Controlling Concurrency in Mobile Computing Environments with Broadcast-Based Data Dissemination

José Maria Monteiro[1,2] and Ângelo Brayner[1]

[1] Departamento de Informática, UNIFOR
Av. Washington Soares 1321, 60811-905, Ceará, Brasil
{monteiro,brayner}@unifor.br
[2] Departamento de Informática, PUC-Rio
R. Marquês de São Vicente 225, 22453-900 - Rio de Janeiro, Brasil
monteiro@inf.puc-rio.br

Abstract. A wireless broadcast environment is defined as a mobile computing environment in which data are delivered to mobile clients by means of a broadcast-based mechanism. Of course, those applications have to see the most recent consistent database state. For that reason, in such a scenario, database servers should synchronize operations for ensuring data consistency and currency of data. However, conventional serializability-based concurrency control protocols are unsuitable for synchronizing transactions in broadcast environments. The major goal of this work is to present a new serializability-based protocol to synchronize transactions in data intensive applications. The proposed protocol saves battery power, since it ensures that mobile clients do not have to contact servers (for requiring locks, for example) to access data. Thus, mobile clients do not need to listen to the broadcast continuously; they listen to the broadcast channel to retrieve data they need. Therefore, the proposed protocol supports client disconnections. We performed simulation analysis to evaluate the performance of the new protocol. The simulation results show that the proposed protocol offers better performance than others protocols.

1 Introduction

The integration of portable computer technology with the wireless-communication technology has created a new paradigm in computer science, the so-called mobile computing. In a mobile computing environment, network nodes are no longer fixed, that means, they do not have a fixed physical location. In such an environment, mobile users using a portable computer (denoted mobile client or host) may access shared information and resources regardless of where they are located or if they are moving across different physical locations and geographical regions.

Mobile computing technology has made possible the development of new and sophisticated database applications. A particular class of such applications can be characterized by having a large number of mobile clients, a small number of servers and a relatively small database. Electronic commerce applications, such as auctions, road traffic management systems and automated industrial plants [8] are examples of database applications, which require the support of the mobile computing technology. Those applications can benefit from a broadcast mode for data dissemination (*push-*

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 1069–1079, 2005.

based approach for data dissemination). In this model, a server repetitively broadcasts data to a client population without a specific request. In turn, clients monitor the broadcast channel in order to retrieve their data items of interest. Conventionally, data are delivered to clients on demand (*pull-based* approach for data dissemination).

Broadcasting data to mobile clients instead of sending them on demand has several advantages. For instance, the database server is not overloaded with requests from a large population of mobile clients and it does not have to send individual messages to a specific client as in pull-based systems. Furthermore, data can be accessed concurrently by any number of clients without any performance degradation, since all mobile clients can simultaneously listen to the broadcast channel.

Therefore, broadcast-based data dissemination has become a widely accepted technique of disseminating data in mobile computing. Many such systems have been proposed and some commercial products for information dissemination in wireless networks already support broadcast. For example consider the AirMedia system [3], which regularly sends CNN news and information to subscribers. Such subscribers should be equipped with a receiver antenna connected to their personal computers.

A broadcast environment is defined as a mobile computing environment in which data are delivered to mobile clients by means of a broadcast-based mechanism. Applications running in a broadcast environment need to read the most recent consistent database state[1]. Therefore, the database server (database system running on a server machine) should ensure that mobile clients "see" the most recent consistent state of the database. In other words, the database server has to guarantee data consistency and currency of data. However, most of the published approaches for controlling concurrency of operations over databases in broadcast environments require that complex control structures should be sent to mobile clients. Besides having to store such structures, the clients need to be in active state for longer period of time in order to manage those structures. Approaches such as invalidation report [7] and update consistency [8] present these drawbacks. For example, in [8] an n x n matrix should be sent to all mobile clients, where n is the number of database objects.

In this paper, we propose a new concurrency control protocol for broadcast environments. The protocol, denoted *temporal serialization graph testing* (TSGT, for short), explores temporal information about database operations (read and write). The proposed protocol does not require that complex structures be sent to the mobile clients. The TSGT protocol reduces the communication traffic between server and clients and minimizes the time interval in which clients need to listen to the broadcast channel.

