Broadcast of Consistent Data to Read-Only Transactions from Mobile Clients[1]

Kam-Yiu Lam, Mei-Wai Au and Edward Chan
Department of Computer Science
City University of Hong Kong
83 Tat Chee Avenue, Kowloon
email: {cskylam|csedchan}@cityu.edu.hk

Abstract

In this paper, we study the inconsistency problem in data broadcast. While data items in a mobile computing system are being broadcast, update transactions may install new values for the data items. If the executions of update transactions and the broadcast of data items are interleaved without any control, it is possible that the mobile transactions, which generated by mobile clients, may observe inconsistent data values. In this paper, we propose a new algorithm, called Update-First with Order (UFO), for concurrency control among the mobile transactions and update transactions. The mobile transactions are assumed to be read-only. In the UFO algorithm, all the schedules among them are serializable. Two important properties of the UFO algorithm are that (1) the mobile transactions do not need to set any lock before they read the data items from the "air"; and (2) its impact on the adopted broadcast algorithm, which has been shown to be an efficient method for data dissemination in mobile computing systems, is minimal.

1 Introduction

Recent advances in wireless communication technology have greatly increased the functionality of mobile information services and have made many mobile computing applications a reality. A number of novel mobile information services, such as mobile shopping aids in a large shopping mall and financial information distribution to users via mobile phones and palmtop computers, have already been implemented. Different innovative applications, such as real-time traffic information and navigation systems, and real-time stock information systems, will no doubt continue to emerge as data-hungry users require instant access to information using their palmtops and notebook computers no matter where they are located.

Due to the intrinsic constraints of mobile computing systems, such as limited bandwidth, limited electric power supply and unreliable communication, the design of an efficient and cost effective mobile computing system requires to solve many new problems, which need not to be considered in a distributed system supported with wired network [AFZ97, FZ98, IB94, PB93, PB98]. One of the most important issues is efficient data dissemination to transactions from mobile clients [IB94, PS98]. The problem is complicated by the limitation of bandwidth and low transmission quality of the mobile network.

In recent years, different data dissemination methods have been proposed to meet the different data requirements of the mobile clients [AFZ97, HV97, LS95, LS97, SNSR98, XSGFR97]. Basically, there are two approaches to disseminate data items from the information server to the transactions from the mobile clients. They are *on-demand* and *data broadcast*. In the on-demand approach, the data items required by a transaction will be sent from the information server on request. This approach is simple but does not scale well with the number of mobile clients. The waiting time for a data item will be very long if there are a large number of transactions waiting for different data items [AFZ97, XSGFR97].

[1] This work was supported in part by City University Strategic Grant #7000584.

In the broadcast approach, the information server periodically and continuously broadcasts data items one by one to the mobile clients. If there is a transaction waiting for a data item, it will get the data item from the "air" while it is being broadcast [AFZ97]. Thus, the cost for data dissemination is independent of client number since a data broadcast can satisfy multiple requests for the same data item, resulting in a much more efficient way of using the bandwidth. It is therefore quite suitable for disseminating substantial amount of information and data to a large number of mobile clients where bandwidth efficiency is a major concern.

An important consideration in data dissemination to transactions from mobile clients is to provide consistent data values to them. In data broadcast, the transactions do not need to inform the database server or set any locks at the database server before they access the data items. They can get the data items from the "air" while the data items are being broadcast. However, if the update of the database is done concurrently, the transactions may observe inconsistent data values. Allowing updates to be interleaved with data broadcast is important in maintaining the "freshness "of the data items. However, this will have the problem of concurrency control. Unfortunately, the concurrency control protocols such as two phase locking, optimistic method or timestamp ordering are not suitable for this kind of systems as they require a lot of overhead in setting locks and detecting data conflicts [BHG87]. In this paper, we study the inconsistency problem in broadcasting of data items to transactions from mobile clients. A new algorithm, called Update-First with Order (UFO), is proposed. In the protocol, we assume that all update are at the database server. We also assume that all the transactions from mobile clients are read-only as most of the mobile computing systems only allow the generation of read-only transactions from mobile clients for retrieving different types of information such as stock data, traffic information and news updates. Since for a lot of common broadcasting system such as the weather and traffic information systems number of update transactions are much less than the number of read-only transactions. We assume the number of data conflicts between update transaction and the read-only mobile transaction is low.

