EFFICIENT MOBILE CLIENT CACHING SUPPORTING TRANSACTION SEMANTICS

IlYoung Chung and Chong-Sun Hwang
Dept. of Computer Science and Engineering
Korea University
5-1 Anam-dong, Seongbuk-ku
Seoul 136-701, Korea

ABSTRACT

In mobile client-server database systems, caching of frequently accessed data is an important technique that will reduce the contention on the narrow bandwidth wireless channel. As the server in mobile environments may not have any information about the state of its clients' cache(stateless server), using broadcasting approach to transmit the updated data lists to numerous concurrent mobile clients is an attractive approach. In this paper, a caching policy is proposed to maintain cache consistency for mobile computers. The proposed protocol adopts asynchronous(non-periodic) broadcasting as the cache invalidation scheme, and supports transaction semantics in mobile environments. With the asynchronous broadcasting approach, the proposed protocol can improve the throughput by reducing the abortion of transactions with low communication costs. We study the performance of the protocol by means of simulation experiments.

INTRODUCTION

Mobile computing enables people with unrestricted mobility. It can satisfy people's information needs at any time and in any place. Due to the recent development of the hardware such as small portable computers and wireless communication network, data management in mobile computing environments has become an area of increased interest to the database community[Elmagarmid(1997), Badrinath(1992), Pitoura(1995a), Alonso(1993)].
In general, the bandwidth of the wireless channel is rather limited. Thus, caching of frequently accessed data item in a mobile computer can be an effective approach to reducing contention on the narrow bandwidth wireless channel [Pitoura(1995b), Dunham(1995), Imielinsky(1993)].
However, once caching is used, a cache invalidation strategy is required to ensure the cached data in mobile computers are consistent with those stored in the server[Wilkinson(1990), Franklin(1993a), Gottemukkala(1996)].
Several proposals have appeared in the literature regarding the support of transactions in mobile systems[Elmagarmid(1995), Jing(1995), Wong(1995), Narasayya(1993)]. However, most of these approaches didn't attempt to make use of a common feature in wireless systems: the ability that the server has to broadcast information to the mobile clients. Because the server in mobile environments may not have any information about the state of its clients' cache(stateless server), using the broadcasting approach to transmit the updated data lists to numerous concurrent mobile clients is an attractive approach[Barbara(1994)]Considering these limitations of the mobile environments, Barbara and Imielinski proposed an approach that a server periodically broadcasts an invalidation report that reports the data item which have been updated[Barbara(1997), Barbara(1994)]. This approach is attractive in mobile environment because a server need not know the location and the connection status of its clients, and because the clients need not establish an uplink connection to invalidate their cache.
In this paper, we present a protocol that adopts asynchronous(non-periodic) broadcasting, to reduce the wait time of a transaction that has requested commit, and to reduce the abortion of transactions that may show conflicts in the periodic broadcasting strategy. With asynchronous broadcasting approach, the protocol can reduce the number of broadcasting occurrence under high ratio of updates, and can reduce the abortion of transactions under low ratio of updates.
The remainder of this paper is organized as follows. Section 2 introduces the mobile transaction model. In section 3, we describe and discuss our caching protocol. Section 4 presents experiments and results, and finally we state our concluding remarks in section 5.

THE MOBILE TRANSACTION MODEL

Figure 1 presents a general mobile database system model. In this model, both a database server and a database are attached to each fixed host[Wu(1996), Acharya(1995)]. Users of the mobile computers may frequently query databases by invoking a series of operations, generally referred to as a transaction. Serializability is widely accepted as a correctness criterion for execution of transactions[Bernstein(1993), Tannenbaum(1995), Franklin(1994), Berstein(1987)]. This correctness criterion is also adopted in this paper. A database server is intended to support basic transaction operations and as resource allocation, commit, and abort.
Each mobile support station(MSS) has a coordinator which receives transaction operations from mobile hosts and monitors their execution in database servers within the fixed networks. Transaction operations are submitted by a mobile host to the coordinator in its MSS, which in turn sends them to the distributed database servers within the fixed networks for execution[Elmagarmid(1997), Alonso(1993)].