The rest of the paper is organized as follows. In section 2, we outline the characteristics of broadcast environments. Section 3 describes the transactional model that we will use in this work. In section 4, we describe and analyze the proposed protocol for concurrency control in broadcast environments. In section 5, the most important mechanisms for concurrency control in broadcast environments will be described and discussed. Section 6 shows the results of our simulation experiments. Section 7 concludes this work and outlines future works.

[1] Roughly, we can say that a consistent database state represents an acceptable view of the real world

2 Mobile Computing Environments with Broadcast-Based Data Dissemination

The main components of a broadcast-based data dissemination environment are described next. The database consists of a collection of interrelated data items. The database server (DBMS) is responsible for storing and managing data of the database. The broadcast server periodically broadcasts data items to clients. The clients, in turn, are mobile computers. Applications running on mobile clients perform read and write operations on database items which are cached by mobile clients.

The broadcast-based data dissemination differs from the traditional model for data transfer between clients and server. Traditionally, data are sent from the servers to clients on demand. In broadcast environments, the server periodically broadcasts data items to a client population without a specific request. Each broadcast period is called broadcast cycle or *bcycle*, while the content of broadcast is called *bcast*. Clients monitor the broadcast channel and retrieve data items they need.

From a transaction processing point of view, it is important to note that, when data items are broadcast to mobile clients, they are accessed by local transactions running on those clients. On the other hand, transactions running on the database server can update those data items after they were broadcast. Thus, it is likely that a mobile client reads data item instances which do not exist anymore in the database. Of course, such a phenomenon should be avoided. Furthermore, since mobile clients can be disconnected for long periods of time, transactions running on mobile clients are likely to be long-living transactions.

3 Transaction Model

A database consists of a collection of disjoint objects representing entities of the real world. The set of values of all objects stored in a database at a particular moment in time is called database state. Database states represent snapshots of the real world. They can only reflect static aspects of the world. However, a database must also reflect changes in the real world. Such changes are captured by the notion of state transition. State transitions represent "jumps" from a particular database state to another (an updated snapshot of the real world).

The real world imposes some restrictions on its entities. Additionally, databases must capture such restrictions, denoted consistency constraints. We can couple the concept of database state to consistency constraints. If the values of objects of a particular database state satisfy all the consistency constraints, the database state is said to be consistent.

Application programs containing operations on database objects are tools whereby state transitions are realized in a database. From the concurrency control perspective, not all operations of a program are relevant. Only database operations have to be considered. A transaction is an abstraction which represents a sequence of database operations resulting from the execution of an application program. Hence, transactions are modeled as finite sequences of operations on database objects. We use the notation $r_i(x)$ ($w_i(x)$) to represent a read (write) operation by a transaction T_i on object x. $OP(T_i)$ denotes the set of all operations executed by T_i . We will assume that the

execution of a transaction preserves the database consistency, if this transaction runs entirely and isolation from other transactions.

We categorize transactions in a broadcast environment in two classes. One class comprises transactions executed at the mobile clients. Transactions belonging to this class are called mobile transactions. Transactions belonging to the second class are called server transactions, since they run at the database server.

Transactions are executed concurrently. The concurrent execution of a set T of transactions is realized by interleaving the operations of transactions in T. The execution of several interleaved transaction is modeled by a structure called schedule. Formally, a schedule over a set $\mathfrak{T}=\{T_1,T_2, \ldots,T_n\}$ of transactions represents an interleaved sequence of operations of transactions in $\mathfrak{T}$ which is an element of the shuffle product $T_1*T_2*\ldots*T_n$. Serial executions of transactions are modeled by means of the notion of serial schedules. The precedence relation (execution order) between two operations in a schedule S is represented by $<_S$. For example, the notation $p <_S q$ indicates that operation p was executed before q in schedule S. Two operations of different transactions conflict (or are in conflict) if and only if they access the same object of the database and at least one of them is a write operation. It is important to note that not all schedules are valid; only some of them preserve database consistency. Hence, identifying whether a schedule is correct is a key point in transaction management.

