Data Caching Issues in Client / Server Database Systems

Yma Pinto & P R Rao

Department of Computer Science & Technology, Goa University,Goa –403 206.
yma/rao>@unigoa.ernet.in

With Client/Server Systems becoming the defacto in many dispersed organizations, the current focus in research should also be on improving performance in such systems. The key to performance and scalability of Client/ Server Systems is the exploitation of relatively inexpensive client resources. One such performance improvement could be the use of client disks to store the query results previously obtained from servers for the future use of these clients. This kind of dynamic caching can decrease network traffic and query response time , thus improving throughput. This paper looks at the various issues involved in handling such dynamic caching of query results.

1. Introduction

In recent years, powerful technological and market forces have combined to affect a major shift in the nature of computing and data management. These forces have had a profound effect on the way Data Base Management Systems are designed. The widespread adoption of various Client/Server Architectures have made distributed computing the defacto for many application domains.

A Client/Server System is a distributed software architecture that executes on a network of interconnected desktop machines and shared server machines. Client processes typically execute on desktop machines where they provide responsive interaction to users. Server processes are shared resources providing services to multiple requesting clients. Such distributed processing results in significant though rather complex challenges in performance improvement.

The key to performance and scalability in Client/ Server database systems is the exploitation of the relatively inexpensive resources provided by client machines. Client disks can be used to store data requested from the server across transaction boundaries. This intermediate storage would improve data access times, thereby increasing throughput and performance. This use of client caching is an effective way of exploiting client resources and can be used as an effective strategy for query optimization in Client/Server database systems without sacrificing availability.

In typical commercial relational database systems with client configurations [30], caching aims to avoid disk traffic and is done on server side only, based on buffering of frequently accessed disk blocks or pages. The assumption is that clients are low end workstations whose role is limited to transmission of an SQL query and presentation of the result. With the rapid growth of high end workstations, clients are capable of performing intensive computations on their own, using the server database as a remote resource to be accessed when necessary. Increased local functionality and autonomy can lead to less network traffic, faster response time, lesser server bottlenecks and increased server throughput.

For a centralized database system, selection of alternative query execution strategies is made by measuring costs involved in I/O and operations performed on the system. In a distributed environment transmission costs between sites must also be considered. Some authors [35] feel that transmission costs add a relatively small overhead. This is true if the bandwidth used for communication is very large, which is not the case in geographically dispersed distributed computing environments. a more practical approach would consider communication costs while determining a solution to a query in its query optimization strategy. In some papers [2] the authors feel that minimization of transmission cost is the only important goal.

An efficient way of reducing transmission costs is dynamically caching of a query result at the client site. This can enhance the overall performance of Client/Server database systems especially when the operational data spaces of clients are mostly disjoint. In effect, such a caching of locally pertinent and frequently used data constitutes a form of dynamic data replication, whereby each client dynamically defines it's own data space of interest.

Caching can be divided into two basic types (i)intratransaction caching and (ii) intertransaction caching. The former refers to caching of data within transaction boundaries. It is implemented by purging any cached data items which are not locked by the current active transaction. The latter refers to caching whereby clients are allowed to retain locally cached data across transaction boundaries. Such cached copies are, by default, not protected by regular concurrency control mechanisms. Thus a number of complex issues need to be researched upon to maintain the ACID properties of future transactions on these caches. Further, the effectiveness of caching depends to a large extent on the assumption that there is a significant "locality of access" in the system workload on the client workstations. Page based caching tends to favour a spatial locality of access whereas a predicate based caching scheme is based on a temporal locality of access.

This paper is organized as follows : Section 2 gives a brief survey of the related work in this area, Section 3 deals with the reasons favouring a caching strategy on the client nodes, and the various issues involved in the same. Section 4 looks at a possible architecture for a Semantic Query Caching strategy, and Section 5 concludes the paper.

2. Related Work

The concept of caching intermediate query results for speeding up subsequent query processing has been widely studied. Improved system performance stems from saving (part of) subsequent query computations by utilizing previously cached results. Early applications of this technique can be found in [16, 28, 36, 34, 22, 21, 3]. [16] is one of the earlier works that reused data while evaluating common subexpressions in queries. In [28,34] query results were cached in relational database systems to avoid repeated computations. [34] proposes a view caching scheme that uses the notion of "logical access paths" and "incremental access methods". Simulation results of the above techniques were presented in [14]. For the query results cached on client disks, the server maintained update logs. Each query on the client cache resulted in an explicit refresh request to the server to compute and propagate the differential changes relevant to the cache. Although caching query results on a local workstation can parallelize query processing among clients and reduce network contention, fetching incremental update logs was found to be a bottleneck with increasing updates.

