High Performance and Scalability Through Associative Client-Side Caching

Julie Basu
julie@cs.stanford.edu
Stanford University
Computer Science Department
and Oracle Corporation

Meikel Pöss
s_poess@ira.uka.de
Technical University of Berlin
Computer Science Department
D-10587 Berlin, Germany

Arthur M. Keller
ark@cs.stanford.edu
Stanford University
Computer Science Department
Stanford, CA 94305-9020, USA

March 24, 1997

Abstract

*A*Cache is an associative client-side caching scheme proposed in [6]. The cache at a client site dynamically loads query results and uses predicates based on the queries to describe its current contents. These predicate descriptions are used to determine query containment for local query evaluation at the client and also for cache maintenance. In this paper, we demonstrate via a simulation study that the A*Cache system provides high performance and scalability with respect to different workloads and large number of clients.*

1 Introduction

Client-server configuration is a popular architecture for modern databases. This setup involves one or more server processes that manage a repository of persistent data and handle requests for data retrieval and update from multiple client processes. Clients are autonomous entities that may be located on the same machine as the database server or on different machines. With the current prevalence of distributed computing, the latter scenario is most common, where client transactions are initiated from desktop workstations and communicate with the server through explicit messages across a local-area or a wide-area network.

The server response is a critical factor in the performance of a client-server database. Resources of the server are shared among all clients, and can become the bottlenecks in scaling the system to large database sizes and many clients. Optimizing the performance of the server has thus been a major focus of commercial systems. However, the revolution in computer hardware technology has made client resources relatively cheap and plentiful. Today's *smart* clients can perform intensive computations locally, using the database as a remote resource that is accessed only when necessary. Therefore, the system design must now take into consideration client memory and CPU for data caching and query evaluation purposes. Despite the potential cost of maintaining data cached at client sites, a number of recent studies for object-oriented databases [3, 4, 11] have demonstrated that increased client-side functionality generally improves the system performance.

A new client-side caching scheme for client-server databases was proposed in [6]. The client cache, henceforth called the *A*Cache*, dynamically loads query results during transaction execution, and uses query predicates to formulate predicate descriptions of the cache contents. A*Cache supports *associative* data reuse across transactions — new queries are compared against the cache description using predicate-based reasoning [7, 8] to determine if the query can be evaluated locally. Predicate descriptions of client caches are also registered at the server, and are used to generate update notifications for cache maintenance. The clients the notifications to maintain the validity of their cache descriptions and data.

The A*Cache scheme provides a number of benefits over ID-based caching for *navigational* purposes, as in [3]. A primary advanantage is that it supports the processing of associative queries locally at client sites, thus improving data reuse. Server indexes are not required for query execution at client sites — the clients may construct local indexes to faciliate query evaluation. The cache maintenance method is completely flexible, e.g., automatic refresh or invalidation upon update, and can vary by client and even by individual queries. Several new optimization and approximation techniques can be designed in this context; a detailed discussion of the design choices appears in [6].

Operating the A*Cache requires reasoning with predicates in a dynamic environment, naturally raising questions about its performance and scalability. Earlier papers have demonstrated the effectiveness of A*Cache in the presence of moderate-to-high update loads [1]. More results on the performance of the A*Cache scheme for different workload types will be presented in the full version of this paper. We present below a brief summary of our scalability results for the A*Cache scheme.

2 Overview of A*Cache Architecture

The persistent data store is resident at the server and transactions are initiated from client sites, with the server providing transactional facilities for shared data access and recovery. The configuration is non-shared memory, so that the address space of each client process is disjoint from that of the server and of other clients. Separate subsystems exist at each client site and at the central server for cache management. In this paper, we assume that a client runs a single transaction at any given time. Thus, local concurrency control and lock management issues are not considered at the client. Figure 1 shows an client-server A*Cache system with one client. Due to space constraints, we omit detailed description of the various components here; details of the system operation can be found in [1, 6].

3 Related Work

A couple of recent studies [2, 10] have examined associative access to a client cache. Both of these studies are related to the associative caching model presented in [6] but are limited to read-only scenarios. The *semantic* caching study in [2] investigates cache replacement policies for no-update workloads. A cache manager called WATCHMAN for read-only caching of query results in data warehousing environments is presented in [10]. Neither of these two studies consider the important issue of cache maintenance when there are database updates, and the performance and scalability of the system under update loads.

