is higher than cost of refreshing the snapshots. At the same time, users can still decide to refresh the stored snapshot, based on their own decision environment. Our scheme thus enhances the data retrieval process, while supporting a more efficient data retrieval at both the user level and the data warehouse level.

QUERY CACHING

The common approach to query processing assumes that the information needs of users must be satisfied by the most current and up-to-date data. In particular, whenever a query for information is submitted, the database management system provides the desired answer by evaluation of the query at (or after) submission time. Once the answer is given to the user, the query processor drops both the query and the answer from consideration. Consequently, if the same query is resubmitted, by the same user or by another, the query must be reevaluated.

But this currency assumption may not uniformly apply for all the database users. Certain applications, such as long-tern planning and analysis activities, may need a long-term static view of the data rather than current operational data. Executive information systems often intentionally ignore the most current, and somewhat random, operational data in favor of long-term quarterly or even annual data. The use of slightly old date may be tolerated even in some operational applications, if it can save processing time and cost needed to sift through large volumes of current operational data (see reference 1.) Of

course, once the degree of staleness exceeds an acceptable level, the cost of possibly-erroneous decisions resulting from the use of stale data may justify the reevaluation of queries and the refreshing of answers.

Many different means for defining and maintaining snapshots can be devised. We suggest to augment the database by a ***query cache*** in which snapshots of recently invoked queries and their evaluated answers will be stored. This parallels the data warehouse approach, wherein the warehouse and datamarts contain snapshots of operational data which are refreshed periodically (see references 2 and 3 for issues related to the use and maintenance of a data warehouses for decision support applications.) In these systems, however, the decision whether to use stale data, and the timing of data refresh, are defined by preset warehouse parameters and not by the users. Further, in accordance with the data currency approach, the users of the warehouse are assumed to act independently, so that each userís query is evaluated against the warehouse and discarded (with the answer) once the answer has been passed to the user.

We, in contrast, support the reuse of snapshots of queries and their answers, by keeping them in the cache. The reuse of the snapshots is not limited to repeated presentations of stored answers. Rather, a comparison of a newly submitted query with the cached queries, may lead to a modification of the submitted query so that it uses some of the data in the cached answer. We also give the user a complete freedom regarding the use of the cached snapshots and the timing of the reevaluation of the query against the operational database.

To demonstrate, consider the database tables:

COURSES = (Course#, Description, College, Level) and OFFERINGS = (Course#, Term, Enrollment). Suppose user A submits the query:

Select College, Course#, Enrollment

from COURSES, OFFERINGS

where Level = Graduate and Term = Spring_97

and COURSES.Course# = OFFERINGS.Course#

This query, and the resulting evaluated answer, are stored in the cache as, say, Snapshot_Enrollment_A. Suppose user B later submits the query:

Select Course#, Enrollment

from COURSES, OFFERINGS

where Level = Graduate and Term = Spring_97 and College = Business

and COURSES.Course# = OFFERINGS.Course#

If the cached data is used, user Bís query can be processed much more efficiently, as:

Select Course#, Enrollment

from Snapshot_Enrollment_A

where College = Business

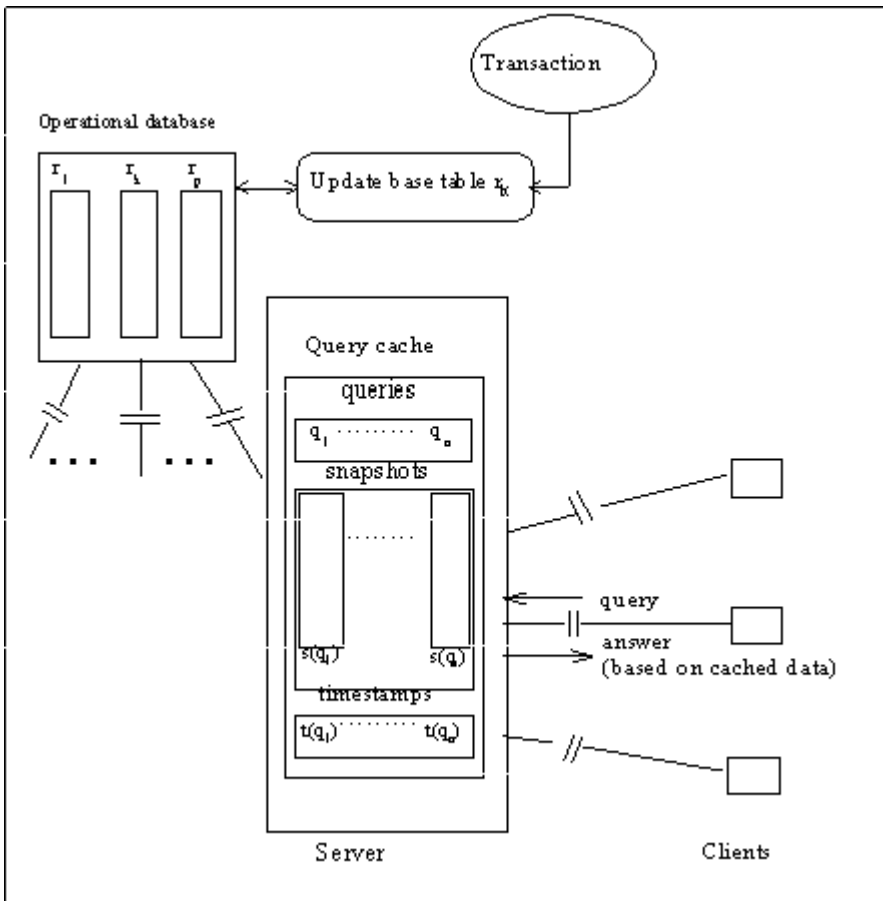

Figure 1. Architecture of the query caching scheme. Each entry in the cache is a triplet (q,s(q),t(q)) of the defining query q, the corresponding answer s(q), and the time t(q) at which the snapshot was generated or last refreshed.

A schematic view of the query cache scheme is given in Figure 1. Among the degrees of freedom available to users of the cache are:

Refresh time can be periodic, regardless of the actual use of the stale query and its answer; it can depend on the actual history of use of the query; or it can be on demand.

The **refresh method** can be a full refresh, in which the former answer is discarded and a new answer is fully reevaluated; or it may be a differential refresh, in which changes since the last refresh are maintained by the system and are applied when needed.

In a differential refreshing scheme modifications, additions and deletions are applied to the database tables. Each of these is checked against the queries stored in the query cache, to determine their relevance to the stored stale snapshots. Relevant changes are stored in a snapshot difference table. When a snapshot is requested from the query cache the contents of the difference table is applied to the stored answer. The relevant entries of the difference table are eliminated only when the related snapshots are refreshed.

A differential refresh strategy may be particularly attractive in a relatively slow-changing database, in which the rate of updates to the database that may impact stored usersí queries is relatively low. In this case, an attractive alternative is a differential snapshot which comprises the original snapshot, together with all additions, deletions and modifications (see references 4 and 5 for some example of differential refresh schemes.)

An important consideration is the **location of the answersí evaluation activities in a distributed database**, which may considerably affect the usersí decision regarding the use of the cached snapshots and the **location of the cache**. If refresh activities are done in a remote location, the user must take in account not only the query reevaluation and refresh costs, but also the transmission costs to and from the remote node. In case the transmission costs and/or delays are significant, the total cost of generating a fresh answer may be significant and may further justify the use of a stale answer.

Consequently, in a distributed environment a local query cache may be attractive. While this almost surely be the case for a cache that is maintained for a single user, an optimization model has to be constructed for a multi-user cached system. Such a model must take into account the overall structure of the distributed database (e.g., replicated, hierarchical, etc.), as well as the pattern of usersí queries, their timing, and their inter-relations.

Many additional modeling and optimization issues may be considered with regard to the maintenance and management of a query cache, including:

Searching the cache for a subsuming query, and - once such a query is found - constructing the modified query to be applied against the corresponding snapshot.

Estimating and alerting users as to the **degree of staleness** associated with a snapshot. If the snapshot is based on a single table, a simple definition of the degree of staleness would be the number changes to the base table since the last refresh. But when the snapshot is based on multiple tables, defining the degree of staleness is non-trivial. This estimate may be later used in a determination of the refreshing scheme, as discussed above.

Implementation of the refresh schemes. Depending upon the selected scheme, there may be a need for relevance checking algorithms to determine whether updates will affect a particular snapshot; this, for example, is the case for a differential refresh scheme. A complete solution for a single table, in which projections and selections are allowed, can be given in polynomial time by a clause satisfiability algorithm for a properly constructed clause. For a multi-table query with join operations, the problem is known to be hard.

Defining **performance evaluation** models to determine the set of snapshots that should be kept in the cache, and the most efficient order of their processing. In particular, this involves the order in which snapshots should be searched in the cache, the sequence of entered queries, and the determination whether a particular query is partially or fully subsumed by the stored snapshots.

A model for the **management of the cache**. Such a model will probably result in a modified LRU strategy, that takes in account the capacity of the cache, the refreshment policy adopted by users, the sequence of arriving queries, etc.

We are exploring some of these issues, and will report the results in future papers.

REFERENCES

1. Y.M. Babad. and A.N. Saharia, "Use of Stale Answers in Database Applications," *Proceedings of the 13th International Conference on Information Systems,* 1992.

2. K. Strehlo, "Data Warehousing: Avoid Planning Obsolescence," *Datamation,* Vol. 42, No. 2, 1996, pp. 32-36.

3. C. White, "The Key to a Data Warehouse," *Database Programming & Design,* Vol. 8, No. 1, 1995, pp. 23-25.

4. A.N. Saharia and G. Diehr, "Refresh Schemes for Remote Materialized Views," *Information System Research,* Vol. 1, No. 3, 1990, pp. 277-307.

5. A. Segev and J. Park, "Updating Distributed Materialized Views," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 1, No. 2, 1989, pp. 173-184.