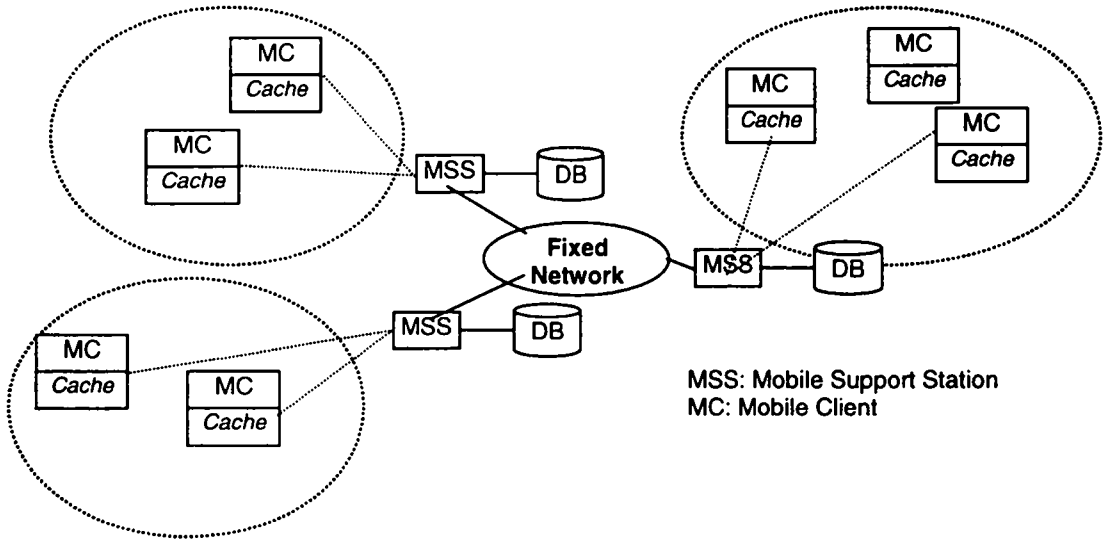

Figure1: The mobile database system architecture

In general, there are two ways to structure a mobile transaction processing system[Narasayya(1993)]; the first situation is that the mobile host behaves like a remote I/O device, and the data must be placed in the static part of the network[Pitoura(1995a), Dunham(1995)]. This situation arises when the amount of resources on the mobile device is small. Secondly, we can consider a mobile host as a full fledged server to manage transaction, and place data locally in mobile hosts[Elmagarmid(1995), Jing(1995), Franklin(1992)].
In this paper, we adopt an way between these two situation to structure a mobile transaction processing system. That is we still keep the data locally, but we treat it as a cache rather than as a primary copy. Thus, as shown in figure 1, each mobile host can have its own cache to maintain some portion of data for later reuse. The transaction operation that accesses the data item stored in the cache can be processed without the interaction with the database server. If the data item does not exist in the cache of the mobile host, it sends an message to the server to download the appropriate data item.

OUR PROTOCOL

In this section, we present our concurrency control protocol, which adopts asynchronous broadcasting approaches, supporting the stateless server. We assume that there is a central server that holds and manages all data. Also, we assume that only one transaction may be initiated by a mobile client at any time. That is, a mobile client can initialize a transaction only after the previous transaction has finished.

Asynchronous Broadcasting

The proposed protocol adopts broadcasting approach to maintain cache consistency, and to control the concurrent transactions. With the broadcasting approach, the mobile client sends the commit request of the transaction after executing all operations, to install the updates in the central database. Then the server decides commit or abort of the requested transactions, and notifies this information to all of the clients in its cell with the broadcasting invalidation reports. The broadcasting strategy does not require high communication costs, nor require the server to maintain additional information about the mobile clients in its cell, thus is attractive in mobile databases[Barbara(1994), Narasayya(1993)].
Some proposals have appeared in the literature regarding the use of the broadcasting for the control of the concurrent transactions, and all of these approaches adopt synchronous(periodic) manner as the way of broadcasting the invalidation reports [Barbara(1997), Barbara(1994)]. In these schemes, the server broadcasts the list of invalidating data items and the list of transactions that will be committed among those which have requested commit during the last period. These schemes present some problems which arise with the synchronous manner of broadcasting approach.