[1] uses quasi copies in an attempt to reduce overheads involved in updating caches to preserve cache consistency. They allow the caches to diverge from the server data in a controlled fashion. In [5] overheads in updates are reduced by detecting cases when updates to a base relation cannot affect a derived relation, and for detecting when a derived relation can be correctly updated using no data other than the derived relation itself and the given update operation.

The ADMS+/- query optimizer [31] uses the ViewCache an the Incremental Update strategies to store and maintain cache consistency. In the ViewCache architecture, a unique tuple identifier is assigned to each tuple of a base relation. The ViewCache contains a collection of tuple identifiers of records that satisfy the query [12]. These ViewCaches have a hierarchical organization that permits sharing of access paths and confines update propagations.

Query evaluation on a set of "derived relations" is studied in [28]. [36, 22] addressed the problem of caching query results to support queries with procedures and functions. [36] uses a predicate based partial indexing scheme for materialized results of procedure valued attributes.

[26] uses page answers and page traces as the smallest indivisible unit in the cache. Their intelligent database caching is used over a predetermined or repetitively reevaluated query set.

[41,42] Proposed client data caching using object ids. [8,9,7] use page based mechanisms of data caching architectures for caching query results at client sites. In these schemes, storage, retrieval and maintenance of cached

objects at client sites are done based on object ids. As a result, associative queries that access database objects using a predicate on an object class are difficult to handle locally. This is an extremely relevant and important issue in caching of query results in database systems.

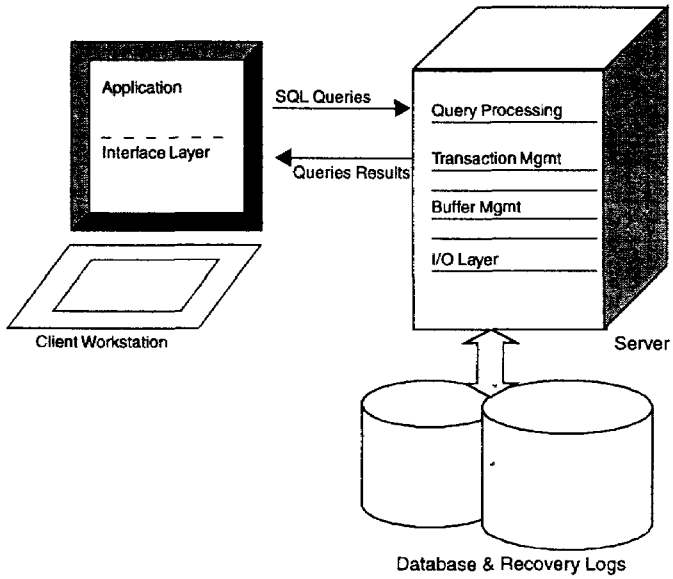
Semantic Query Caching has been proposed as a data caching scheme for client/server environments [13,23,7]. Under this architecture, the clients maintain in their caches, semantic descriptions of the data that is cached along with the associated result of the previously cached query.

3. A Case for Client Caching.

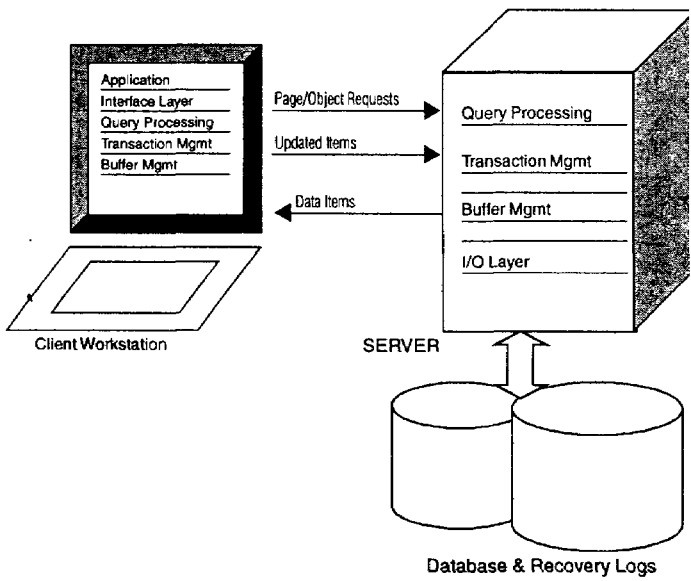
The main reason for the emergence of client/server database systems was to improve the scalability of the system by offloading a major part of the functionality of the server onto the client. Clients interact with the server by means of requests which can be categorized according to the unit of interaction between the client and the server processes. Requests can be in the form of queries(query shipping architectures) or can be requests for specific data items(data shipping architectures).