4 Simulation Analysis

In this simulation study, we consider a single relation with 100,000 tuples of 200 bytes each in a Wisconsin-style scenario [5]. The relation has two indexed attributes *unique1* and *unique2* that are unclustered and clustered respectively. Each query is a linear range selection either on *unique1* or on *unique2*. The predicates in a cache description are linear intervals corresponding to query ranges, and they are scanned in sequence for checking query containment and for generating update notifications. Figure 2 shows our workload parameters for the simulation. The system paramaters such as CPU speeds of the server and client CPU speeds are omitted due to space constraints — they appear in [1].

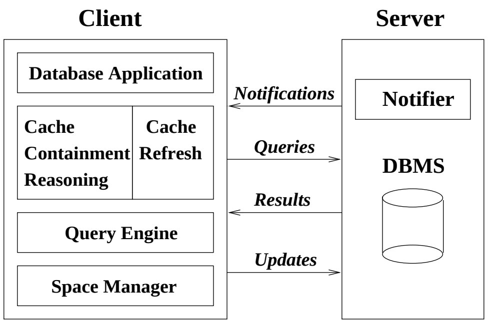

Figure 1: A*Cache System with 1 Client

DBsize	100,000	Database size in tuples
TupleSize	200	Tuple size in bytes
QueryLength	100	Query length in tuples
PrivateRatio	80%	Private region as a % of database size
PrivateAccessProb	50%	Access probability of private region
SharedAccessProb	20%	Access probability of shared part ($= 1 - PrivateAccessProb$)
PrivateWriteProb	20%	% of update transactions in private region
SharedWriteProb	20%	% of update transactions in shared region

Figure 2: Workload Parameters for Simulation

Our workloads model different degrees of shared data contention among clients. The database is logically divided into a shared and a non-shared (i.e., *private*) part. Each client can access data in the shared part and in its own private region. The private region for client i is defined by the linear interval : $[\frac{DBsize*PrivateRatio}{NumClients}*(i-1), \quad \frac{DBsize*PrivateRatio}{NumClients}*i)$. We define data access probabilities *PrivateAccessProb* and *SharedAccessProb*, and update probabilities *PrivateWriteProb* and *SharedWriteProb* for transactions in the private and shared regions of the database respectively. These quantities are the same for each client. Transactions consist of a single associative query or update, with an update transaction reading and writing all tuples that it accesses.

Figure 3 shows the A*Cache performance when the number of clients in the system is scaled up. The number of clients is varied from 20 to 70 while other parameters, such as the buffer size at the server, are held constant. Data access is by the clustered index *Unique2*. Half of the queries here are within the private region of each client and the *UpdateRatio* is 20%. Although A*Cache has a larger number of notifications with more clients, the query response time is better than the no-caching scheme. This result is due to utilization of the CPU power and memory of the additional clients by A*Cache. It shows that A*Cache is highly scalable with respect to the number of clients in the system.

In Figure 4, we varied the query length from 50 tuples (1KByte) to 300 tuples (6KBytes) for a database size of 10,000 tuples and 10 clients. Notifications increase with larger query lengths, but A*Cache performs much better than the no-caching case. The reason for the linear increase in the no-caching query response time is the almost linear increase in execution time at the server