Let S be a schedule over a set $\mathfrak{T}=\{T_1, T_2, \ldots, T_n\}$ of transactions. The serialization graph for S, denoted GS(S), is defined as the directed graph SG(S) = (N,E) in which each node in N corresponds to a transaction in $\mathfrak{T}$. The set E contains edges of the form $T_i \rightarrow T_j$, if and only if $T_i, T_j \in N$ and there are two operations $p \in OP(T_i)$, $q \in OP(T_j)$, where p conflicts with q and $p <_S q$. A schedule S is conflict serializable if and only if the serialization graph for S (SG(S)) is acyclic. A schedule S is correct if it is serial or conflict serializable.

4 Synchronizing Database Operations in a Broadcast Environment

In this section, we will describe and analyze the concurrency control protocol we propose for synchronizing database operations (belonging to different mobile and server transactions) in a broadcast environment. The proposed protocol, called temporal serialization graph testing (TSGT), ensures that broadcast environment applications access consistent and current data.

The TSGT protocol is based on a similar strategy used by the conventional serialization graph testing protocol [5]: the dynamic monitoring and management of an always acyclic conflict graph. In contrast to the classic serialization graph testing, the TSGT exploits temporal information w.r.t. the moment in which a mobile transaction operation (read or write) is executed on a given database item.

In our approach, we have decided to distribute concurrency control functions among mobile clients and the database server. Thus, we assume that the server and the clients execute specific functionalities, in order to manage the transaction processing in a broadcast environment. In the following, we describe such functionalities.

During each broadcast cycle, the server broadcasts the data items together with a timestamp. We will assume that data item values sent in broadcast during each cycle correspond to the database state immediately before the beginning of the broadcast

process. In other words, data instances sent during a broadcast correspond to them produced by all the transactions that had executed commit operations until the beginning of the broadcast cycle. Such transactions will be called "committed" transactions. Accordingly, the database server should store two versions of each data item O_i:

(i) a version corresponding to the result yield by the last committed transaction which has updated O_i, and;
(ii) a version corresponding to the result yield by the last non-committed transaction which has updated O_i;

The server is also responsible for building and managing the temporal serialization graph for a schedule, named *global schedule*, consisting of operations belonging to mobile and server transactions. A global schedule models the temporal execution order in which operations of mobile and fixed transactions are executed. This is possible because mobile clients send to the server timestamps for database operations they execute. In Section 4.2, we describe how those timestamps are defined.

Periodically, clients must send a package (message) to the server. Such a package consists of database objects on which a mobile transaction has executed a database operation (read or write) and the operation type. This information is sent together with the corresponding timestamp. Information of operations already informed does not need to be sent again. When a client receives a commit or an abort request of a mobile transaction T_i, it sends a message to the server consisting of request (commit or abort). After that, the client waits for an acknowledgement from the server in order to execute the commit or abort operation.

4.1 Running Example

We motivate the applicability and feasibility of our proposal by describing an application of electronic commerce. In such an application the stock of the main technology companies is available to auction in an electronic stock exchange. Now consider the following set of transactions, which read and update values of the stock: T_1: $r_1(IBM)r_1(SUN)C_1$; T_2: $w_2(IBM)C_2$; T_3: $r_3(IBM)r_3(SUN)C_3$; T_4: $w_4(SUN)C_4$; T_5: $w_5(SUN)C_5$

The transaction T_2, T_4 and T_5 are executed at the server. On other hand, the transaction T_1 is executed on the client A, while the transaction T_3 on client B. Now consider the global schedule GS presented in figure 1.

We assume that, in the execution scenario presented in figure 1, the packages containing information about the read operations of mobile transactions during the $bcycle_n$ arrive at the server before sending $bcast_{n+1}$. The serialization graph for the schedule GS is illustrated in figure 2 (a). Observe that the graph presents a cycle of the form $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$. Therefore, schedule SG is not conflict serializable (correct).

