JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 18, 445-457 (2002)

Short Paper

Adaptive Transaction Management Protocols for Mobile Client Caching DBMSs*

ILYOUNG CHUNG, LE GRUENWALD*, CHONG-SUN HWANG** AND SOON YOUNG JUNG**+
Department of Computer Science
Purdue University
West Lafayette, IN 47097, U.S.A.
E-mail: iychung@cs.purdue.edu
**School of Computer Science*
University of Oklahoma
E-mail: ggruenwald@ou.edu
***Department of Computer Science and Engineering*
Korea University
SeongBuk-Gu, Seoul 136-701, Korea
E-mail: hwang@disys.korea.ac.kr
****+Department of Computer Science Education*
Korea University
SeongBuk-Gu, Seoul 136-701, Korea
E-mail: jsy@comedu.korea.ac.kr

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), the use of a *broadcast approach* to transmit the list of updated data to numerous concurrent mobile clients is an attractive approach. In this paper, two caching methods are proposed to support transaction semantics at mobile clients. Proposed protocols adopt *asynchronous broadcasting* and *hybrid broadcasting* as the way of sending control messages, in order to dynamically adapt to system workload (update pattern, data locality). We study the performance of the proposed protocols by means of simulation experiments.

Keywords: mobile computing, client-server databases, cache consistency, broadcasting, concurrency control

1. INTRODUCTION

Mobile computing provides people with unrestricted mobility. It can satisfy people's

Received November 24, 1999; revised June 20 & December 28, 2000 & March 2, 2001;
accepted April 25, 2001,
Communicated by Arbee L. P. Chen.
*This work was supported by the Post-doctoral Fellowship Program of Korea Science and Engineering Foundation(KOSEF).

information needs *at any time and in any place*. Due to the recent development of 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 [1].

In general, the bandwidth of the wireless channel is rather limited [1, 2]. Thus, *caching* of frequently accessed data in a mobile computer can be an effective approach to reducing contention on the narrow bandwidth wireless channel [3, 4]. However, once caching is used, a cache invalidation strategy is required to ensure that cached data in mobile computers are consistent with that stored in the server [4, 5].

In mobile database model, both a database server and a database are attached to each fixed host. Users of the mobile computers may frequently query databases by invoking a series of operations, generally referred to as transactions. *Serializability* is widely accepted as a correctness criterion for execution of transactions [5-8]. This correctness criterion is also adopted in this paper. The database server is intended to support basic transaction operations such as resource allocation, commit, and abort.

Each mobile support station (MSS) has a coordinator which receives transaction operations from the 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 [2, 9].

Several proposals have appeared in the literature regarding the support of transactions in mobile systems [7, 8, 10]. And *broadcasting* is an approach that is widely accepted to maintain cache consistency, and to control the concurrent transactions in mobile environments [2, 4]. With the broadcasting approach, a mobile client sends a commit request message of a transaction after executing all operations to install the updates in the central database. Then the server decides to commit or abort the requested transaction, and notifies all clients in its cell with broadcasting control messages. The broadcasting strategy does not require high communication costs, nor require the server to maintain additional information about mobile clients in its cell, thus is attractive in mobile databases.

All approaches using broadcasting strategy adopt *synchronous* (*periodic*) methods to broadcast those messages [7]. In these approaches, the server broadcasts the list of updated data items and the list of transactions that will be committed among those which have requested a commit during the last period. These approaches present some problems due to the synchronous manner of broadcasting. In this paper we present two broadcasting strategies in order to resolve the problems of the synchronous approach. Proposed protocols can dynamically adapt to the update pattern and data locality of each mobile client.

First, we present a protocol that adopts *asynchronous* (*aperiodic*) *broadcasting*, to reduce the waiting time of a transaction that has requested commit, and to reduce aborts of transactions that may show conflicts in the same period by synchronous broadcasting. With asynchronous broadcasting approach, the protocol can reduce the number of broadcasting occurrence under low probability of updates, and can reduce aborts of transactions under high probability of updates. Second, we introduce another adaptive protocol which is a *hybrid* of the synchronous and asynchronous algorithm, in order to dynamically adjust the broadcasting method according to the system workload.

The remainder of the paper is organized as follows. In section 2 and section 3 we

describe and discuss the asynchronous broadcasting protocol, and hybrid broadcasting protocol. Section 4 presents experiments and results, and finally we state our concluding remarks in section 5.