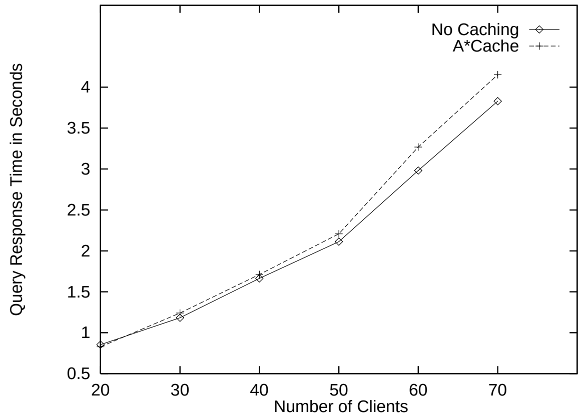

Figure 3: Varying Number of Clients

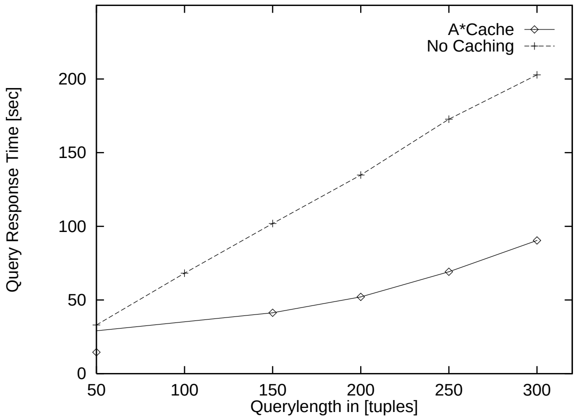

Figure 4: Varying Query Length

with increasing query length, causing the server CPU and buffer utilizations as well as the query response time to increase. In contrast, A*Cache re-uses local data effectively (*CacheHitRatio = 63%*), so that majority of queries are executed at the client and not at the server. Therefore, the query response time for A*Cache increases less than linearly with the query length.

5 Implementation Approaches

To determine the feasibility of implementing the A*Cache scheme with current technology, we examined the issues in developing a prototype system with commercial database server, and a main-memory database (*SmallBase* [12]) serving as the client-side data store with query execution facilities. If the server is assumed to be a "black-box" opaque entity, the notifier must be an "add-on" external module. Although closer integration with the server would yield a more optimal implementation, it is possible to use the data replication facilitites provided by the commercial systems to generate update notifications. For example, in Oracle, updates to a relation can be automatically inserted in a special table called a ***snapshot log*** [9], which can be subsequently examined by a separate notifier process. Likewise, client-side modules such as the cache containment reasoning system and the update handler can be layered on the basic query engine. In brief, extending a client-server system to support client-side A*Caches is very possible with today's technology. Our colleagues, Kurt Shoens and Marie-Anne Neimat, have been implementing such a prototype system.

6 Conclusions

Associative caching is an important technique to improve the performance and scalability of a client-server database by better utilization of client CPU and memory. Our simulation experiments (only a few of which are reported in this abstract) demonstrate that A*Cache provides high performance even for moderately high update loads and under many different contention scenarios. The system is highly scalable with respect to the number of clients and query sizes, and performs consistently better than the no-caching scheme in most cases.

Future work includes the investigation of different cache maintenance policies, such as invalidation of infrequently-used query results and update of frequently-used ones. Using advanced techniques to determine query containment[7, 8] and to generate update notifications are other aspects of future work.

References

[1] J. Basu, M. Poess, and A. M. Keller, "Performance Analysis of Associative Caching in the Presence of Updates", Submitted for Publication, Feb. 1997. Available on the WEB at *http://www-db.stanford.edu/ basu/pub/acache_performance.ps*

[2] S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, and M. Tan, "Semantic Data Caching and Replacement," *22nd Intl. Conference on Very Large Data Bases (VLDB 96)*, Bombay, India, September, 1996.

[3] M.J. Franklin, "Caching and Memory Management in Client-Server Database Systems," *PhD thesis*, University of Wisconsin-Madison, 1993.

[4] M.J. Franklin, B.T. Jonsson, and D. Kossmann, "Performance Tradeoffs for Client-Server Query Processing," *Proc. ACM SIGMOD Conf. on Management of Data*, 1996, Montreal, Canada, pp. 149-160.

[5] Jim Gray, "The Benchmark Handbook for Databases and Transaction Processing Systems,", Morgan-Kaufmann Publishers Inc., 1993.

[6] A.M. Keller and J. Basu, "A Predicate-based Caching Scheme for Client-Server Database Architectures," *The VLDB Journal,* Vol. 5, No. 1, Jan 1996, pp. 35-47.

[7] A.Y. Levy and Y. Sagiv, "Queries Independent of Updates," *19th Int. Conf. on Very Large Data Bases*, Dublin, Ireland, August 1993.

[8] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava, "Answering Queries Using Views," *Proc. PODS Conf.*, 1995.

[9] Oracle Corporation, "Oracle7 Server Replication Manual," 1995.

[10] P. Scheuermann, J. Shim, and R. Vingralek, "WATCHMAN: A Data Warehouse Intelligent Cache Manager," *Proc. VLDB Conf.*, Bombay, India, 1996.

[11] Y. Wang and L.A. Rowe, "Cache Consistency and Concurrency Control in a Client-Server DBMS Architecture," *Proc. ACM SIGMOD Conf. on Management of Data*, Denver, CO, May 1991, pp. 367-376.

[12] M. Heytens, S. Listgarten, M.-A. Neimat, and K. Wilkinson, "SmallBase: A Main-Memory DBMS For High-Performance Applications," Technical Report, Hewlett-Packard Laboratories, Dec 1994.