The organization of the remaining parts of the paper is as follows. Section 2 reviews the related work on concurrency control in data broadcast. Section 3 defines our transaction models for mobile computing systems with data broadcast. Section 4 discusses the data inconsistency problems using examples. Section 5 introduces our new algorithm, Update-First with Order (UFO) and discusses its properties, correctness and implementation. Finally, the Conclusions and Future Work of the paper are in Section 6.

2 Related Work

Although the research in mobile computing systems has received a lot of attention in recent years, the important issue of concurrency control in data broadcast has been greatly ignored. Traditional concurrency control protocols are not suitable to mobile computing systems due to their heavy overhead. For example, the two phase locking usually requires a transaction to set lock on a data item before it is allowed to access the data item [BHG87]. The overhead for locking and lock conflict detection will be very high in a mobile environment. However, in data broadcast, a transaction from a mobile client may access a data item at any time without informing the database server while the data item is being broadcast. This is the basic rationale behind using data broadcast and setting locks is clearly impractical.

Due to the poor quality of service of mobile network, it is not easy to ensure data consistency and to detect data conflict. One common solution is to relax the consistency requirement. In [PB95], a two-level consistency model is proposed. Semantically related data are grouped together into a cluster. The data inside a cluster are mutually consistent. However, certain degrees of inconsistency are allowed among data at different clusters.

To our knowledge, until now, the only study on concurrency control for data broadcast is [SNSR98] in which a control matrix is used for concurrency checking. For a database of n data items, a matrix of size $n \times n$ is used. For each broadcast cycle, the control matrix is broadcast in addition with the data items. A mobile client is required to perform a consistency checking using the matrix before reading any data item from the broadcast schedule. The checking is to ensure that no transaction in the live set of any current transaction writes onto any data item in its read set, otherwise the mobile client has to abort and restart its transaction. Clients can issue update transaction in addition to read transaction. The write operations are performed on a local copy of data items at the client. At the end of a transaction, the whole transaction including all of the read and write operations and the cycle numbers in which they are performed will be sent to the server for commitment. Therefore, the update model is conceptually the same as one which centralizes all updates at the server.

The major drawback of this approach is the large overhead needed to maintain the matrix for concurrency control and conflict checking. The matrix will use up a substantial percentage of broadcast bandwidth especially when the size of data item is small or the size of the database is large. Therefore using such a large matrix is impractical in many real-life applications. Maintaining the control matrix also involves complicated processing at the server. Another undesirable characteristic in [SNSR98] is the clients and server may observe different serialization order on update transactions. The impact on the mobile clients caused by this feature depends on nature of the applications. However, it would be more desirable if they can observe the same global update serialization order.

In review of the lack of an efficient data consistency control in a broadcast environment, we are going to introduce the Update-First with Order (UFO) algorithm in Section 6. The UFO algorithm can maintain the serializability of update transactions at the database server and the read-only mobile transactions. The algorithm has minimal overhead and can be applied in different broadcast models. The UFO algorithm also aims at reducing the number of abort at mobile clients.

3 Transaction Models

A mobile computing system consists of an information server and a number of mobile clients connected to the server through a mobile network such as a cellular radio system. The mobile clients represent users equipped with mobile units which communicate on low bandwidth wireless channels with the information server.

