Semantic Query Caching for Heterogeneous Databases

Parke Godfrey
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, Maryland 20783-1197
U.S.A.
godfrey@arl.mil

Jarek Gryz
Department of Computer Science
York University
Toronto, Ontario M3J 1P3
Canada
jarek@cs.yorku.ca

Abstract

Query caching can play a vital role in heterogeneous, multi-database environments. Answers to a query that are available in cache at the local client can be returned to the user quickly, while the rest of the query is evaluated. The use of caches can optimize query evaluation. By caching certain sensitive data locally, caches can be used to answer the parts of queries that involve the sensitive data, so it need not be shipped across the network. Most prior cache schemes have been tuple-based or page-based. It is unclear, however, how these might be adapted for multi-databases. We explore a more flexible *semantic query caching* (SQC) approach. In SQC, caches are the answer sets of previous queries, labeled by the query expressions that produced them. We promote developing the technology, based on logic, to manipulate semantic caches, to determine when and how caches can be used to answer subsequent queries, and to optimize via cache use.

Proceedings of the 4th KRDB Workshop
Athens, Greece, 30-August-1997

(F. Baader, M.A. Jeusfeld, W. Nutt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-8/

1 Introduction

Semantic Query Caching (SQC) has been proposed as a data-caching scheme for client-server environments [5, 12]. Under this architecture, the client maintains in cache *semantic* descriptions and the associated answer sets of previous queries. If the current query is answerable from cache, no communication with the servers is necessary. If a query is only partially answerable from cache, the amount of data needed from the servers may be substantially reduced.[1] SQC provides an alternative approach to *page-caching* and *tuple-caching* architectures [3, 6], in which the unit of transfer between servers and clients is a page or a tuple, respectively. The key advantage of SQC over other approaches is its flexibility: it can be used to answer new queries which are semantically related to previous queries that were cached.[2] Tuple-based and page-based caches are relatively inflexible. The new query must ask the *same* sources for much the *same* information, in the *same* format. Otherwise, it is not possible to determine the relevant pages or tuples. (See [5] for specific disadvantages.) Furthermore, in heterogeneous environments, it is not clear what tuple-based or page-based caching across databases even means.

We propose to extend the SQC paradigm in two orthogonal directions: first, we advocate broadening the domain of applications of SQC into heterogeneous database environments; and, second, we propose a logic-based framework for SQC and describe techniques for query optimization that become possible within the framework.

[1] The cache should also be *consistent*; that is, it is up-to-date. We ignore the issues of cache consistency, updating, and currency here, although we acknowledge their importance. We also do not discuss cache replacement strategies.

[2] This is commonplace during query sessions, in which a user asks a number of related and follow-up queries.

Our focus is on mediation-based systems over databases with structured data, such as provided by *TSIMMIS* [15] or the *Information Manifold* [14]. Systems such as these can be built over hundreds of data sources, each "wrapped" to translate the data models and languages of local databases into global concepts of the mediator. Queries are issued via the global concepts to the mediators, which identify the relevant sources, issue local queries to them, and then synthesize the results. Query optimization in heterogeneous environments has received increased attention recently [2, 7, 13], and by us as well [8, 9, 10].

We contend that SQC is needed in these environments and is critical for viable optimization strategies. In addition, SQC can help attend a number of other issues that arise in mediated, distributed environments.

- *Query optimization.*
  - *Improvement in overall query response time. (Traditional optimization.)* Since part, or all, of query processing can be done by the client via caches, the workload at the database servers is reduced. If the answer set of a query is large, computing part of it at the client also provides savings in network communication.
  - *Saving money.* In environments where there are monetary charges for information, such as in electronic commerce, caching techniques can be used to optimize over these monetary costs (instead of computational cost). That is, caching can save us from buying the same answer many times.
  - *Optimization of queries with few answers.* If the cardinality of the query's answer set—for instance, that there is only *one* answer—can be determined in advance (perhaps by reasoning with integrity constraints) and the number of answers to the query in cache is equivalent to the known cardinality, then the cached answer set is known to be complete.
- *Data Security.* We can limit the shuttling of sensitive data across the network by storing it at the client. Such data does not have to consist of complete tables; it can be defined as parts of tables in the same way views are defined. Except for the fact that caches defined in this way are never purged from the client, they can be treated in the same way cached queries are.
- *Fault tolerance.* Some databases may not be accessible at a given time. If a query can be partially computed from caches, at least *some* of the answers can be returned to the user.
- *Approximate answering.* Sometimes a good approximation of aggregate values such as *average* can be obtained from caches. If it can be determined that a cache contains a representative sample of the tuples over which the aggregate function is to be computed, then it can be evaluated just over cache. Techniques for approximate computation of averages are investigated in [11].
- *Better user interaction.*
  - *Answer set pipelining.* A subset of the answers that are computable at the client by cache can be returned to a user *promptly*, while remaining answers are being evaluated.
  - *Indirect answering.* The information that the query is contained in cache may sometimes be all that the user requires. This happens, for example, when in a sequence of queries it can be determined that the next query does not add any new "tuples" to the answers previously retrieved. Such information not only improves the query response time, but also can provide the user with valuable semantic information about his or her queries.
  - *Limiting the size of the answer set.* In some applications, a user may not be interested in retrieving all answers, but may be satisfied with just some (that is, with just a subset of a complete answer set). It might also be the case that the user might want to terminate the query evaluation if he or she finds that the answer set is larger than expected. In both cases, query processing can be terminated after retrieving answers from cache.