In query shipping systems, a client specifies a request for the required data items by submitting a query to the server. The server responds by processing the query and transmitting the query results back to the client. In contrast, the data shipping approaches perform the bulk of the query processing on the client workstations, and as a result, much more DBMS functionality is placed on the clients (See Figure 1a, 1b). Data shipping systems can be page_ servers or object_servers. Page servers interact using physical units of data(e.g. pages) and the client is responsible for mapping between pages and the corresponding logical unit(e.g. tuple). In the case of object servers, interaction between client request and server response is by using the logical units of data .The server thus becomes responsible for mapping between pages and objects.

The advantage of query shipping over data shipping is mainly a reduction in communication costs and client space requirements, as only requested data items are transmitted to the clients. Further, query shipping provides an easy migration path from the traditional approach wherein the database query execution engine resided at a central site. Most commercial



(a) A Query Shipping Architecture.



(b) A Data Shipping Approach

Fig.1

Relational DBMS's adopt this approach [24]. In contrast, however, most commercial OODBMS products adopt the data shipping approach. The reasons are improved system scalability as most of the computations are carried out by the client

and server throughput does not decrease dramatically even as more users are augmented. Further, responsiveness is improved by moving data closer to the applications requesting the data [17]. Data shipping approaches also have their drawbacks.

In either of these architectures, if query results are retained across transaction boundaries, they could be used to answer related, follow on queries. Thus, for requests that can be answered from locally cached data, there would be a sizeable decrease in query processing costs, by eliminating or reducing communication costs to the server for data transfer. Client caching could thus be viewed as a combination of Dynamic Replication and Second, Class ownership. Dynamic replication because caches are created, merged, updated and destroyed, based on users requests at client sites. This is in contrast to static replication in which data replication at nodes is done as a part of the physical database design. Second class ownership allows consistency to be preserved without sacrificing availability. This is similar to the concept used in the CODA distributed file system [27]. Client caching requires an optimization between increase in utilization of client resources, decrease in communication costs and increased overheads introduced in cache consistency maintenance and data availability.

The advantages of caching data across transaction boundaries can be summarized as Improvement in overall query response time.

Since part, or all, of the query processing can be done by the client via the cache on the local disk, data will be made available faster. Further, workload at the central server will be reduced, thereby decreasing server bottlenecks.

Communication cost decrease

There is a substantial savings in subsequent data transfer costs, specially if the result sets are large.

Fault tolerance

Sometimes the database server may not be available, thus caching may enable some queries to be handled locally.

Data security

If sensitive data is cached, the number of times such data is shuttled across the network is reduced.

Answer set pipelining

The subset of answers computable at the client by the cache can be returned to the user while the remaining answers are being computed.

Thus, caching is a good way of optimizing query execution as long as efficient query strategies are used. A bad caching architecture can lead to an increase in query execution time, and worse still, system inconsistencies. The increase in query execution time stems from the fact that it is required to be determined whether a particular request is satisfiable by the cache or not. This is the cache completeness or the cache containment issue. Caching also implies an increased complexity in maintaining the consistency of these caches with the server database. This is referred to as the cache coherency problem. Other relevant issues are the manner in which the caches are stored on the local clients, the indexing structures built to enhance query processing, and the maintenance of these caches for further use. [3,33,40,34,4] have proposed various methods for storing the cached data. [37,22] discussed the problem of selective caching and cache replacement. A more recent study that addresses simultaneous issues on cache allocation and replacement has been studied in WATCHMAN[38]. A rudimentary effort at merging caches and their semantic descriptions has been made[13]. In [32,20,6], different strategies for updating cached data are explored. Various protocols and algorithms have been designed to maintain cache consistency [25,41]. [17] has done a taxonomy of protocols by classifying them as Avoidance and Detection based approaches to maintain cache currency.