In the information server, a server database is maintained to keep track of the information of the external environment. Some of the data items are highly dynamic such as stock prices, news updates, as well as traffic and weather conditions. To maintain the validity of data items, update transactions have to be installed into the database whenever the status of objects in the external environment has changed. They capture the most current status of the objects and refresh the values of the corresponding data items in the database. Otherwise, the values of the data items will be stale and they may become useless. It is assumed that the update transactions are short and consist of sequence of read and write operations. It is further assumed that a well-formed concurrency control protocol, such as two phase locking and optimistic method [BHG87], is used for concurrency control among the update transactions.

Under the data broadcast approach, the information server broadcasts data items from the database one by one continuously until the end of a broadcast cycle. The length of a broadcast cycle may be fixed or variable. Then, it starts another broadcast cycle immediately. Different data items may require different time to broadcast. The selection of data items to broadcast is performed by a broadcast algorithm. In the last few years, different broadcast algorithms have been suggested to select the data items for broadcast such as based on the deadlines of the transactions and the access frequencies of the data items [LS97, XSGFR97]. It is assumed that the broadcast process is modeled as a transaction, called broadcast transaction (BT). It is a long read-only transaction. The execution of the BT and the update transactions are interleaved.

The mobile clients issue transactions, called mobile transactions (MT), to request data items at the information server. It is assumed that each mobile transaction is a collection of simple data requests (read operations). They are read-only transactions. It is further assumed that the data requests can be performed in any order. Usually, they are short transactions with one to several data requests.

Assumptions of the transaction model are summaries as follows:

(1) All update transactions are at the database server.
(2) All mobile transactions are read-only and their read-sets are unordered i.e. data items required by a mobile transaction can be received in any order.
(3) The conflicts between update transactions and mobile transactions are low which is the case in a lot of common broadcast systems such as weather and traffic information systems.
(4) Mobile transaction has a drop period. Mobile transaction has to abort and restart if it cannot get all required data from broadcast program within the drop period.

4 Data Inconsistency in Data Broadcast

Since broadcast of data items is performed concurrently with the execution of update transactions, it is possible that the mobile transactions may read inconsistent data values due to uncontrolled interleaving of execution of update transactions and broadcast process (broadcast transactions).

In this paper, we adopt serializability as the correctness criterion for transaction execution [BHG87]. If the serialization graph is acyclic, then the schedule is serializable. In the following three examples, we will illustrate the problem that if data broadcast is used, the final schedule among the update transactions and mobile transactions may be non-serializable.

Example 1: Two data conflicts between an update transaction and a mobile transaction

Suppose the update transaction, U, will update data item d_5 and then data item d_2, and a mobile transaction, MT, wants to read d_2 and d_5. If the schedule is:

i) Broadcast transaction (BT) broadcasts d_2
ii) MT reads d_2
iii) U updates d_5
iv) U updates d_2
v) Broadcast transaction (BT) broadcasts d_5
vi) MT reads d_5

The mobile transaction MT may observe inconsistent data values. The serialization graph is cyclic such as MT $\rightarrow$ U$\rightarrow$ MT. Thus, it is non-serializable. The reason is that MT reads a data item, d_2, which is in conflict with U before the update from U. However, it reads another conflicting data item, d_5, after the update from U.

Example 2: A mobile transaction conflicts with two (or more) update transactions

Even though the serialization order between an update transaction and a mobile transaction is not cyclic, the final serialization graph can still be cyclic due to transitive dependencies. Suppose there are two updates U_1 and U_2 such that U_1 will update d_2 and d_1, and U_2 will update d_1 and d_5. If the schedule is:

i) Broadcast transaction (BT) broadcasts d_2
ii) MT reads d_2
iii) U_1 updates d_2
iv) U_1 updates d_1
v) U_2 updates d_1
vi) U_2 updates d_5
vii) Broadcast transaction (BT) broadcasts d_5
viii) MT reads d_5

The serialization graph is cyclic such as $U_2 \rightarrow$ MT $\rightarrow U_1 \rightarrow U_2$.

Example 3: Non-serializability involving two or more broadcast transactions.