- if two or more conflicting transactions have requested commit during the same period, only one of them can commit and others have to abort.
- the mobile client is blocked on the average for the half of the broadcasting period until it decides commit or abort of the transaction.

In this paper, we adopt asynchronous broadcasting approach as the way of sending the invalidation report. Unlike the schemes using periodic broadcasting, in our scheme, invalidation reports are broadcasted immediately after a commit request arrives. Using the asynchronous broadcasting approach, most transactions that may show conflicts in the same period by synchronous broadcasting, can avoid them, thus the protocol can reduce the abortion rate of transaction processing. Also, the blocking time of the transaction that have sent the commit request can be reduced, as the server immediately broadcasts the invalidation reports which notifies commit or abort.

The Protocol

Our protocol uses a modified version of optimistic control[Liskov(1992), Franklin(1994), Franklin(1993b), Wang(1991)], in order to reduce the communication costs on the wireless network. With optimistic approach, all operations of a transaction can be processed locally using cached data, and at the end of the transaction, the mobile client sends the commit request message to the server. Then the server immediately broadcasts the invalidation report including the data items that should be invalidated, and the mobile client identifier. Receiving invalidation report, the invalidating action preempts the ongoing transaction that shows conflicts with the now-committed transaction. With this approach, the mobile client can early detect the conflicts of a transaction that should be aborted later at the server.
All data items in the system are tagged with a sequence number which uniquely identifies the state of the data. The sequence number of data item is increased by the server, when the data item is updated. The mobile client includes the sequence number of its cached copy of data(if the data item is cache resident) along with the commit request message.
We now summarize our protocol for both the mobile client and the server.

Mobile Client Protocol:

- Whenever a transaction becomes active, the mobile client opens two sets, read-set and write-set. Initially, the state of a transaction is marked as reading. Data item is added to these sets with the sequence numbers, when the transaction requests read or write operation on that data item. The state of the transaction is changed in updating state, when the transaction requests write operation.
- Whenever the mobile client receives an invalidation report, it removes copies of the data items that are found in the invalidating list. And, if any of the following equations becomes true, the transaction of reading state is changed into read-only state, and the transaction of updating transaction is aborted.
- When a transaction of read-only state requests write operation, the mobile client aborts the transaction.
- When a transaction is ready to commit, the mobile client commits the transaction of reading state or read-only state locally. If the state of the transaction is updating, the mobile client sends a commit request message along with the read-set, write-set and the identification number of the mobile client.
- After that, the mobile client listens to the broadcasting invalidation report which satisfies any of the following. If the invalidation report, satisfying above equations, is attached with the identification number of this mobile client, the transaction is committed, otherwise aborted.

Server Protocol:

- Whenever the server receives a commit request from a mobile client, and if sequence numbers of all data items in read-set and write-set are identical with the server's, it broadcasts the invalidation report along with the invalidating list, which is the list of data item in the write-set of the commit request, and identification number of the mobile client. Otherwise, the server just ignores the commit request.

The protocol described above adopts asynchronous broadcasting, thus the server immediately broadcasts invalidation reports, when it receives a commit request from a mobile client. Our protocol has some advantages with the asynchronous approach. At first, when write operations are infrequent at mobile clients, the protocol can reduce the communication costs by broadcasting invalidation reports only when updating transaction occurs. It is unnecessary to send invalidation reports periodically without data items that should be invalidated. On the other hand, when updating transactions occur frequently, the protocol can avoid many aborts, by reducing the conflicts between updating transactions. In synchronous broadcasting approach, when two or more updating transactions are conflicting in a period, only one of them can commit, as invalidation reports are broadcast once for a period. Our protocol can avoid much of these aborts, because mobile clients are informed the list of updated data items immediately. In this case, the increased number of broadcasting is not an additional communication overhead that may degrades the throughput of transaction processing, as the server initiates only one broadcasting for a committing transaction. As shown in the above protocol, no notification is required for an aborted transaction. The server just ignores the commit request with sequence number which has fallen behind the server's number.