2 Answering Queries by Semantic Query Caches

A cache is useful when some of the query's answers can be obtained via the given cache, with project and select operations. (A cache is stored as a relational table.) If the query is answerable *entirely* from a cache, no communication with the servers is necessary. Otherwise, a single *trimmed query* [12] is sent for processing. We propose to relax this basic paradigm in two ways:

1. we advocate to allow for all relational operations to be performed over the caches (at the client); and

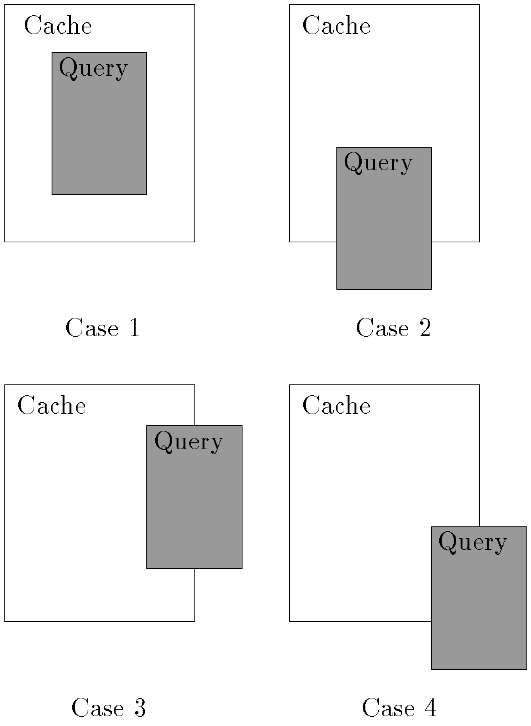

Figure 1: Possible relations between cached and current queries.

2. we allow for the possibility of a *dialogue* between clients and servers, not simply requests for answers to a query.

Consider Figure 1. The boxes represent tables: the horizontal represents rows (tuples), and the vertical columns (attributes). Assume initially that the only operations to be performed on the cached query (the white boxes) and the current query (the grey boxes) were selects and projects. The current query may overlap with a cached query in a number of possible ways. The query may be computable entirely from a cache (case 1), or only partially (cases 2, 3, and 4). Case 2 represents the situation when *some* of the answers to the query are available from cache; we refer to this case as a *horizontal* partition of a query. Case 3 represents what we call, a *vertical* partition of a query. If a cache contains the key column(s) of the relation in the query, the missing columns could be imported from the server, to be joined with the cache locally. Case 4 represents a mixed partition of a query.

The scenario above can be generalized so that both caches and queries may involve joins. Then the white boxes in Figure 1 may represent (the resulting table of) an arbitrary join of *several* caches stored locally. Even in the simplest case (1)—when all columns of interest of the query are in caches—it may be impossible to compute the answer set to a query without more information from the servers. For example, if a cached query $\mathcal{C}$ is the join $R_1 \bowtie ... \bowtie R_n$ and the current query $\mathcal{Q}$ is the join $R_1 \bowtie ... \bowtie R_n \bowtie R_{n+1}$, then we would need to import at least the join column(s) of the relation R_{n+1} to the client to evaluate $\mathcal{Q}$ in cache. Therefore it may be necessary for clients and servers to engage in *dialogues* to decide when this strategy is better than evaluating entirely the query at the servers.

Cases 2, 3, and 4 introduce yet another complication: *trimming* the query to evaluate at the servers. This is not necessary, but one can optimize by not evaluating the entire query at the servers when many answers are already known from cache. Ideally, one would want to evaluate only for the answers that were not in cache. Of course, trimming strategies cannot optimize universally; sometimes, a trimmed query will be more expensive to evaluate than would be the original query. We consider a trimming strategy in [9] that we believe is an optimization in general.[3] How "trimming" should be done depends on the *goal* of SQC: answer set pipelining generally benefits from heavier use of caches than is necessarily the case for optimizing the overall query response time.