2. ASYNCHRONOUS BROADCASTING PROTOCOL

In order to reduce aborts of transactions, and in order to adjust the frequency of the broadcasting occurrence, we present two concurrency control protocols, which adopt *asynchronous broadcasting* and *hybrid broadcasting*. In this section, we present the first protocol, *Asynchronous Broadcasting Protocol (ASBP)*, which can adapt to the update probability.

2.1 Asynchronous Broadcasting

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) method as the way of broadcasting control messages [3, 7]. In these schemes, the server broadcasts the list of data items that should be invalidated, and the list of transactions that will be committed among those which have requested commit during the last period [7]. These schemes present some problems due to the synchronous manner of broadcasting approach.

- if two or more conflicting transactions have requested a 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 to commit or abort the transaction.

In this section, we adopt the *asynchronous broadcasting approach* as the way of sending control messages. Unlike schemes using periodic broadcasting, in our scheme, those messages are broadcast *immediately after a commit request arrives*. Using the asynchronous broadcasting approach, most aborts due to conflicts within the same period by synchronous broadcasting can be avoided, thus the protocol can reduce the abortion rate of transaction processing. Also, the waiting time of a transaction that has sent a commit request can be reduced, as the server immediately broadcasts messages which notifies commit or abort.

2.2 The Protocol

Our protocol uses a modified version of optimistic control [5, 6] in order to reduce the communication costs on the wireless network. With the 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 *comm_request* message to the server. Then the server *immediately* broadcasts the *comm_notification* including data items that should be invalidated, and the mobile client identifier. Receiving *comm_notification*, the invalidating action *preempts* the ongoing transaction, thus transactions which are concurrently processed locally and show conflicts with the committing transaction should be aborted.

With this approach, the mobile client can detect the conflicts of a transaction earlier without interaction with the server [7, 10, 11].

All data items in the system are tagged with a sequence number which uniquely identifies the state of the data. The sequence number of a data item is increased by the server, when an updating transaction which writes on the data is committed. All data items updated by a transaction are given the same sequence number. 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. Before describing the algorithms used at mobile clients, information which is created at mobile clients are presented.

***read_set* :** List of data items which are read by a transaction. Data items in this list is attached with sequence numbers that were maintained at mobile clients.

***write_set* :** List of data items which are written by a transaction. Data items in this list is attached with sequence numbers that were maintained at mobile clients.

***sequence_no* :** Sequence number of a transaction. It is determined when the first operation of the transaction accesses requested data item.

***commit_request* :** A message which is sent to the server when a transaction T_i is completed. This message is attached with the identification of T_i, *read_set* and *write_set* which is read and written by T_i.

We now summarize our protocol for mobile clients. In the transaction processing protocols in mobile environments, the dependency upon the server that is caused by correctness checking of transactions can be further considered. If mobile clients can decide commit or abort of a locally executed transaction only with information broadcasted from the server (without uplink message), the overall pathlength of transactions can be significantly shortened, and the throughput can be improved through the offloading of the wireless network. Although transactions which updated data items should be sent to the server because of the update installation, special consideration can be given to read-only transactions. As a result, the proposed protocol can increase autonomy of mobile clients which can be gained at the correctness checking of read-only transactions.

Mobile Client Protocol:

- Whenever a transaction becomes active, the mobile client opens two sets, *read_set* and *write_set*. When the transaction requests a read or write operation on data item *x*, it is added to those sets along with the sequence number of this data item. The *sequence_no* of the transaction is set to the number of the data item which is accessed by the first operation of the transaction. Initially, the state of a transaction is marked as *reading*. The 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 into *updating* state when the transaction requests a write operation.
- Whenever the mobile client receives a *commit_notification*, it removes copies of the data items that are found in the *invalidating_list*. And, if any of data items in the *in-*

validation list is found in *read_set* or *write_set*, the transaction of *reading* state is changed into *read-only* state, and the transaction of *updating* state is aborted.

- When a transaction of *read-only* state requests any write operation, the mobile client aborts the transaction.
- When a transaction is ready to commit, and if the state of the transaction is *reading* or *read-only*, the mobile client checks the sequence numbers of data items in *read_set* and *write_set*. If all of them are less than or equal to the *sequence_no* of the transaction, the mobile client commits the transaction, otherwise aborts it. 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 transaction.
- After that, the mobile client listens to the broadcasting *commit_notification*. If any of data items in the *invalidation list* is found in *read_set* or *write_set*, and is attached with the identification number of its own transaction, the mobile client commits the transaction. Otherwise, the mobile client aborts the transaction, and removes copies of the data items that are found in *invalidating_list*.