A mobile transaction may not be able to find all its required data items in a single broadcast cycle. Non-serializability may occur over more than one broadcast transaction. For example if the schedule is:

i) BT_1 broadcasts d_2
ii) MT reads d_2
iii) End of BT_1
iv) U updates d_5
v) U updates d_2
vi) BT_2 broadcasts d_5
vii) MT reads d_5

The serialization graph is cyclic such as MT$\rightarrow$ U $\rightarrow$ MT.

Note that in the determination of serializability of the serialization graph, we ignore the broadcast transactions as they are considered as “intermediate transactions”. They do not have any real effect on the database consistency. Their only function is to provide data items for the mobile transactions to read. (The reason of treating it as a transaction is to facilitate the analysis. Since the mobile transactions read data items from broadcast transactions, their serialization orders are always BT $\rightarrow$ MT.)

The data inconsistency problems in the first two examples can be resolved by using a serial execution of the broadcast and update transactions. That is they are executed one after one and no concurrent of execution. For example each transaction, both broadcast and update, will be defined with a unique time-stamp based on their arrival time. Their execution order is then following their time-stamps. In this way, the final schedule will be serial. However, this serial approach has two main problems. Firstly, the concurrency control of system will be greatly affected. Note that the waiting time of the update transactions can be very long as the broadcast transactions are long transactions and this may affect the validity of the data items. Secondly, even if the serial approach is used, the problem in the third example is still unresolved.

5 Update-first with Order (UFO) Algorithm

The biggest problem in implementing concurrency control in data broadcast is that the mobile transactions may read a broadcast data items at any time while it is being broadcast and the server in general does not notice this. Thus, instead of detecting data conflicts among mobile transactions and the update transactions, the UFO algorithm checks data conflicts among the broadcast transactions and update transactions.

The basic principle of the UFO algorithm is to ensure that if data conflicts occur between a broadcast transaction and an update transaction, the serialization order between them will be U → BT. (BT → U will only be allowed if it is impossible for a mobile transaction, which reads from BT, will dependent on U directly or transitively[2], e.g., all the existing conflicts between the broadcast transaction and the update transaction are BT → U, and the mobile transaction has been completed before the start of the update transaction.) Since mobile transactions read data items from broadcast transactions, their serialization orders are always BT → MT. Thus, serialization order between the update transactions and the mobile transaction will be U → MT and the final serialization graph among the mobile transactions and update transactions will be serializable. (The correctness of the UFO algorithm will be discussed in details in section 5.4.)

5.1 Execution of Update Transactions

In the UFO algorithm, it is assumed that a well-formed concurrency control protocol such as two phase locking [BHG87] is adopted for resolving data conflicts between the update transactions. The execution of an update transaction is divided into two phases: the execution phase and update phase. Permanent database update is done during the update phase. The purpose of maintaining a short duration for update is to reduce the blocking time on the broadcast transactions by the update transactions. During the update phase, the broadcast of the data items will be stopped.

During the execution phase, an update transaction will be executed and data conflicts with other update transactions will be resolved according to the adopted concurrency control protocol. The updates of the data items are written in a private workspace of the transaction during the execution phase. When all operations of a transaction have been executed, it enters the update phase in which permanent updates to the database will be performed by copying the new values from the private workspace into the database.

[2] The case will be explained in detail in section 5.3.

5.2 Details of the UFO Algorithm

Before an update transaction starts its update phase, the system detects data conflict between the update transaction and the broadcast transactions in the current and previous broadcast cycles. At the start time of the update phase, the set of data items to be updated by the update transaction will be known as all its operations have been completed. At the same time, the set of the data items to be read (broadcast) by a broadcast transaction is also known as it is resulted from a broadcast algorithm [LS97]. Their sets of data items will be compared. If they are overlapped, there is a data conflict. The conflict resolution methods used to resolve the data conflict are dependent on when the update phase starts with respect to the broadcast of the conflicting data items. If it is before the broadcast, no action needs to be done as the serialization order will be U → BT. The only problematic case is that the broadcast of the conflicting data items is before the arrival of the update transaction. In the UFO algorithm, this will be resolved by re-broadcast, as we will discuss in section 5.2.2.