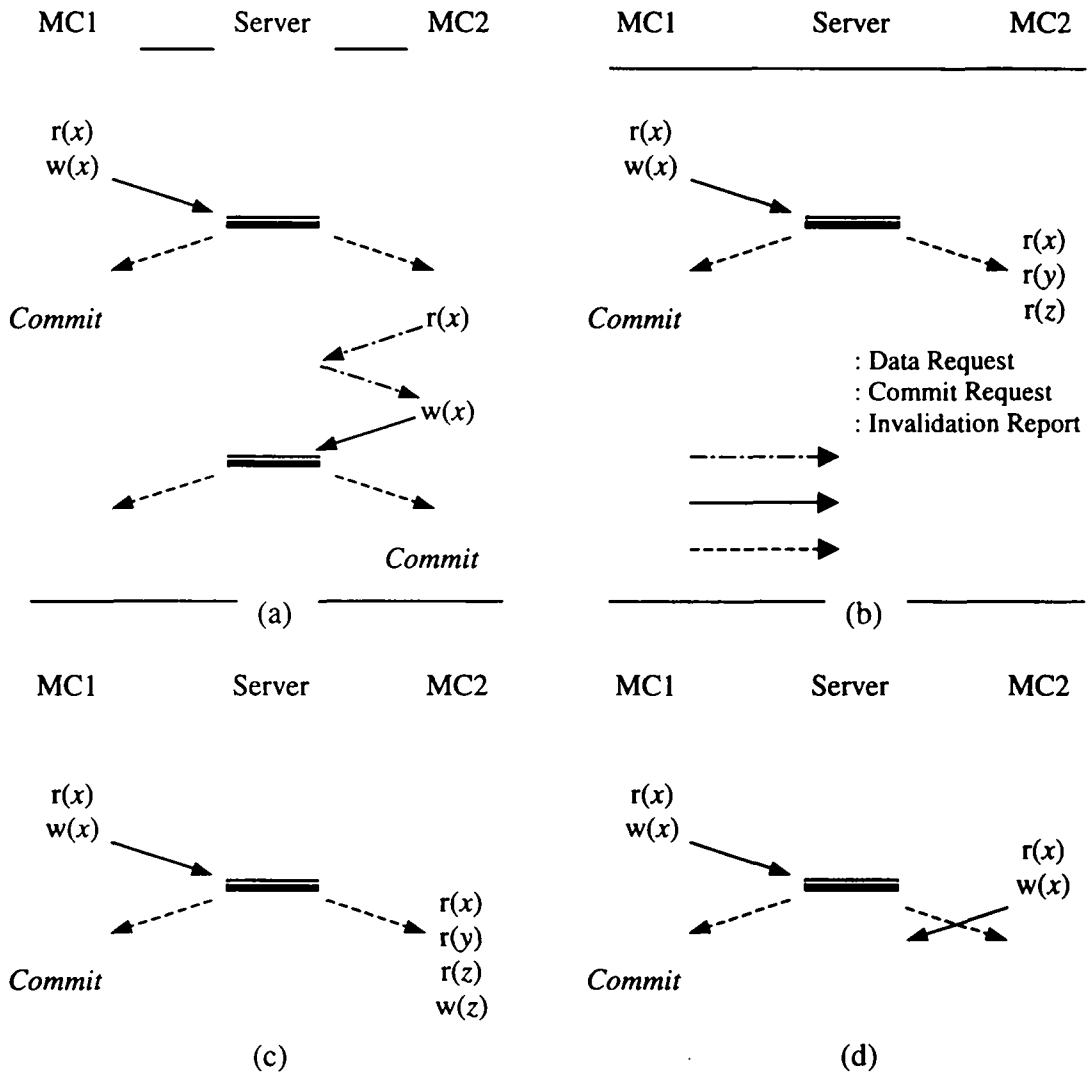

Figure2 : Example execution

Example execution under the protocol is shown in Figure 2. This example uses two mobile clients and a server. In this example, r(x) and w(x) denote a read and write operation performed by a transaction on data item x. In (a), the transaction initiated in mobile client 2 can be committed by the server, as it can read the data item x that has been updated by the transaction in mobile client 1, by the immediate invalidation report. With the synchronous broadcasting approach, if the commit requests of the two transactions arrive in the identical period, both transactions cannot be committed. In (b) and (c), the state of the transaction in mobile client 2 is changed from reading state to read-only state, when the mobile client receives the broadcasting invalidation report. In case of (b), the transaction can commit locally, as the state of the transaction remains read-only state, when the transaction is ready to commit. However, in (c), the transaction will be aborted, because it requests write operation in read-only state. In example (d), the mobile client 2 sends the commit request slightly before it receives the invalidation report that indicates the updates of the data item x. The server ignores the commit request, as the sequence number of data item x in the message is lower than the server's. No broadcasting message is required for the aborted transaction, and the transaction in mobile client 2 is aborted, receiving the invalidation report of data item x, with the identification of mobile client 1. Thus, in our protocol, the server initiates one broadcasting for each commit of transactions.

PERFORMANCE

In this section, we develop the simulation model and present the results of experiments, in order to evaluate the performance of the proposed algorithm. We can divide our simulation model into three parts: database model, transaction model, and system model.

Database Model

Database is composed of multiple data objects which have the same attributes. There are two attributes for each data object: identifier and sequence number. The Database model parameters are summarized in Table 1.

Parameter	Meaning
Nobject	Number of objects in the database
CachePercent	Percentage of cache size

Table 1: Database parameters

Transaction Model

The transaction model supports the following operations.