When a mobile client begins a transaction T_i, the initial state of T_i is *reading*, and is changed into *updating*, if T_i requests any write operation in mobile client. As shown in the mobile client protocol, completing all operations (when T_i is ready to commit), the mobile client sends *commit_request* message, only when the state of T_i is *updating*. When transactions are *reading* or *read-only*, the mobile client decides commit or abort autonomously without interaction with the server.

When T_i requests no write operation, the decision of commit or abort of T_i is done locally by the mobile client, using the *commit_notifications* broadcasted asynchronously by the server. In this case, the state of T_i is changed into *read-only*, if the mobile client is informed about the commit of a conflicting transaction T_j by a *commit_notification*, and T_i precedes T_j in serialization order. The mobile client aborts the T_i, if T_i requests any write operation during *read-only* state, because such write operations can produce a conflict between T_j and T_i, which is unacceptable for serializable execution.

When T_i of *reading* or *read-only* is ready to commit, the mobile client checks sequence numbers of all data items accessed by the transaction, and compares them with the *sequence_no* of T_i. If any of them is larger than *sequence_no*, the mobile client aborts T_i, because serializability cannot be guaranteed in this case.

We illustrate this mobile client protocol using the state diagram as shown in Fig. 1. Unlike transactions of *reading* or *read-only* state, the decision of committing a transaction of *updating* state is done by the server, as the mobile client cannot determine the serialization order of the transaction autonomously if the transaction has executed any write operation. Thus, when an *updating* transaction is ready to commit, the mobile client sends a *commit_request* message to the server, as shown in Fig. 1. In order to determine whether to commit or abort of an *updating* transaction in the server, all data items in the system are tagged with a sequence number as shown in the mobile client algorithm.

The server continuously performs the following algorithm.

Server Protocol:

- Whenever the server receives a *commit_request* from a mobile client, it compares the sequence numbers of all data items in *read_set* and *write_set* with the server's.

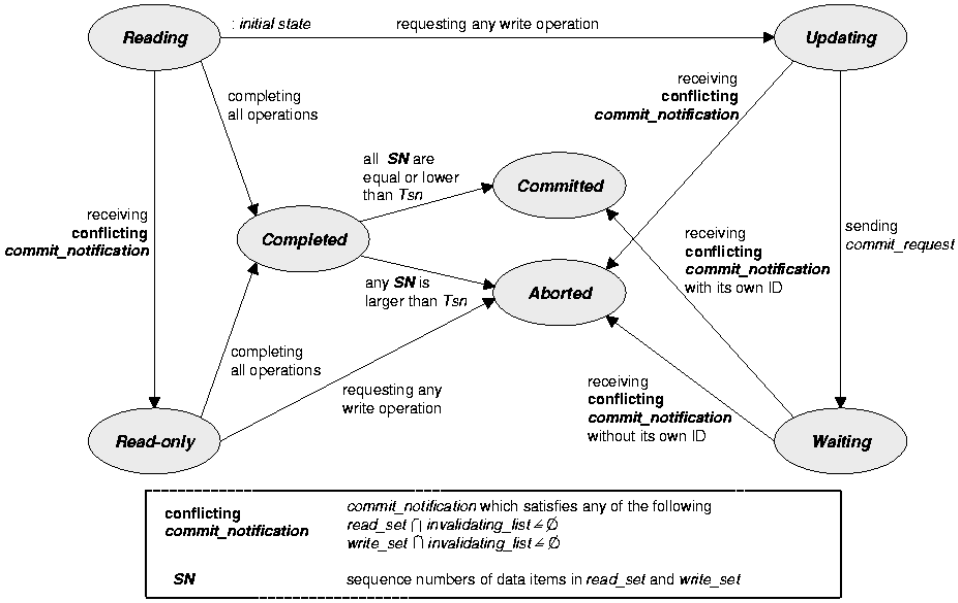

Fig. 1. States of a transaction in the mobile client.