5.2.1 Broadcast Transactions

A problem we need to resolve in detecting data conflict is that a mobile transaction may span more than one broadcast cycle (broadcast transaction), e.g., it reads from more than one broadcast cycle as in the case of example 3 in section 4. Thus, checking data conflict with the current broadcast transaction is not sufficient. The most straight forward way to solve the problem is to include broadcast transactions in the previous cycles for data conflict checking. However, how many cycles be checked? If too many cycles need to be checked, the overhead might be too heavy and may not be practical. It turns out the drop period, which is defined as the deadline of a mobile transaction minus its arrival time, can be used to estimate how many cycles to check:

Number of previous cycles to be checked, n_c
$= DP_{max}$ / time for one broadcast cycle

where DP_{max} is the maximum drop period of all mobile transactions from the mobile clients.

Due to the low bandwidth, the time required to complete one broadcast cycle is quite long. For example, if it takes 0.05 second to broadcast one data item, then 10 seconds is required to broadcast 200 data items. If we set the drop period to be 30 seconds, then we just need to check three more cycles.

5.2.2 The UFO Algorithm

In this section, we will state the UFO algorithm formally. First we define the following quantities.

$$BT = \sum_{j=0}^{n_c} BT_{i-j}$$

for any current broadcast cycle i

O_{BT} = set of data items of broadcast transaction, BT

O_U = set of data items of update transaction,U
$B_A = \{x \in O_{BT} \cap O_U \mid x$ is already broadcast when U arrives$\}$

Before the permanent update starts, the following algorithm is performed:

```
If    O_BT ∩ O_U = { }
Then      BT and U have no dependency
Else
      If      B_A = { }
      Then  the serialization order is U → BT
      Else
            For each data item i ∈ B_A
                  re-broadcast data item i
            Next
            the serialization order  is U → BT
      End If
End If
```

We will now give some examples on the UFO algorithm and then discuss the properties as well as the correctness of the algorithm.

5.3 Examples

Referring to the examples in section 4, the serialization graphs will be acyclic under the UFO algorithm.

Example 1:

i) Broadcast transaction (BT) broadcasts d_2
ii) MT reads d_2
iii) Compare the data sets of U and BT
iv) U updates d_5
v) U updates d_2
vi) BT re-broadcast d_2
vii) MT reads the most updated value of d_2
viii) BT continue it process and broadcasts d_5
ix) MT reads d_5

The serialization graph is acyclic such as U→ MT.

Example 2:

i) BT broadcasts d_2
ii) MT reads d_2
iii) Compare the data sets of U_1 and BT
iv) U_1 updates d_2
v) U_1 updates d_1
vi) BT re-broadcasts d_2
vii) MT reads d_2 again
viii) Compare the data sets of U_2 and BT
ix) U_2 updates d_1
x) U_2 updates d_5
xi) BT continues its process and broadcasts d_5
xii) MT reads d_5

The serialization graph is acyclic such as $U_1 \rightarrow U_2 \rightarrow$ MT

Example 3:

i) BT_1 broadcasts d_2
ii) MT reads d_2
iii) End of BT_1
iv) Start of BT_2
v) Compare the data sets of U and BT_1 and BT_2
vi) U updates d_5
vii) U updates d_2
viii) BT_2 re-broadcasts d_2
ix) MT reads d_2 again
x) BT_2 broadcast d_5
viii) MT reads d_5

The serialization graph is acyclic such as U → MT.

5.4 Properties, Correctness and Implementation of the UFO algorithm

5.4.1 Properties

Two important properties of the UFO algorithm are: (1) its impact on the adopted broadcast algorithm is minimal; and (2) it can ensure that all the schedules among the mobile transactions from mobile clients and update transactions are serializable.