Aside from the above work which focussed on the problems of cache maintenance and management, the problem of how to identify the useful cached data for computing queries, referred as query matching or query containment problem was addressed in [16,28]. In their work, however, query optimization was not considered inclusively. This is not satisfactory as blindly using cached data in query computations without optimization may result in a query plan even worse than the one optimized from the original query without utilizing any cached result. An effort in this respect has been done in [31,13,18]. When a query is posed at the client site, it is split into two disjoint subqueries – a probe query which retrieves a portion of the result available in the client cache(s) and a remainder query which retrieves the remaining missing tuples from the server. The client caches are considered to be consistent with the database at the server by implementation of a suitable cache consistency protocol.

In more recent works semantic query caching has been proposed in the data caching schemes for client /server environments [13,23]. Under this architecture, the client maintains in its caches, semantic descriptions and

associated answer sets of previous queries. The unit of transfer between servers and clients, is no longer a page, but a set of tuples comprising a query result. The key advantage of semantic query caching is its flexibility. It can be used to answer new queries which are semantically related to previous cached queries. In such architectures, all relational operators can be allowed to be performed on the cache.

The following example illustrates the importance of caching

Example

Assume that a central server has some Tourism Information brochures online containing information on Hotels, besides other relevant information

Hotel(Hotel#, Name,Type,Rates,Location)

(Let Qij denote an ith query at client site Cj)

A user at a client site C1 in Goa may want to know of all hotels in Calangute.

Q11

Select * from Hotel where Location = 'Calangute'

The above information is retrieved from the central server and cached at C1. Any other user wanting information about hotels in Calangute at this site can obtain it readily.

Suppose now a user at C1 wants only luxury hotels in Calangute

Q21

Select * from Hotel where Location ='Calangute' and

Type= 'Luxury'

Since Q21 is a subset of Q11, the query need not be sent to the server but can be answered from the previous locally cached result set.

If Q31 is

Select Name, Rates from Hotel where

Location = 'Calangute' or Location = 'Colva'

Since Q31 can be partially satisfiable in the cache , it is split up into the probe query which is

Select Name, Rates from Hotel where

Location = 'Calangute'

and the Remainder query which is

Select Name, Rates from Hotel
where Location = 'Colva'

which is sent to the server.

On obtaining results from the server of hotels in Colva, the results of the probe and remainder queries are combined and presented to the user application requesting the data. Further, the result set of the remainder query is merged with the previous cached data of Q11. The cache description also changes to reflect this fact. The server also keeps track of the fact that the above result sets are cached at client C1 so as to be able to send notifications to maintain consistency of the caches.

4. A Semantic Query Caching Architecture for Client/Server Database Systems

Before implementing the SQC modules on the clients and the server, it is necessary to devise a model that considers the various aspects of a semantic query caching strategy. The following architecture will serve as a reference for simulating/implementing the Semantic Query Caching scheme for Client/Server database systems.

The Cache Manager intercepts requests of the application program to determine if the query can be processed locally or if it has to be routed to the server. It needs to synchronise with the other client modules to finally present the result to the requesting application program.

The Cache Containment Logic module is invoked when a query needs to be compared with the current cache description, to determine if the query is wholly or partially contained in the cache. Many algorithms have been studied to answer queries from materialized views [43,37,12,11,29]. Such algorithms can be used in this module. If a query is partially contained within the locally cached data, this subset of the results can be obtained in parallel with obtaining the rest of the query from the server.

The trimmed query sent to the server serves to reduce potentially redundant calculations and saves network bandwidth. The cache manager can

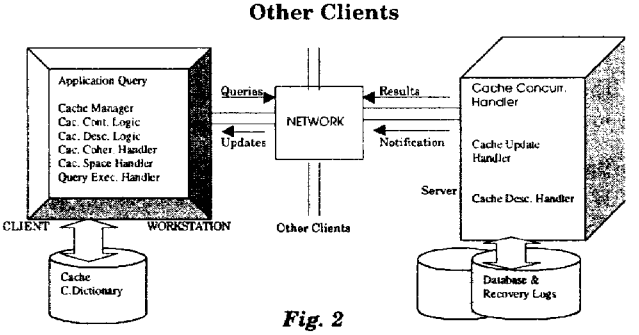


Fig. 2

then synthesize the rsults for presentation to the client.