- *ReadObject*: Reads an object from its cache. If the object does not exist in the cache, reads it from the server database.
- *WriteObject*: Updates an object in the client cache and increase the sequence number.
- *CommitTR*: Commit a transaction.
- *AbortTR*: Abort a transaction.

A transaction is modeled by a finite loop of *ReadObject* and *WriteObject* operations, which are followed by *CommitTR* or *AbortTR* operation. Table 2 summarizes the parameters that characterize a transaction type. The number of *ReadObject* and *WriteObject* operations in a transaction is called *TRSize*, which is uniformly distributed between *MinTRSize* and *MaxTRSize*. The parameter *ProbWrite* indicates the probability that *WriteObject* operations occur in a transaction. The delay parameters are exponentially distributed delay times used to model interactive system.

Parameter	Meaning
TRSizet	Number of operations in a transaction
PROBWrite	Probability that write operation occurs
ReadDelay	Average delay of a read operation
WriteDelay	Average delay of a write operation
TRState	State of transactions that are processed

Table2: Transaction parameters

System Model

The system model consists of a network manager and clients and server modules. The parameter for all the modules are summarized in Table 3.

Parameter	Meaning
NClient	Number of mobile clients
NetDelay	Average communication delay on the wireless network
DBAccessDelay	Average delay to access database
CacheAccessDelay	Average delay to access cache
ReadHitProb	Probability of read hit at mobile client

Table3: System parameters

Experiments

In this section, we present results from several performance experiments involving the protocol described in section 3. The main performance metric presented is system throughput, measured in committed transactions per second. The throughput results are, of course, dependent on the particular settings chosen for the various physical system resource parameters. Thus while the throughput results show performance characteristics in what consider to be a reasonable environment, we also present other performance measures, the number of messages and the percentage of aborts, to provide additional insights into the fundamental tradeoffs between protocols. Table 4 shows the values of the simulation parameters used for these experiments.