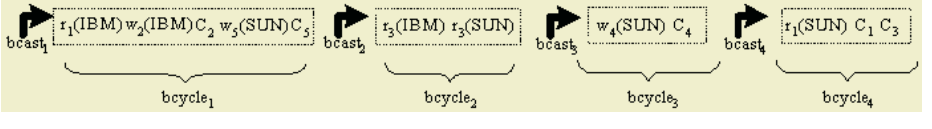
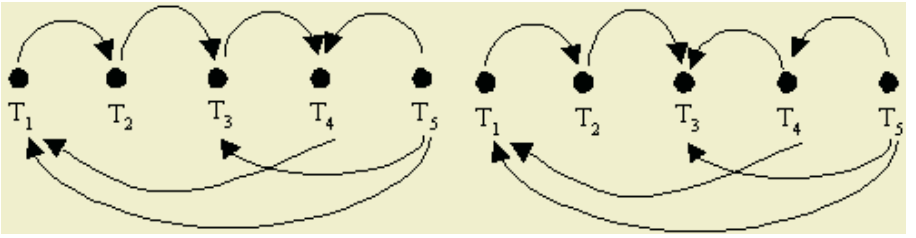


Fig. 1. Schedule GS



(a) Serialization Graph for GS (b) Serialization Graph for GS'

Fig. 2. Serialization Graphs for Schedules GS and GS'

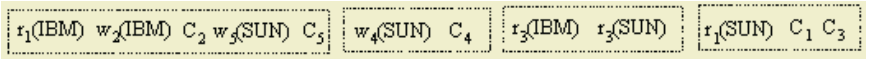


Fig. 3. Schedule GS'

Now, we assume that, for some reason (problems in the communication links, for example), the package which contains the information about the read operations executed by transaction T_3 during $bcycle_2$ is late. In this case, the server will see the schedule GS' which is showed in figure 3. The serialization graph for GS' is depicted in figure 2 (b).

In figure 2 (b), the serialization graph for GS' does not present cycles. Consequently, the schedule GS' will be considered conflict serializable, that is, it is correct.

However, observing the database state we can see that it is not consistent. That means, the correct edge between T_3 and T_4 in the serialization graph should be $T_3 \rightarrow T_4$ (as depicted in figure 2 (a)) and not $T_4 \rightarrow T_3$ (figure 2 (b)). Therefore, an incorrect execution was considered correct improperly. Therefore, the conventional serialization graph is not sufficient to identify incorrect schedules in broadcast environments. To avoid the occurrence of such phenomenon, we propose the TSGT protocol, which will be described in the next section.

4.2 The Temporal Serialization Graph Testing Protocol

The TSGT protocol consists basically of monitoring and management an always acyclic graph. The graph maintained by TSGT protocol is called temporal serialization graph. This graph is constructed based on temporal information about the moment when a data item was read or updated.

Definition 1. $C(p_i(x))$ denotes the timestamp value for the operation $p_i(x)$. The value for $C(p_i(x))$ is defined as follows. If T_i is a transaction executed on the server, then $C(p_i(x))$ is the cycle number when the operation $p_i(x)$ is executed. On the other hand, if T_i it is a mobile transaction, then $C(p_i(x))$ represents the cycle number in which a transaction T_j has executed its commit operation, if $w_j(x) \in OP(T_j)$ and T_j is the last transaction which has performed a write operation on x. When a mobile transaction T_k executes an operation $p_k(x)$, this timestamp will be associated to $p_k(x)$.

Definition 2. The temporal precedence relation (temporal execution order) between two operations in a schedule S is denoted by $\prec^t_S$. For example, $p \prec^t_S q$ indicates that

operation p is temporally executed before q in a schedule S. Let p and q operations in a Schedule S, we define that $p \prec^t_S q$ if and only if one of the following conditions holds:

i) $C(p) < C(q)$
ii) $C(p) = C(q)$ and $p \in OP(T_i)$, $q \in OP(T_j)$, T_i is a mobile transaction, $q <_s p$, T_j commits in a cycle, whose number is greater than C(p).
iii) $C(p) = C(q)$ and $p \in OP(T_i)$, $q \in OP(T_j)$, T_i and T_j are transactions executed on the server, $p <_s q$.
iv) $C(p) = C(q)$ and $p \in OP(T_i)$, $q \in OP(T_j)$, T_i is a mobile transaction, T_j is a transaction executed on the server, $p <_s q$.