The Cache Description Handler is a module that keeps track of the predicates that need to be inserted/deleted from the client cache descriptions stored in the Cache Dictionary. Modifications are also possible when predicates are merged and updates are filtered to caches.

The Cache Coherence Handler is responsible for propagation of updates from the server to the corresponding caches. Various cache coherence protocols based on the Avoidance and Detection approaches have been studied in [17]. In the detection class of algorithms, the caches are updated only when there is a cache hit, whereas, in the avoidance protocols, the caches are kept consistent with the server through server notifications. This could involve a greater network traffic. An hybrid algorithm could use an avoidance approach for caches with a high hit ratio and a detection approach otherwise.

The Cache Space Handler besides managing the storage of new query results on the local disk has the added responsibility of implementing the cache replacement policy. The space handler can also build indices and implement relevant clustering techniques for more efficient query evaluation on locally cached data.

The Query Execution Engine needs to have the basic capabilities to retrieve and update the cached data. Commit and Rollback for local updation is also required.

On the Server side, the normal database server needs to be enhanced with additional modules to handle client caches. The additional modules are the Cache Description Handler, Cache Concurrency handler and the Cache Update Handler.

Cache description handler is similar in functionality as that on the client and is required by the server as it is the responsibility of the server to maintain client cache consistency. The Cache Update Handler sends notifications to the relevant clients whenever their cached data needs to be updated. Updates can be done incrementally, by sending only the relevant incremental logs [19]. Updates on caches can also be done synchronously (in case of avoidance protocols) as in the Read Once/Write All protocols [41,15]. The Cache Concurrency handler needs to be enhanced to support queries/ transactions that evaluate the client sites.

5. Conclusion

When caching was initially proposed, main memory was used to store the cached data. For better cost effectiveness, caches can be made to reside on disks. To improve performance to a greater extent, a co-operative caching strategy can be applied to all the client caches. This is a study we propose to do in the near future. A number of query caching strategies are still in their infancy and will be included in our future efforts.

References

1] R. Alonso, D. Barbara and H. Garcia_Molina, "Data Caching Issues in information Retrieval System", ACM TODS, Sept. '90.

2] P.M.G. Apers, A.R. Hevner and S.B. Yao, "Optimization Algorithms for Distributed Queries", IEEE-TSE, SE-91, 1983.

3] A.M.Adiba and B.G. Lindsay , "Database Snapshots", Proceedings. of 6th International Conference on VLDB , 1980.

4] J. Basu ,"Associative caching in Client/Server Databases" , Ph.D. thesis, Stanford University, March '98.

5] J.Blakeley, N. Coburn and P.A.Larson , "Updating Derived Relations: Detection of Irrelevant and Autonomously Computable Updates", ACM-TODS 14, 1989.

6] J.Blakeley, P.A.Larson and F.W.Tompa, "Efficiently Updating Materialized views", Proceedings. of ACM-SIGMOD ,1986.

7] J.Basu, M.Poess and A.Keller, "Performance Evaluation of Centralized and Distributed Index Schemes for Page Server OODBMS", Technical Report, CS Department, Stanford University, March 97.

8] M.Carey, M. Franklin, M.Livny and E.Shekita, "Data Caching Tradeoffs in Client/Server DBMS Achitectures", Proceedings of ACM-SIGMOD, Colorado, May 91.

9] M.Carey, M.Franklin and M.Zaharioudakis ,"Fine Grained Sharing in a Page Server OODBMS", Proceedings of ACM-SIGMOD, Minneapolis, May 94.

10] M.Carey and M.Livny, "Conflict Detection Tradeoffs for Replicated data" ACM-TODS, Dec. '91.

11] S.Chaudhuri , R.Krishnamurthy , S. Potomianos and K. Shim, "Optimizing Queries with Materialized Views", Proceedings of IEEE Conference on Data Engineering, 1995.

12] C.M.Chen and N.Roussopoulus , "Implementation and Performance Evaluation of the ADMS Query Optimizer : Integrating Query Result Caching and Matching", Proceedings of EDBT Conference,1994.

13] S.Dar , M.Franklin , B.Jonsson , D.Srivastava and M.Tan, "Semantic Data Caching and Replacement", Proceedings of International Conference on VLDB, Bombay, 1996.

14] A. Dellis. and N.Roussopoulus , "Evaluation of an Enhanced Workstation Server DBMS architecture", Proceedings of 18th International Conference on VLDB, 1992.