The implementation of the UFO algorithm is simple and does not require any changes in the mobile clients. The mobile transactions can read a data item whenever it is being broadcast without asking the server for permission. All the checking will be done at the server. The overhead for the data conflict detection at the server should be low. We note that the only overheads of the algorithm are:

(1) the division of the execution of update transactions into two phases;
(2) the checking of the data sets of the broadcast transactions and the update transaction whenever an update transaction wants to start its update phase.;
(3) the broadcast process has to be stopped while an update transaction is in the update phase; and
(4) the re-broadcast of the conflicting data items in case the broadcast of the conflicting data items occurs before the update phase of the update transaction starts.

Dividing the execution of update transactions into two phases is trivial and should not incur heavy overhead. (This is similar to the deferred update approach [BHG87].) The overhead for checking conflicting data sets should be low as the number of data items to be updated by an update transaction is usually small. To speed up the checking process, the data items to be updated may be sorted according to their IDs. The waiting time of the broadcast transaction for the update transaction will not be too long as the length of the update transactions should be very short (remember that their execution is divided into two phases). The number of re-broadcast should also be very small since the probability of data conflict is low. Even if there is a data conflict, re-broadcast is only required for the case where the conflicting data items are broadcast before the update phase of the update transaction causing the data conflict.

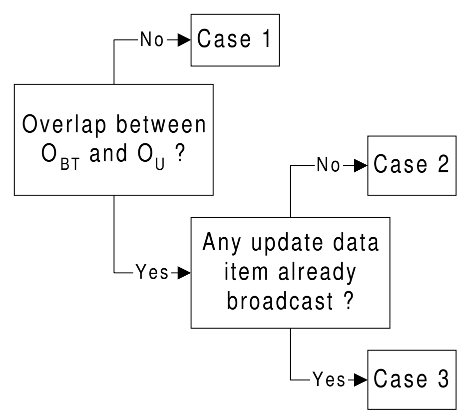
Other than data re-broadcast, the UFO algorithm does not affect the basic mechanism of the underlying broadcast algorithm. Furthermore, an advantage of the re-broadcast is that every time a data item is being broadcast, it will be the most updated version.

5.4.2 Correctness

We have seen how the UFO algorithm has low overhead and minimal impact on the broadcast mechanism. Now, we will prove that using the UFO algorithm, the serialization graph among the mobile transactions and the update transaction is always acyclic. In the proof, we will concentrate on the serializability between the update transactions and the broadcast transactions. If their serialization order is U $\rightarrow$ BT, the serialization among the mobile transactions and the update transactions will be U $\rightarrow$ MT since mobile transactions read data items from BT.

In the proof below, all cases regarding the data conflict between BT and U will be considered. These cases corresponding to different sections of the algorithm defined in section 5.2.2. In each case, the serialization graph (SG) will be considered. SG = (N, E) where N is the set of nodes representing the set of transactions. Since the only transactions for all cases is BT and U, therefore N = {BT, U}. E is the set of ordered pairs (T_i, T_j) representing that there is at least one pair of conflict operations between T_i and T_j and the operation in T_i precedes the one in T_j. In our presentation, $T_i \longrightarrow T_j$ will be used to represent the order pairs (T_i, T_j).

Cases that may occur are summarized in the following diagram:



Case 1

This case corresponds to the 'if-then' part of the first level if-then-else loop in 5.2.2.

In this case there is no overlap between the sets of data items of BT and U. i.e. $O_{BT} \cap O_U = \{\}$. Therefore there is no conflict operations between BT and U, and E = {}. As a result SG is an unconnected graph, with two nodes BT and U, and hence is acyclic.

Case 2

This case corresponds to the 'if-then' part of the second level if-then-else loop in 5.2.2.

In this case there is an overlap between the sets of data items of BT and U, i.e. $O_{BT} \cap O_U \neq \{\}$. However $\forall x \in O_{BT} \cap O_U$, x is not broadcast yet when the update phase starts. Therefore $w_U(x) \longrightarrow r_{BT}(x)$ and E = {U $\longrightarrow$ BT}. The resultant SG is again acyclic.