Definition 3. Let S be a schedule over a set $\Im=\{T_1,T_2,...,T_n\}$ of transactions. We define the temporal serialization graph (TSG) for S, denoted TSG(S), as a directed graph TSG(S) = (N,E), where:

(i) $N=\Im$, that is, each node in N represents a transaction in $\Im$, and;
(ii) E represents the set of edges $T_i \rightarrow T_j$, where
- $T_i, T_j \in N$;
- there are two operations $p \in OP(T_i)$ and $q \in OP(T_j)$, which are in conflict and;
- $p \prec^t_S q$.

A schedule S is conflict serializable if and only if the temporal serialization graph for S (TSG(S)) is acyclic. A schedule S is correct if it is serial or conflict serializable.

Next, we describe how a TSGT scheduler manages the temporal serialization graph. When a scheduler starts running, the TSG is created as an empty graph. During each broadcast cycle, the server broadcasts the values of data items (last value written by committed transactions) with the respective timestamp. The timestamp for each data item can be sent in the message header or together with each data item. For each read operation, the client stores the value and the identification of the read item, together with the respective timestamp. Periodically, clients should inform to the server, which read operations they have executed. That is, a client sends periodically a package containing the item identification and the respective timestamp for each read operation. As soon as the scheduler receives the first operation of a new transaction T_i, a node representing this transaction is inserted in the TSG. For each operation $p_i(x) \in OP(T_i)$ which is received, the scheduler executes the algorithm shown in Figure 4.

In order to illustrate the correctness of the TSGT protocol, consider the example presented in Section 4.1. Observe that the graph produced by the basic TSGT protocol corresponds to the correct serialization graph for the schedule GS (see figure 1). The protocol described above ensures that the scheduler identifies that $C(w_4(SUN)) > C(r_3(SUN))$, inserting, thus, the edge $T_3 \rightarrow T_4$, and not $T_4 \rightarrow T_3$. Therefore, the TSGT protocol captures the information that the operation $r_3(SUN)$ was temporally executed before $w_4(SUN)$.

Thus far, we have analyzed global schedules with mobile transactions involving only read operations. However, the protocol proposed in this work can control concurrency in environments with mobile transactions involving write (update) operations as well, while ensuring database consistency. Next, we show how the TGST protocol can be used to control concurrency in such environments. First, we need to make the following observation. The semantic of write operations of a mobile transac-

tion stays that an operation $w_i(x)$ is in fact executed, when it arrives at the database server.

Remark 1. Let $C(p_i(x))$ be the timestamp value for the operation $w_i(x)$, where T_i is a mobile transaction. $C(p_i(x))$ represents the cycle number when the operation $p_i(x)$ arrives at the server.

Step 1. The scheduler checks if there exists a conflicting operation $q_j(x) \in OP(T_j)$ which has been already scheduled. If there is such an operation $q_j(x)$, then the scheduler inserts an edge between T_i e T_j. In order to include such an edge correctly, two different cases should be considered:

Case 1. T_i is a transaction executed on the server. In this case, the scheduler will execute the following temporal verification:

If $C(q_j(x)) \leq C(p_i(x))$

Then, the scheduler inserts an edge on the form $T_j \rightarrow T_i$.

Else

The scheduler inserts an edge on the form $T_i \rightarrow T_j$.

Case 2. T_i is a mobile transaction. In this case, the scheduler will execute the following temporal verification:

If $C(q_j(x)) < C(p_i(x))$

The scheduler inserts an edge on the form $T_j \rightarrow T_i$

Else

If $C(q_j(x)) > C(p_i(x))$

The scheduler inserts an edge on the form $T_i \rightarrow T_j$

Else

If T_j has already executed the *commit* operation

The scheduler inserts an edge of the form $T_j \rightarrow T_i$

Else

The scheduler inserts an edge of the form $T_i \rightarrow T_j$

Step 2. The scheduler verifies if the new edge introduces a cycle in the temporal serialization graph. In the affirmative case, the scheduler rejects the operation $p_i(x)$, undoes the effect of the operations of T_i and removes the edge inserted. Otherwise, $p_i(x)$ is accepted and scheduled

Fig. 4. A scheduler implementing the TGST protocol

4.3 Correctness of the TGST Protocol

Next, we prove that schedules produced by the TSGT protocol are conflict serializable. That means, the TSGT protocol ensures database consistency.