Parameter	Value
Nobject	1,000
CachePercent	5%
TRSize(Max)	15
TRSIZE(Min)	3
ProbWrite	0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25 %
ReadDelay	0.01 sec
WriteDelay	0.04 sec
Nclient	20
NetDelay	0.2 sec
DBAccessDelay	0.05 sec
CacheAccessDelay	0.01 sec
ReadHitProb	0.5 sec

Table4: Simulation parameter settings

Results

In this section, we present the results from several performance experiments. We ran a number of simulations to compare the behavior of the proposed protocol and the protocol with periodic broadcasting. Figure 3 shows the required number of messages to commit a transaction with the proposed protocol and the synchronous one. The message counts of each protocol increases in a sublinear fashion in whole range of update ratio. When the update ratio is lower than about 13%, our protocol using asynchronous broadcasting approach requires less messages than the synchronous protocol, because broadcasting is rare with low ratio of updating transactions. If the period is longer than 13%, the frequently broadcasted invalidation reports are the main factor that cause our protocol to require more messages. In the synchronous protocol, the message costs does not increase so rapidly as our protocol, because the invalidation reports are broadcasted periodically. Figure3: Impact of updating ratio on message counts

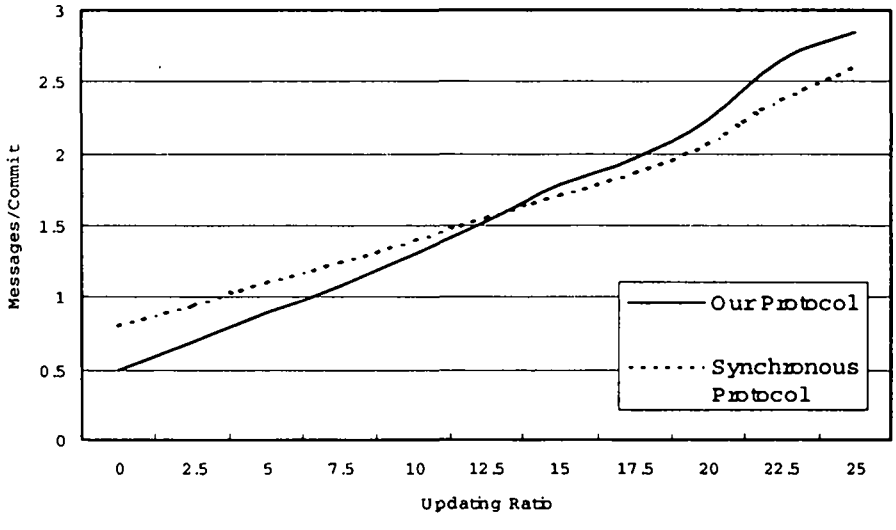

Figure3: Impact of updating ratio on message counts

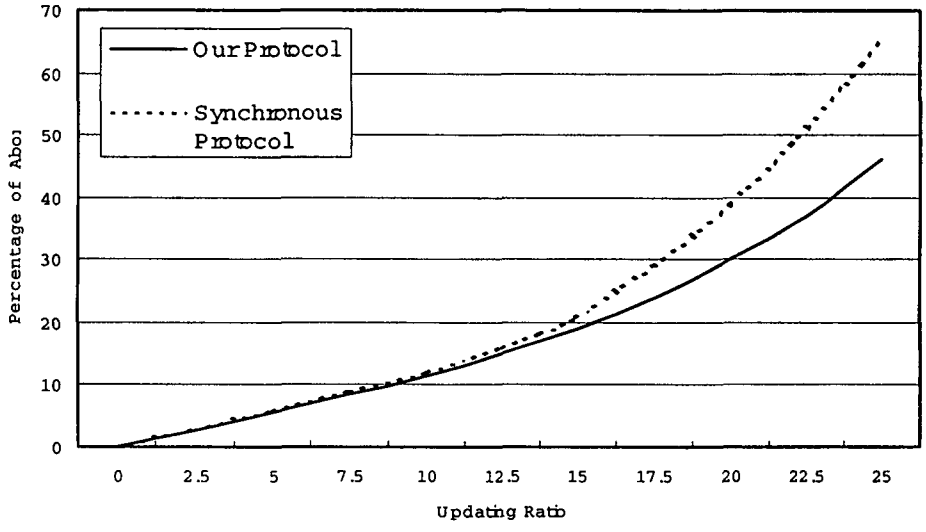

Figure4: Impact of updating ratio on aborts

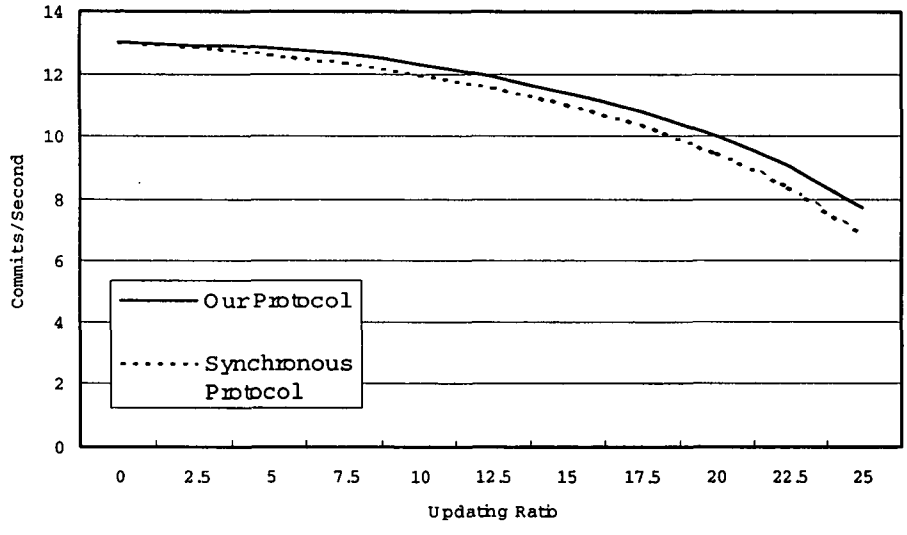

Figure5: Impact of updating ratio on throughput

Figure 4 presents the percentage of aborts of transaction that take place both at the client side and at the server. As can be seen, when updating ratio increases, the percentage gets to be significant. When the updating ratio is lower, most aborts are due to the transaction's read operation on the stale cached data. It occurs when an invalidation report is not delivered by disconnection, or