Case 3

This case corresponds to the 'else' part of the second level if-then-else loop in 5.2.2.

In this case, there is an overlap between the sets of data items of BT and U. i.e. $O_{BT} \cap O_U \neq \{\}$. In addition, $\exists x \in O_{BT} \cap O_U$ such that x is already broadcast when the update phase starts. At this point in time, E = {BT $\rightarrow$ U} since $\forall x \in B_A$, $r_{BT}(x) \rightarrow w_U(x)$. Therefore if a mobile transaction commits on or before this point in time, SG is acyclic.

If the broadcast transaction continues its original broadcast schedule after the update phase, then U $\rightarrow$ BT will be added to E since $\forall x \notin B_A$, $w_U(x) \rightarrow r_{BT}(x)$. i.e. E = {BT $\rightarrow$ U, U $\rightarrow$ BT} which is a cyclic schedule. According to the algorithm in 5.2.2 $\forall x \in B_A$, x will be re-broadcast at once after the update phase to clear the dependency BT $\rightarrow$ U. Therefore E = {U $\rightarrow$ BT} and the resultant SG is again acyclic.

5.4.3 Reducing the Number of Data Re-Broadcast

Although under normal situations, the number of data re-broadcast due to the data conflicts should be small, the cost still can be expensive if the sizes of the conflicting data items are very large such as video files. One way to reduce the number of data re-broadcast is to re-broadcast a data item only if the conflicting update transaction has other data conflicts with broadcast transactions or other update transactions. The reason is that if re-broadcast is not performed, the serialization order will be BT $\rightarrow$ U. Thus, there is a risk of having a cyclic serialization graph if the update transaction conflicts with other transactions. However, if there is no further data conflict involving the update transaction, the serialization graph will not be cyclic.

When there is a data conflict between a broadcast transaction and an update transaction, and the size of the conflicting data item is very large, the serialization graph of the conflicting update transaction with other update transactions and broadcast transaction will be maintained and checked for the period equal to the maximum drop period of the mobile transactions. If up to that time the set of data items updated by the update transaction has not been updated by any other update transactions and no broadcast transaction "reads" these data items, the re-broadcast can be cancelled and the graph can be destroyed. Otherwise, re-broadcast has to be done immediately.

5.4.4 Implementation

The UFO algorithm can be adopted on different broadcast models. We take two models, a broadcast disk model which only has only one downlink broadcast channel and a hybrid model which has two downlink channels, one for periodic broadcast and one for on-demand data items, as examples and describe how UFO algorithm can be applied.

5.4.4.1 Broadcast Disk Model

In this model, the server broadcast data items periodically. The broadcast schedule is computed at the beginning of each broadcast cycle. The UFO algorithm will be implemented with a separate update process. Execution of each update transaction is divided into two phases, the execution phase and the update phase. The update process will interrupt the broadcast process when the update phase began. Then the UFO algorithm will be execute and any conflict data items will be identified and re-broadcast.

5.4.4.1 Hybrid Model

In this model, there are two downlink channels. One channel is for periodic broadcast and the other one for on-demand requested data items. The UFO algorithm will again be implemented with a separate update process as in the broadcast disk model. When an update transaction begin its update phase, it will interrupt both the periodic broadcast process and the on-demand broadcast process for data conflict checking with the use of the UFO algorithm. If there is data conflict with either broadcast schedule, then re-broadcast will take place on the corresponding channel.

6 Conclusions and Future Work

The Update First with Order (UFO) algorithm is an efficient way for concurrency control in a broadcast environment. It has minimal impact on server, client and the broadcast schedule. The server is only required to check if any update data items of a transaction falls in the already broadcast set of a broadcast schedule. If it is the case, the server will re-broadcast the data items at once after the update to refresh client data items. On the client side, it is only required to listen to the broadcast schedule and capture any item required when it is broadcast or re-broadcast. On the side of the broadcast schedule, it may occasionally need to include some conflict data items for re-broadcast. So overhead in both processing and bandwidth are minimal. In addition, using this algorithm, all clients and server will have the same view on the serial order of update transactions.