If all of them are identical to the server's, take the following steps:
– put the identification of the transaction to the *commit_notification*.
– put the *invalidating_list*, which is the list of data items in the *write_set* of the *commit_request*, to the *commit_notification*.
– install the values of data items which have been updated by the transaction in the server, and give them unique sequence number which is increased sequentially by the server.
– broadcast the *commit_notification*.

Otherwise, the server just ignores the *commit_request*.

The server decides commit or abort of an *updating* transaction, by comparing sequence numbers of data items in *read_set* and *write_set* with those maintained in the server. If all of them are consistent with the server's, the server can conclude that the transaction has executed its operations after receiving *commit_notifications* of conflicting transactions which precede in serialization order. Thus the server decides to commit the transaction. On the other hand, if the sequence number of any data item in *read_set* and *write_set* is lower than the server's, the server cannot commit the transaction, as the mobile client executed the transaction without knowing the result of conflicting transactions which precede in serialization order.

The protocol described above adopts asynchronous broadcasting, thus the server immediately broadcasts *commit_notifications*, when it receives a *commit_request* from a mobile client. Our protocol has some advantages with this asynchronous approach. First, when write operations are infrequent at mobile clients, the protocol can *reduce the com-*

munication costs by broadcasting *commit_notifications* only when updating transaction occurs. It is unnecessary to send *commit_notifications* 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. With synchronous broadcasting approach, when two or more updating transactions are conflicting within the same period, only one of them can commit, as *commit_notifications* are broadcast once for a period. Our protocol can avoid most of these aborts, because mobile clients are sent the list of updated data items immediately.

3. HYBRID BROADCASTING PROTOCOL

In this section, we present another protocol which integrates two broadcasting approaches, synchronous broadcasting and asynchronous broadcasting.

3.1 Hybrid Broadcasting

ASBP, proposed in section 2, may resolve these problems with the asynchronous broadcasting which sends *commit_notifications* immediately after receiving *commit_requests*. Thus, ASBP can dynamically adjust the frequency of the broadcasting occurrence according to the update pattern of transactions. However, for data items that are used by a specific mobile client exclusively, applying an asynchronous approach is quite wasteful. In this case, it is sufficient to send the updated data items periodically, along with the list of data items updated during the last period.

Thus synchronous protocol can be advantageous if a high degree of locality is shown on accessed data by mobile clients. When a data item is updated by a transaction, and if there is no other mobile clients caching this data, a synchronous broadcasting strategy is advantageous, because delayed *commit_notification* of data items does not cause aborts of other transactions, and it dose not seriously degrades the throughput of transaction processing. So, it is unnecessary to send *commit_notifications* separately, wasting the broadcasting bandwidth of the wireless channel.

On the other hand, when mobile clients do not show such locality on data items that are accessed, an asynchronous broadcasting approach is more attractive, since the immediate invalidation of the updated data items can prevent a large portion of aborts of other transactions accessing local copies of these data items.

Considering the tradeoff, integrating these two broadcasting strategies (synchronous and asynchronous broadcasting) yields another adaptive algorithm, which we call *hybrid broadcasting protocol (HBP)*. In HBP, the server dynamically adopts appropriate broadcasting strategy between synchronous broadcasting and asynchronous broadcasting according to the locality of data items which have been updated. With HBP, when the server receives *commit_request* from a mobile client, it checks if any data item which has been updated by the transaction is widely shared by many clients' cache or not.

If all updated data items are cached by few mobile clients, the server just put these items to the *invalidating_list*. Immediate broadcasting of data items which are cached by few clients or no other client except the updating one is quite wasteful. Thus, in this case, HBP just adds those data items to the *invalidating_list* that will be broadcast periodically. On the other hand, when the server receives a *commit_request* of a transaction which has

updated widely shared data items, HBP applies the asynchronous approach to send the *commit_notifications*. So, in this case, the server immediately sends data items which have been updated, along with the *invalidating_list* produced after the last broadcasting. Thus, with HBP, *commit_notifications* are broadcast in one of the following two situations:

- when a *commit_request* contains widely shared data items in its *write_set*.
- when it is time to send periodic *commit_notifications*. (unless the *invalidating_list* is empty)

3.2 The Protocol

Now, we need to explain how to apply the appropriate broadcasting approach according to data items which have been updated by transactions. Because ASBP itself is adaptive to the change of update frequency, as we mentioned in section 3, HBP should control the broadcasting mechanism based on the *locality* of cached copies that should be invalidated. For this we define the *sharing state* for each data item, in order to classify them. Based on this sharing state, the server adopts an appropriate broadcasting approach for each updated data item.