when a mobile client requests commit of a transaction before it receives the invalidation report which include the data item referenced by the transaction. As the updating ratio gets higher, Aborts increase with both protocols, mainly because of the increased conflicts between transactions. In case of the synchronous protocol, the percentage increases more rapidly, as the number of transactions that show conflicts in the same broadcasting period increases.

Figure 5 shows the throughput results for two protocols. As shown in this figure, the throughput degrades with increasing write operations, because of the message costs for updating transactions. When the updating ratio is very low, two protocols show almost the same performance, because there are few transactions that are aborted by the server, in both cases. However, the throughput of the synchronous protocol degrades more rapidly as the updating ratio increases, because of increasing aborts of transactions. With synchronous approach, lots of aborts caused by conflicts may happen, as the server checks the conflicts between transactions once in a period. In our protocol, most of such aborts can be avoided by broadcasting invalidation reports immediately after receiving a commit request.

CONCLUSIONS

We have presented a new protocol to support transactions in mobile client-server environments. The proposed protocol adopts broadcasting cache invalidation strategy to support the stateless server scheme, a common feature in mobile environments. In this paper, we adopt asynchronous approach as the way of broadcasting invalidation reports. With this approach, our protocol can reduce communication costs when updating is rare, and can avoid most aborts of adjacent conflicting transactions without additional communications on the wireless network. Simulations were conducted to evaluate the performance of the protocol. Our performance experiments show that relative merit of the proposed protocol under various updating ratio.