Theorem 1. Let *TSGS* be the set of schedules over the set $\Im=\{T_1, T_2, \ldots, T_n\}$ of transactions produced by a TSGT protocol and CSR the set of all conflict serializable schedules over $\Im$. Then *TSGS*=CSR [6].

Sketch of Proof. It is easy to show that TSGS $\subset$ CSR. We only need to observe that every global schedule S produced by the TSGT protocol has an acyclic temporal serialization graph. By definition, TGST(S) represents the conventional serialization graph for S, with additional temporal information to capture the correct execution order of the operations in S. In other words, if the TSG for S is acyclic, the serialization graph is too. Therefore, S$\in$CSR, consequently, *TSGS*$\subset$CSR. To prove that TSGS$\supset$CSR, we have to show that every schedule S $\in$ CSR can be produced by a TSGT protocol. We can show this by induction on the length of S that every operation

p in S may not originate a cycle in the TSG and, thus, p may be executed. As already mentioned, the TSG represents the conventional serialization graph for S, with temporal information.

5 Related Work

In this section, we will describe and analyze the most important proposals for the concurrency control in broadcast environments. Initially, we will discuss the invalidation reports approach proposed in [7]. According to this approach, each *bcast* is preceded by an invalidation report. Such a report represents a list of all data items that was updated on the server during the previous broadcast cycle. Client read the invalidation report periodically. A mobile transaction T is aborted if an object x previously read by T appears in the invalidation report. This approach discards some conflict serializable schedules. Moreover, the client cannot be disconnected for long periods of time, since the client needs to read every invalidation report.

The multiversion broadcast mechanism [2] consists of keeping previous versions of data items, in order to reduce the number of aborts of mobile transactions. In this approach, the server, besides broadcasting database objects, broadcasts multiple versions for each object. Let C_0 be the broadcast cycle number during which the client transaction T performs its first read operation. To each new read operation, the transaction T tries to read the value with the largest version number C_n, such that $C_n \leq C_0$. If this version is not available the transaction is aborted. Therefore, this approach does not eliminate the necessity of aborting mobile transactions. Moreover, it generates an overhead in the execution of the read operations and to maintain the multiple versions for each database object..

Shanmugasundaram et al. [8], proposes a mechanism which uses as correctness criterion an extension of the criterion called update consistency. According to this proposal, in each *bcycle* the server broadcasts an *n* x *n* matrix, where n is the number of database objects. This matrix will be used by clients in order to ensure the consistency of read operations. For that, clients should listen to the broadcast channel during a larger period of time in order to retrieve the control matrix. It is important to note that the clients also have to store the control matrix.

As we can observe, most of the proposals described in this paper requires that control structures are transmitted by the server during each broadcast cycle and that mobile clients store and manage such structures. For this reason, we can claim that TSGT protocol is more efficient for the concurrency control in broadcast environments than the existing proposals.

6 Experimental Results

In order to evaluate the performance of the proposed protocol, we compared it with the F-MATRIX [8] and multiversion [2] protocols. We evaluated the performance of these protocols based on the following metrics:

- **Transaction Response Time:** This metric indicates the time interval between the time a transaction T is submitted by a client and the time that T ends its execution through a commit operation (including the time involved in restarts).

- **Transaction Restart:** This metric indicates the number of restarts occurred for a set of concurrent transactions. Observe that this metric indirectly measures the abort rate.

6.1 Simulation Environment

The simulation environment is based on the model used in [8]. It consists of a server, a client, and a broadcast server for transmitting both the data objects and the required control information. A mobile transaction is processed until it is committed. Only read-only transactions are executed on the client. Update transactions are executed on the server. A small database (300 data objects) helps to intensify data conflicts by creating hot-spot effect. The objects that the transactions access are determined using a random distribution function. The transaction length indicates the number of operations in a transaction. In the simulation experiments we used 8 operations with the default value for the server transaction length.