15] M. Franklin and M.Carey, "Client Server Caching Revisited", Proceedings of International Workshop on Distributed Object Management. , Edmonton, Canada, Aug. '92.

16] S.Finkelstein, "Common Expression Analysis in Database Applications", Proceedings of ACM-SIGMOD, 1982.

17] M. J. Franklin, "Caching and Memory Management in Client/Server Database Systems", Ph. D. thesis, University of Wisconsin, Madison, 1993.

18] P. Godfrey and J.Gryz, "Semantic Query Caching for Heterogeneous Databases", KRDB, Greece, 97.

19] A.Gupta , I.S.Mumick and S.Subrahmanian, "Maintaining Views Incrementally", Proceedings of ACM_ SIGMOD, Washington DC, May 93.

20] E.N.Hanson, "Performance analysis of view materializaton strategies" ,Proceedings. of ACM-SIGMOD,1987.

21] J.M.Hellerstein and M.Stonebraker , "Predicate migration: Optimizing queries with expensive predicates", Proceedings of ACM-SIGMOD,1993.

22] A.Jhingram , "A Performances Study of Query Optimization Algorithms on a Database System Supporting Procedures", Proceedings of VLDB '88.

23] A. M. Keller and J.Basu, "A Predicate Based Caching Scheme for Client/Server Database Architecture", VLDB Journal, 5(2): April'96.

24] S. Khoshafian, A.Chan , A.Wong and H.Wang , "A Guide to Developing Client/Server SQL Applications" , Morgan Kaufmann , San Mateo, CA '1992

25] W.Krin, J.Garza , N.Ballou and D.Woelk , "The Architecture of ORION Next-Generation Database System", IEEE-TKDE, March'90

26] K.Kamel. and R. King, "Intelligent Database Caching Through Use of Page Answers and Page Traces", ACM_TODS, Dec 92.

27] J. Kistler, M. Satyanaranan, "Disconnected Operation in the CODA File Systems", Proceedings of 13th International Symposium on Operating System Principles, Pacific Grove, CA Oct '91.

28] P. A. Larson and H.Z.Yang , "Computing Queries from Derived Relations : Theoretical Foundations", Research Report, University of Waterloo, 87.

29] A. Y. Levy, A.O.Mendelzon, Y.Sagiv and D.Srivastava , "Answering Queries Using Views", Proceedings of 14th Symposium on Principles of Database Systems, San Jose,CA, May 95.

30] Oracle 7 Server concepts Manual, Oracle Coorporation, December, 1992.

31] N. Roussopoulos , C. M. Chen and S.Kelly , "The ADMS Project, Views R Us," IEEE Data Engineering Bulletin, June 95.

32] N. Roussopoulos and H.Kang ,"Principles and Techniques in the Design of the ADMS+/-", Computer 19(12):19-25,1986.

33] N. Roussopoulos, "The Logical Access Path Scheme of a Database", IEEE Transactions on Software Engineering, SE-8(6),1982.

34] N. Roussopoulos, "An Incremental Access Method for ViewCache : Concept, Algorithm and Cost Analysis", ACM-TODS 16(3) ,1991.

35] P. G. Sellinger and M.Adiba,"Access Path Selection in Distributed Database Management Systems", Proceedings of 1st International Conference on Databases, Aberdeen, 1980.

36] T. Sellis, "Efficient Supporting Procedures in Relational Database Systems", Proceedings of ACM-SIGMOD, 1987.

37] T. Sellis, "Intelligent Caching and Indexing Techniques for Relational Database Systems", Information Systems '88

[38] P.Schevermann, J. Shim and R.Vingralek, "WATCHMAN: A Data Warehouse Intelligent Cache Manager", Proceedings. of VLDB Conference., Bombay, 96.

39] Y. Expressions with Union and Difference Operators" , Journal of ACM, 1980.

40] P. Valdurez , "Join Indices", ACM_TODS, 12(2), 1987.

41] Y. Wang and L. Rowe, "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture", Proceedings of ACM-SIGMOD International Conference on Management of Data, Denver, 91.

42] K. Wilkinson and M. A. Neimat, "Maintaining Consistency of Client Cached Data", Proceedings of 16th International Conference on VLDB, Australia,1990.

43] H. Z. Yang and P. A. Larson, "Query Transformation for PSJ Queries", Proceedings of VLDB Conference, 1987