REFERENCES

Acharya S.and R. Alonso (1995) "The Computational Requirements of Mobile Machines," **Proc. of the International Conference of Engineering of Complex Computer Systems**

Alonso R. and H. Korth (1993) "Database System Issues in Nomadic Computing," **Proc. of the ACM SIGMOD Conference on Management of Data**

Badrinath B. and T. Imielinsky (1992) "Replication and Mobility," **Workshop on the Management of Replicated Data**

Barbara D. (1997) "Certification Reports: Supporting Transactions in Wireless Systems," **Proc. of IEEE International Conference on Distributed Computing**

Barbara D. and T. Imielinsky (1994) "Sleepers and Workaholics: Caching in Mobile Environments," **Proc. of ACM SIGMOD Conference on Management of Data**

Bernstein A.J. and P/ M. Lewis (1993) **Concurrency in Programming and Database Systems**, Jones and Bartlett Publishers

Berstein P.A. V. Hadzilacos and N. Goodman (1987) **Concurrency Control and Recovery in Database Systems** Addison-Wesley

Chu S.I. and M. Winslett (1994) "Minipage Locking Support for Object-Oriented Page-Server DBMS," **Proc. International Conference on Information and Knowledge Management**

Dunham M.H. and A. S. Helal (1995) "Mobile Computing and Databases: Anything New?," **ACM SIGMOD Record**

Elmagarmid, A., J. Jing and O. Bukhres (1995) "An Efficient and Reliable Reservation Algorithm for Mobile Transactions," **Proc. of International Conference on Information and Knowledge Management**

Elmagarmid, A., J. Jing and T. Furukawa (1997) "Wireless Client/Server Computing for Personal Information Services and Applications," **ACM SIGMOD Record**

Franklin M.J.(1993a) Caching and Memory Management in Client-Server Database Systems **Ph.d. Thesis, Dept. of Computer Science, University of Wisconsin**

Franklin M.J.and M. J. Carey (1992) "Client-Server Caching Revisited," **Proc. of the International Workshop on Distributed Object Management** (Published as Distributed Object Management, Ozsu, Dayal, Valduriez, Morgan Kaufmann, 1994)

Franklin M.J., M. J. Carey and M. Livny (1992) "Global Memory Management in Client-Server DBMS Architecture," **Proc. of the International Conference on the Very Large Data Bases**

Franklin M.J., M. J. Carey and M. Livny (1993b) "Local Disk Caching for Client-Server Database Systems," **Proc. of the International Conference on Very Large Data Bases**

Gottemukkala V., E. Omiecinski and U. Ramachandran (1996) "Relaxed Consistency for a Client-Server Database," **Proc. of International Conference on Data Engineering**

Imielinsky T. and B. R. Badrinath (1993) "Data Management for Mobile Computing," **ACM SIGMOD Record** V. J. Jing O. Bukhres and A. Elmagarmid (1995) "Distributed Lock Management for Mobile Transactions," **Proc. of International Conference on Distributed Computing systems**

Liskov B., M. Day and L. Shrira (1992) "Distributed Object Management in Thor," **Proc. of the International Workshop on Distributed Object Management**, (Published as Distributed Object Management, Ozsu, Dayal, Valduriez, Morgan Kaufmann, 1994)

Narasayya R. (1993) "Distributed Transactions in a Mobile Computing System," **Proc. of Interbational Conference on Parallel and Distributed Systems**

Pitoura E. and B. Bhargava (1994) "Building Information Systems for Mobile Environments," **Proc. of International Conference on Information and Knowledge Management**

Pitoura E. and B. Bhargava (1995) "Maintaining Consistency of Data in Mobile Distributed Environments," **Proc. of International Conference on Distributed Computing Systems**

Tannenbaum A.S. (1995) **Distributed Operating Systems**, Prentice Hall

Wang Y. and L. Rowe (1991) "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture," **Proc. of the ACM SIGMOD International Conference on the Management of Data**

Wilkinson K. and M. Neimat (1990) "Maintaining Consistency of Client Cached Data," **Proc. of the International Conference on Very Large Data Bases**

Wong M.H. and W. M. Leung (1995) "A Caching Policy to Support Read-only Transactions in a Mobile Computing Environment," **Technical Report, The Chinese Univ. of Hong Kong, Dept. of Computer Science**

Wu K.L., P. S. Yu and M. S. Chen (1996) "Energy-efficient Caching for Wireless Mobile Computing," **Proc. of the International Conference on Data Engineering**