3 Reasoning over Semantic Query Caches

We employ the terminology of logic databases and Datalog [19].[4] Whenever a query's answer set is cached, introduce a new predicate, say c, representing the cache, and a rule[5] for the cache predicate into the (intensional) database to reflect the query.

$$c\langle \vec{x} \rangle \Leftarrow q_1\langle \vec{x}_1 \rangle, \ldots, q_k\langle \vec{x}_k \rangle.$$

Let the collection of caches accumulated so far be $c_1, \ldots c_n$. Analytical tools developed for logic databases can be employed to decide when a cache (or combination thereof) answers, or partially answers, the query [20]. The basic inference needed is *conjunctive query subsumption* [19]. Tests for this are well known [4, 16]. This determines when one conjunctive query logically subsumes another. Whenever this is the case, all answers to the latter—say, a cache—are

[3] The trimming strategy, *tuple tagging*, proposed in [9] is for removing parts of a query that are known in advance to evaluate empty, as determined by semantic query optimization techniques. We are considering how to extend this for the more general case when parts of a query are known via cache.

[4] Logic databases are also known as deductive databases.

[5] Actually, we might pedantically argue that it is an *integrity constraint*, not a *rule*, that should be introduced since the cache itself is extensional (that is, materialized as a table).

also answers to the former—say, the query. Thus, the subsumption test alone is sufficient for the simplest case of SQC above (case 1 from Figure 1 without joins considered).

SQC with joins (and especially SQC with dialogue) requires more sophisticated inferencing. In particular, full Datalog subsumption must be considered [1, 18]; that is, when the collection of caches subsume (or partially subsume) the query. Also, when caches and queries are not simply conjunctive queries—that is, they may employ view predicates—more sophisticated reasoning is needed. We consider such issues in [8, 9].

There is also the case when one can determine that a query's answers (or some of its answers) are in cache, but one does not have enough information logically to *identify* those tuples constructible from cache that belong to the query's answer set. There are situations, however, when this weaker knowledge is still useful. (We discussed this in the point *indirect answering* on page 2.) In situations or applications in which a user is asking a sequence of queries in an attempt to find certain information, he or she is not interested to know which answers belong to which query. Thus, when a new query adds no new answers—that is equivalent to say that all its answers are known to be in cache—that is all one needs to know. One does not need to *construct* that query's answer set.

In the case that answers to the query are known to be in cache, but how to identify them is not, always with more information from the servers, one could identify the answers. The same analytic tools we use to determine when a query is answerable via cache can be adapted to identify the "missing" tables that would allow us to infer the answers in cache. So if we want the answers from cache, the client could request such additional tables from the servers in dialogue. It is complex to determine when such a dialogue exchange could be used to optimize query evaluation. It may easily be that the additional information needed from the servers would be more expensive to fetch than just evaluating the query as is at the servers.

However, such capability offers us much more flexibility, because it offers us alternative ways to evaluate the same answers using cache that would otherwise not be available to us. For instance, information we need to extract the answers from cache may be available at currently accessible servers, whereas evaluating the query may not be currently possible because some requisite servers are not accessible. In electronic commerce, the information needed for extraction might be inexpensive, whereas evaluating the query (without exploiting the caches) would be prohibitively expensive. In data security, we might be able to extract answers from cache locally with the use of additional public information from the servers, but not be able to pull those answers directly because they contain sensitive data that the servers are not allowed to ship on the network.

Another important technology needed is to be able to *remove* semantically portions of a query. If we can answer partially a query at the client via caches, ideally the trimmed query sent to the servers would only result in answers not already found (or, at least, would minimize the overlap). This reduces potentially redundant calculations and saves in network bandwidth. In [8], we introduced the the concept of a *discounted query* which can, in part, accomplish this. In [9], we provide an evaluation strategy for discounted queries that is, in general, more efficient than evaluating the queries themselves.

4 Conclusions and Issues

For successful SQC, many issues need to be resolved.

- Reasoning over conjunctive query containment and DATALOG containment is computationally hard [20]. When we allow view (intensional) predicates in queries, this is harder yet. So is SQC worthwhile?
  - While such reasoning may be complex, it is CPU-bound and can be done in main memory. The savings are over I/O and network bandwidth. So as the gap between CPU speeds and I/O speeds widens, we can afford to spend much effort on inference in return for more optimal queries and for pipelined answers.
  - Detecting cache use does not need to be *complete*. We can curtail search early if reasoning threatens to be expensive. Our algorithms should be incremental insofar as they provide some results if stopped early (rather than being *all-or-nothing*).
- What would cost models for SQC be? In particular, what would cost models for dialogue SQC be?
  - Certain applications of SQC are always a win, and, thus, should always be done.
  - Few optimization strategies in wide use today are universally optimizing. They can

back-fire, resulting in worse performance for a given query. However, good optimization techniques perform well in average case. We believe reliable heuristics can be designed as well for optimizing by cache.
    - SQC offers the flexibility to optimize over different criteria (or a combination thereof), such as over monetary charges and execution time.