As the server in mobile computing environment does not know about the state of the client's cache (stateless server) [3], the sharing state should be estimated with some other criteria. In HBP we propose *the number of data requests* for a unit of time as the criteria for the sharing state. Once a data item is updated by a transaction, every cached copy of the data item maintained by each mobile client is invalidated. If a mobile client is to access updated value of the data item, it should send a data request message to the server. Thus, frequent requests on a data item implies that it has been updated recently, and that many copies exist in mobile clients' cache.

Based on this criteria, all data items are assigned one of the following two classes by the server. The sharing state should be managed dynamically as the access pattern or locality of data changes.

- *shared* : data items that are cached by many clients
- *exclusive* : data items that are cached by few or no clients

Now, we need to summarize algorithms of HBP, which utilizes the sharing state of data items. The algorithms running on mobile clients are identical with ASBP, and the server performs the following protocol:

- Whenever the server receives a *commit_request* from a mobile client, it checks the sequence numbers of all data items in *read_set* and *write_set*.
  - If they are identical with the server's, and if all data items in *write_set* are marked as *exclusive* state, the server commits the transaction. The server adds updated data items to *invalidating_list*, and gives them the same sequence number. The identification of the transaction is also attached to the *commit_notification*.
  - If they are identical with the server's, and if there is any data item in *write_set* which is marked as *shared* state, the server commits the transaction, adds the up-

dated data items to the *invalidating_list*, and gives them the sequence number. Then the server sends the *commit_notification* containing *invalidating_list* and transaction identifiers immediately.
– If sequence numbers have fallen behind the server's, the server just ignores the *commit_request*.

- When it is time to broadcast periodic *commit_notification*, the server checks if the *invalidating_list* is empty. If the list is not empty, the server broadcasts *commit_notification*.

4. PERFORMANCE

We ran a number of simulations to compare the behavior of the two proposed protocols, ASBP and HBP, and a protocol with synchronous broadcasting. In this section, we present the results from these performance experiments. We performed experiments under the following conditions: (1) when a low portion of data items are shared by mobile clients, (2) when a high portion of them are shared. In order to provide such experimental environments, we set the system parameter *SharedDegree* to 10% and 40%. This implies that 10% or 40% of entire data items are classified as *shared* state, while the rest are classified as *exclusive* state.

First, Figs. 2 and 3 show the average number of transactions that should be aborted with proposed protocols and the synchronous protocol. As can be seen, more transactions are aborted with increasing write operations, in both Figs. 2 and 3 where *SharedDegree* is low and high, respectively. In Fig. 2 (*SharedDegree* = 10%), increased write operations do not cause frequent conflicts between updating transactions, because most data items are accessed exclusively by each mobile client. Thus, in this case, the protocol with synchronous broadcasting does not suffer a high ratio of aborts. On the other hand, in Fig. 3 (*SharedDegree* = 40%), the number of aborts is more dependent on the probability of write operations. Thus, as the probability of write operations increases, aborts rise significantly with all three protocols. This is because updates on shared data make copies cached by large numbers of mobile clients out-of-date. In the protocol with synchronous broadcasting, aborts increase more rapidly, because more conflicting transactions request commit during the same broadcasting period. In this case the synchronous protocol permits only one transaction to commit, thus all other transactions that show conflicts with the committing one should abort. This shows the primary drawback of the synchronous protocol. For ASBP, the number of aborts increases relatively slowly, as the server sends broadcasting message immediately after receiving *commit_requests*. The immediate broadcasting avoids most aborts of the synchronous protocol, as it reduces transactions' chance to access stale copies of updated data items. HBP shows satisfying results, which are similar to those of ASBP, as it applies the asynchronous broadcasting scheme to updated data items which are likely to be accessed by many clients.