There are a few threads for future work. The first one is to apply the UFO algorithm in the environment where order of read operations at client should be taken into account. The second one is to allow write operations in a mobile transaction. In this case, the detection of data conflict will be much more complicated. The third one is to integrate the algorithm with those broadcast approaches that have information such as index tree and signature on the structure of the broadcast data items. In those cases mobile clients may go to doze mode and then wake up again to capture data items according to the information about the broadcast schedule they received beforehand. Re-broadcast any data item during a broadcast schedule will obsolete all those information. Therefore the UFO algorithm may have to be modified for those cases.

Another issue is we have taken assumption that data conflict between update transactions and read-only mobile transactions is low. However for high data conflict, the number of re-broadcast would be significant and have much impact on the broadcast model. In this case, the UFO algorithm should be enhanced to minimize bandwidth required for re-broadcast. One possibility is to broadcast invalidation message for the update item instead of re-broadcast the update item itself in order to save bandwidth.

References

[AFZ97] Acharya, S., Franklin, M., Zdonik, S., "Balancing Push and Pull for Data Broadcast", in *Proceedings of ACM SIGMOD*, Tucson, Arizona, May 1997.

[BHG87] Bernstein, P.A., Hadzilacos, V., Goodman, N., Concurrency Control and Recovery in Database System, Addision-Wesley Publishing Company, 1987.

[FZ98] Franklin, M. and Zdonik, S., "Data In Your Face": Push Technology in Prospective", in *Proceedings of 1998 ACM SIGMOD Conference*, Seattle, 1998.

[HV97] Hameed, S. and Vaidya, N.H., "Efficient Algorithms for Scheduling Single and Multiple Channel Data Broadcast", Technical Report 97-002, Department of Computer Science, Texas, A&M University, Feb. 1997.

[IB94] Imielinski, T. and Badrinath, B.R., "Mobile Wireless Computing: Challenges in Data Management," *Communications of the ACM*, vol. 37, no. 10, Oct. 1994.

[IVB97] Imielinski, T., Viswanathan, S.and Badrinath, B.R., "Data on Air: Organization and Access", *IEEE Transactions on Knowledge and Data Engineering*, vol 9, no. 3, pp. 353-372.

[LS95] Leong, H.V. and Si, A, "Data broadcasting strategies over multiple unreliable wireless channels", in *Proceedings of the 4th International Conference on Information and Knowledge Management*, pages 96-104, November 1995.

[LS97] Leong, H.V. and Si, A., "Database Caching over the Air-Storage", The Computer Journal. Volume 40, number 7, pages 401-415, 1997.

[PB93] Pitoura, E. and Bhargava, B. "Dealing with Mobility: Issues and Research Challenges," *Technical Report*, Purdue Univ., Nov. 1993.

[PB95] Pitoura, E. and Bhargava, B. "Maintaining Consistency of Data in Mobile Distributed Environment," in *Proceedings of the 15th International Conference on Distributed Computing Systems*, pp. 404-413, 1995.

[PS98] Pitoura, E. and Samaras, G., *Data Management for Mobile Computing*, Kluwer Academic Publishers, 1997.

[SNSR98] Shnmugasundram, J., Nithrakashyap, A., Sivasankaran, R., Ramamritham, K., "Efficient Concurrency Control for Broadcast Environments," Technical Report, 1997-062, Department of Computer Science, University of Massachusetts, Amherst, 1998.

[XSGFR97] Xuan, P., O. Gonzalez, J. Fernandez & Ramamritham, K., "Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments", in *Proceedings of 3rd IEEE Real-Time Technology Application Symposium*, 1997.