- Can caches be kept "reasonably" current inexpensively? What are the criteria for "reasonably" current? How is currency to be achieved and/or guaranteed?
    - In many distributed domains, currency is a less critical issue, as over the Internet.
    - The same analysis tools for determining cache usage can be used to test cache currency.
    - Even when a cache is no longer current, we still can use it safely sometimes, when we know in what way it lost currency. (For instance, we might track the context queries of the updates since the cache was created.)
- What would be a reasonable cache maintenance strategy?
    - Semantic caches can sometimes be merged when closely related. When one cache subsumes another, the "redundancy" should somehow be removed.
    - As with all cache schemes, we need a balance between keeping too much information in cache and assuring effective cache use by future queries.

Semantic query caching can be readily incorporated over existing database systems. Datalog subsumes the relational algebra and many aspects of SQL. Thus, the reasoning machinery needed for SQC is adequate for the types of queries we encounter in today's database systems. While reasoning over caches is computationally hard (and, in Datalog with recursion, is for certain tasks undecidable), we believe there are approaches that can perform well, in average case, and that can sacrifice completeness while still being useful. Other approaches, such as those offered by description logics, are simpler computationally by design. However, they do not readily apply to existing systems. The queries of the underlying systems would have to be restricted to within the description logic, but database systems are rarely so constrained.

We believe that the issues above for SQC can be addressed. If so, SQC will provide valuable tools that address a number of critical issues facing the rise of heterogeneous database environments, including optimization, fault tolerance, data security, and answer pipelining.

Acknowledgments

This research was supported in part by grant NSF IRI-9300691 under Professor Jack Minker at the University of Maryland, College Park. Until recently, Jarek Gryz was affiliated with the University of Maryland, and Parke Godfrey currently is. We thank Professor Minker for his help, guidance, and insights in this work and in general. We also thank the ARL for their continued support of this work.

References

[1] S. Abiteboul, Y. Sagiv, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Adali, S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. SIGMOD*, pages 137–148, Montreal, Canada, June 1996.

[3] M. Carey, M. Franklin, and M. Zaharioudakis. Fine-grained sharing in page server database system. In *Proceedings of Sigmod*, 1994.

[4] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on the Theory of Computing*, pages 77–90, 1977.

[5] S. Dar, M. Franklin, B. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proceedings of VLDB*, 1996.

[6] D. DeWitt, P. Futtersack, D. Maier, and F. Velez. A study of three alternative workstation-server architectures for object-oriented database systems. In *Proceedings of VLDB*, 1990.

[7] W. Du, R. Krishnamurthy, and M.-C. Shan. Query optimization in a heterogeneous DBMS. In L.-Y. Yuan, editor, *Proc. VLDB*, pages 277–291, Vancouver, British Columbia, Aug. 1992. Morgan Kaufmann.

[8] P. Godfrey and J. Gryz. A framework for intensional query optimization. In D. Boulanger, U. Geske, F. Giannotti, and D. Seipel, editors, *Proceedings of the Workshop on Deductive*

Databases and Logic Programming, GMD-Studien Nr. 295, pages 57–68, Bonn, Germany, Sept. 1996. GMD-Forschungszentrum. Held in conjunction with IJCSLP'96.

[9] P. Godfrey and J. Gryz. Intensional query optimization. Technical Report CS-TR-3702, UMIACS-TR-96-72, Dept. of Computer Science, University of Maryland, College Park, MD 20742, Oct. 1996.

[10] P. Godfrey, J. Gryz, and J. Minker. Semantic query optimization for bottom-up evaluation. In Raś and Michalewicz [17], pages 561–571.

[11] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *Proc. SIGMOD*, 1997.

[12] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. *The VLDB Journal*, 5(2):35–47, Apr. 1996.

[13] C. Lee, C.-J. Chen, and H. Lu. An aspect of query optimization in multidatabase systems. *Sigmod Record*, 24(3):28–33, Sept. 1995.

[14] A. Y. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. 22nd VLDB*, 1996.

[15] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceeding of ICDE*, Mar. 1995.

[16] R. Ramakrishnan, Y. Sagiv, J. D. Ullman, and M. Y. Vardi. Proof tree transformation theorems and their applications. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems (PODS)*, pages 172–181, 1989.

[17] Z. W. Raś and M. Michalewicz, editors. *Foundations of Intelligent Systems: Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Artificial Intelligence 1079, Berlin, June 1996. Springer.

[18] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *Proc. 6^{th} ACM Symposium on Principles of Database Systems*, pages 237–249, 1987.

[19] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volumes I & II.* Principles of Computer Science Series. Computer Science Press, Incorporated, Rockville, Maryland, 1988/1989.

[20] J. D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory (ICDT'97)*, Delphi, Greece, Jan. 1997.