Figs. 4 and 5 show the throughput results for all three protocols. As shown in this figure, the throughput degrades with increasing write operations, due to aborts and delay for updating transactions. When only 10% of data items are widely shared by mobile clients, the protocol using synchronous broadcasting shows a stable throughput as can be seen in Fig. 4. When the write probability is low, ASBP and HBP show better perform-

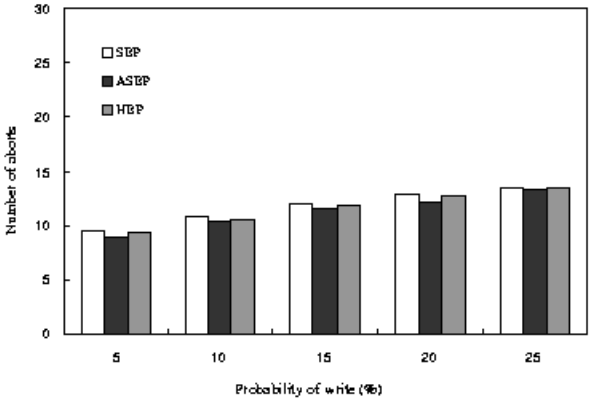

Fig. 2. Average number of aborts with 10% *SharedDegree*.

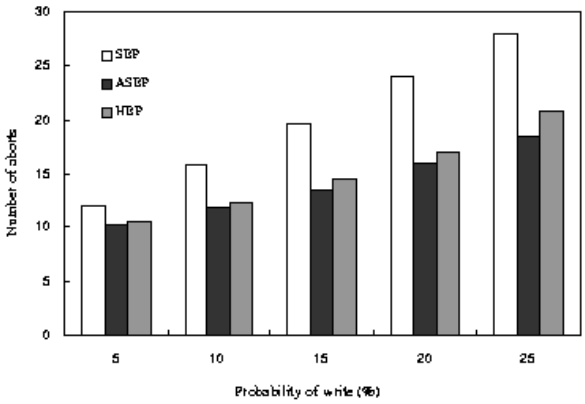

Fig. 3. Average number of aborts with 40% *SharedDegree*.

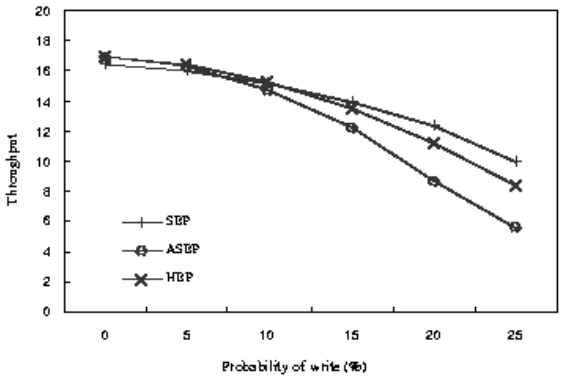

Fig. 4. Throughput with 10% *SharedDegree*.

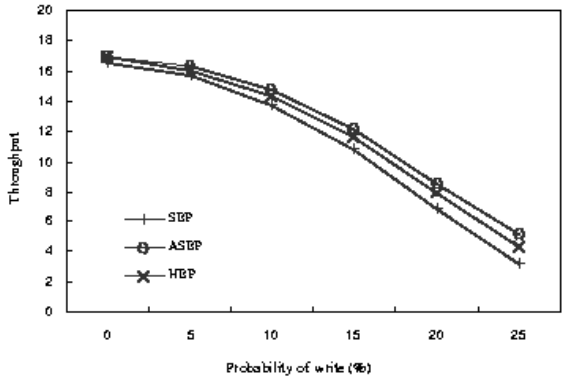

Fig. 5. Throughput with 40% *SharedDegree*.

ance than the synchronous protocol, because of autonomous commit of read-only transactions. However, the throughput of ASBP degrades rapidly with increasing write operations, mainly due to the frequent transmission of broadcasting messages. The immediate broadcasting strategy of ASBP is a communication overhead which cannot reduce aborts of transactions, when *SharedDegree* is low (see Fig. 2). Thus, in this case, the synchronous protocol is more satisfactory, because it does not suffer from a high ratio of aborts without high communication costs. HBP shows intermediate result between ASBP and the synchronous protocol, as it applies immediate broadcasting to 10% of all data items. On the other hand, when 40% of data items are widely shared by mobile clients, proposed protocols show better performance in the whole range of update probability, because ASBP and HBP can avoid a number of aborts with asynchronous broadcasting (see Fig. 3).