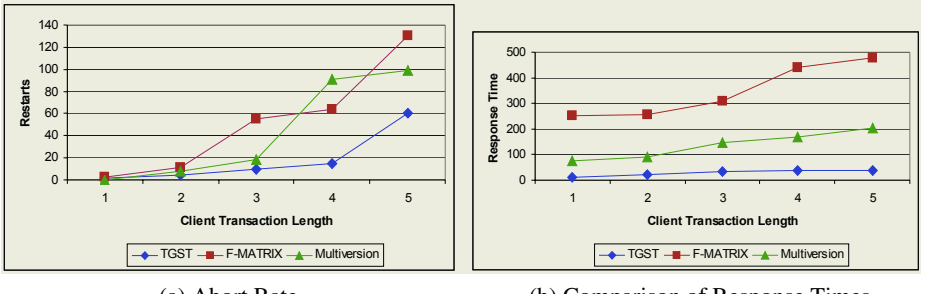
6.2 Simulation Results

Fig. 5 show the results of our simulation experiments. TSGT outperforms F-MATRIX and Multiversion in all the experiments. Furthermore, TSGT is highly scalable with respect to client transaction length and server transaction length.

Figure 5 (a) shows that our protocol presents a lower abort rate than the F-Matrix and Multiversion protocols. It shows that our protocol is scalable w.r.t the length of transactions as well.

Figure 5 (b) shows that transactions are executed in smaller time intervals than the F-Matrix and Multiversion protocols. Observe that, if we have smaller time intervals for executing transactions, we increase the throughput of the system.

Although energy usage has not been evaluated in our simulations we claim that TSGT (in comparison with F-MATRIX and Multiversion protocols) provides reduction in the use of this important and scarce resource, since it reduces the time that mobile clients need be connected (in "active" mode). This is because mobile clients do not require to listen to the broadcast channel continuously; they listen to the broadcast channel only to retrieve data they need. Moreover, with lower abort rates and smaller response times client transactions will commit more quickly, saving energy.



(a) Abort Rate (b) Comparison of Response Times

Fig. 5. Simulation Results

7 Conclusions

In this paper we have proposed a new mechanism for concurrency control in broadcast environments. The proposed protocol, called temporal serialization graph testing protocol (for short, TSGT), ensures that applications in broadcast environments have access to consistent and current data. The TSGT protocol ensures that clients do not need to contact the server to perform their operations. Clients just have to listen to the broadcast channel in order to retrieve data items of interest and can be disconnected for long periods of time. Moreover, clients do not need to store nor to manage complex structures as proposed in [2], [7] and [8]. We performed simulation studies to evaluate the performance of the new protocol. The analysis of simulation results showed that the proposed protocol presents a better performance than F-Matrix [8] and Multiversion [2] protocols.

References

1. Victor C.S. Lee and Sang H. Son. On Transaction Processing with Partial Validation and Timestamp Ordering in Mobile Broadcast Environments. IEEE Transactions on Computers, Vol. 51, No. 10, 2002.
2. Evaggelia Pitoura and Panos K. Chrysanthis. Multiversion Data Broadcast. IEEE Transactions on Computers, Vol. 51, No. 10, 2002.
3. Web Page of Airmedia inc. White Paper, http:// www.airmedia.com
4. A. Brayner, T. Härder and N. Ritter. Semantic Serializability: A Correctness Criterion for Processing Transactions in Advanced Database Applications. DATA & KNOWLEDGE ENGINEERING, 31, 1999.
5. M.A. Casanova. The Concurrency Problem of Database Systems. In Lectures Notes in Computer Science, 116, 1981.
6. J.M. Monteiro. Temporal Serialization Graph Testing: An Approach to Control Concurrency in Broadcast Environments. Msc. Dissertation, Universidade Federal do Ceará, October, 2001 (in Portuguese).
7. E. Pitoura e P. Chrysanthis, Scalable Processing of Read-Only Transactions in Broadcast Push, IEEE International Conference on Distributed Computing Systems, 1999.
8. J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran e K. Ramamritham. Efficient Concurrency Control for Broadcast Environments. Proceedings of the ACM SIGMOD Conference, 1999.