5. CONCLUSIONS

Caching of data items in mobile clients is an effective model that will reduce contention on narrow bandwidth wireless channels. In this paper, we propose two cache management protocols, supporting transaction semantics. ASBP adopts an asynchronous broadcasting strategy to reduce conflicts between transactions and to decrease waiting time after completing all operations. Thus it can adapt to the update probability of transactions in mobile clients. HBP dynamically adopts an appropriate broadcasting strategy between synchronous and asynchronous strategies, according to the locality of cached copies of updated data. Thus HBP can adapt not only to update probability, but also to locality of data copies. Simulations were conducted to evaluate the performance of proposed protocols. Our simulations show that ASBP performs well when large part of data items are widely shared by clients in their cache, due to reduced aborts of transactions. Also, our results show that HBP can achieve a satisfactory balance between abort probability and broadcasting costs.

REFERENCES

1. E. Pitoura and B. Bhargava, "Maintaining consistency of data in mobile computing environments," in *Proceedings of International Conference on Distributed Computing Systems*, 1995, pp. 404-413.
2. E. Pitoura and G. Samaras, *Data Management for Mobile Computing*, Kluwer, Boston, 1998.
3. D. Barbara and T. Imielinsky, "Sleepers and workaholics: caching strategy in mobile environments," *VLDB Journal*, Vol. 4, 1995, pp. 567-602.
4. J. Jing, A. Elmagarmid, A. Helal, and A. Alonso, "Bit sequences: an adaptive cache invalidation method in mobile client/server environments," *Mobile Networks and Applications*, Vol. 2, 1997, pp. 115-127.
5. M. J. Franklin, "Caching and memory management in client-server database systems," Ph.D. Thesis, Dept. of Computer Science, University of Wisconsin, 1993.
6. P. A. Berstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Massachusetts, 1987.
7. D. Barbara, "Certification reports: supporting transactions in wireless systems," in *Proceedings of IEEE International Conference on Distributed Computing Systems*, 1997, pp. 466-473.
8. J. Shanmugasundaram, A. Nithrakashyap, and R. Sivasankaran, "Efficient concurrency control for broadcast environments," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1999, pp.85-96.
9. T. Imielinski and B. Badrinath, "Mobile wireless computing: challenges in data management," *Communications of the ACM*, Vol. 37, 1994, pp. 18-28.
10. E. Pitoura and P. K. Chrysanthis, "Exploiting versions for handling updates in broadcast disks," in *Proceedings of International Conference on Very Large Databases*, 1999, pp .114-125.
11. V. C. S. Lee and K.-W. Lam, "Optimistic concurrency control in broadcast environments: looking forward at the server and backward at the clients," in *Proceedings of International Conference on Mobile Data Access*, 1999, pp. 97-106.

IlYoung Chung received his BS, MS and Ph.D. degree in computer science from Korea University, Seoul, Korea in 1994, 1996 and 2001, respectively. He is currently a post-doctorate research associate in the Department of Computer Sciences at Purdue University. His research interests include distributed systems, transaction processing, distributed databases, and mobile databases.

Le Gruenwald received her BS in Physics from the University of Saigon, Vietnam in 1978, MS in Computer Science from the University of Houston in 1983, and Ph.D. in Computer Science from Southern Methodist University in 1990. She was a software engineer at White River Technologies, a lecturer in the Computer Science and Engineering Department at Southern Methodist University, and a member of technical staff in the Database Management Group at the Advanced Switching Laboratory of NEC, America. She is a Samuel Roberts Noble Foundation Presidential Professor and an associate pro-

fessor in the School of Computer Science at University of Oklahoma. Her research interests include distributed and mobile databases, real-time main memory databases, web databases, object-oriented databases, data warehouse, data mining, and multimedia databases.

Chong-Sun Hwang received his BS and MS degrees in mathematics from Korea University, Seoul, Korea in 1966 and 1970, respectively, and his Ph.D. degree in computer science and statistics from University of Georgia in 1978. From 1978 to 1980, he was an assistant professor at University of South Carolina, Lander. He is currently a professor in the Department of Computer Science and Engineering at Korea University, Seoul, Korea. His research interests include distributed systems, distributed algorithms, fault tolerant systems, and mobile computing.

SoonYoung Jung received his BS, MS, and Ph.D. degrees in computer science from Korea University, Seoul, Korea in 1990, 1992 and 1997, respectively. From 1997 to 2000, he was an senior researcher at Korea Advanced Integration Technology, ECO Inc.. He is currently a professor in the Department of Computer Science Education at Korea University, Seoul, Korea. His research interests include web-based education systems, database systems, knowledge management systems, and mobile computing.