A THEORY OF CURRENCY AND CONSTISTENCY IN BROADCAST DATABASES

E. Pitoura, P.K. Chrysanthis, K. Ramamritham

26-99

Preprint no. 26-99/1999

Department of Computer Science
University of Ioannina
451 10 Ioannina, Greece

A Theory of Currency and Consistency in Broadcast Databases

Evaggelia Pitoura* Panos K. Chrysanthis† Krithi Ramamritham‡

Abstract

In this paper, we develop a general theory of currency and consistency for an extended client/server environment in which the server broadcasts items of interest to a large number of clients. Such environments are part of an increasing number of emerging applications in wireless mobile computing systems; they also provide a scalable means to deliver information in web-based computing applications, for example in publish-subscribe systems. We introduce various criteria of currency and consistency and present a framework to precisely define protocols for enforcing such criteria. We then show how the various protocols introduced in the literature fit in our framework and how new protocols can be advanced by varying the parameters of the framework.

1 Introduction

While traditionally data are delivered from servers to clients on demand, a wide range of emerging database applications benefit from a broadcast mode for data dissemination. In such applications, the server repetitively broadcasts data to a client population without a specific request. Clients monitor the broadcast channel and retrieve the data items they need as they arrive on the broadcast channel. Such applications typically involve a small number of servers and a much larger number of clients with similar interests. While the concept of broadcast delivery is not new [3, 26, 11], data dissemination by broadcast has recently attracted considerable attention ([14], [22]), due to the physical support for broadcast provided by an increasingly important class of networked environments such as by most wireless computing infrastructures, including cellular architectures and satellite networks. The explosion of data intensive applications and the resulting need for scalable means for providing information to large client populations are also motivated by the dramatic improvements in global connectivity and the popularity of the Internet [9, 28].

As such systems continue to evolve, more and more sophisticated client applications will require reading current and consistent data despite of updates at the server. In most current research (for example, [6], [2], [13], and [17]), updates have been mainly treated in the context of caching with no transactional semantics. Transactions and broadcast were first discussed in the Datacycle project [11] where special hardware is used to detect changes of values read and thus ensure consistency. The Datacycle architecture was extended in [4] for the case of a distributed database where each database site broadcasts the contents of the database fragment residing at that site. More recent work includes among others the F-matrix technique [24] for enforcing a relaxed form of serializability; the deployment of the broadcast medium for transmitting concurrency control related information to the clients so that part of transaction management can be undertaken by the clients [5]; the SGT method [20] that is based on serialization graph testing, maintaining multiple versions per item [19] and the BCC method [18] that is based on timestamps.

[1]Computer Science Department, University of Ioannina, GR45100 Ioannina, Greece, pitoura@cs.uoi.gr
[2]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, panos@cs.pitt.edu
[3]Computer Science Department, Indian Institute of Technology, Bombay, krithi@cse.iitb.ernet.in

Our work in this paper is different from previous work in that instead of proposing specific protocols for broadcast databases, we develop a general theory for correctness in such settings. Our model provides the necessary tools for arguing about the correctness and other properties of the various protocols. In addition, it provides the basis for the advancement of new protocols. An interesting feature of our model that is applicable to non-broadcast systems as well is the distinction made between currency and consistency. Currency can be enforced in non-transactional systems. We relate currency and consistency through the definition of strict currency which is the type of consistency we get in transactional systems.

The remainder of this paper is structured as follows. In Section 2, we introduce the broadcast model, give background definitions, define various broadcast schedules based on currency, and set the assumptions for our theory. In Section 3, we define various models of consistency that provide clients with transaction consistent database snapshots. In Section 4, we present our currency model that is based on two different types of currency, strict and oldest value currency. In Section 5, we present our currency control theory that defines the way server communicates updates to their clients. In Section 6, we present our consistency control theory based on the Read-Test theorem. Finally, in Section 7, we show how the various protocols introduced in the literature fit in our model, and in Section 8 we present our conclusions and plans for future work.

2 The Problem

2.1 The Broadcast Model

In our setting, a server periodically broadcasts data items from a database to a large client population. Each period of the broadcast is called a broadcast *cycle*. The items to be broadcast at each cycle are determined by a broadcast program and depend on the client access preferences. Each client listens to the broadcast and fetches data as they arrive. This way data can be accessed concurrently by any number of clients without any performance degradation. However, access to data is strictly sequential, since clients need to wait for the data of interest to appear on the channel. Clients do not need to listen to the broadcast continuously. Instead, they may tune-in to read specific items. Such selective tuning is important especially in the case of portable mobile computers, since they most often rely for their operation on the finite energy provided by batteries and listening to the broadcast consumes energy. However, for selective tuning, clients must have some prior knowledge of the structure of the broadcast that they can utilize to determine when the items of interest appear on the channel. Alternatively, the broadcast can be self-descriptive, in that, some form of directory information is broadcast along with data. Techniques for broadcasting index information along with data are given for example in [16, 13].

In this paper, we consider the case of data being updated at the server. Our goal is to provide consistency and currency guarantees for the data read by clients. Our protocols take into account the following characteristics of broadcast systems. First, such systems are characterized by an asymmetry between the communication capacity from the server to the clients and the communication capacity of the backchannel from the client to the server. This asymmetry is the result of the huge disparity of the transmission capabilities of clients and servers as well as the scale of information flow since a very large number of clients is connected to a single server. Furthermore, for scalability reasons, it is important to limit the clients requests to the server to avoid overwhelming it. These reasons as well as decreasing the latency of client transactions justify enforcing currency and consistency at the client. In particular, these considerations motivate the following model assumptions:

1. The server is stateless in that it does not maintain any particular control information about its clients.

2. All updates are performed at the server and disseminated from there.

3. To get currency and consistency related information, clients do not contact the server directly instead such control information is broadcast along with data.

We make no specific assumptions about concurrency control at the server other that the schedule of server transactions is serializable.

2.2 Currency of Broadcast Data

Assume that a client makes a request to read an item x at time t_r and then waits for the item to appear on the channel say at time t_c. The value of x that the client reads at t_c is the value placed on the channel at t_c - d, where d is the communication delay. Minimizing the time $t_c - t_r$ that the client has to wait to get the item of interest is the goal of broadcast scheduling that is of designing broadcast programs (for example, [1]). Such broadcast programs may also take into account timing constraints, for example the fact that a client may need to read particular data items before a deadline [7, 27].

The work in this paper is complementary to broadcast scheduling. In fact, our theory is independent of the particular schedule in use. The question we address is, given a broadcast program, what kind of currency and consistency guarantees can be provided for the values read by clients and what is a general framework for enforcing them. Thus, when we talk about the time of a read operation we refer to time t_c. Let us consider first, what value is placed on the broadcast for an item x. Our first assumption is that only values produced by committed transactions appear on the broadcast. Besides that, there are two reasonable choices.

- *Latest value scheduling:* When an item x is scheduled to appear on the broadcast at time t, the value that the server places on the broadcast channel at time t is the most recent value of x (that is the value of x produced by all transactions committed at the server by t).
- *Periodic update scheduling:* Data values on the broadcast change at pre-specified intervals called *currency intervals*. In particular, the value of item x that the server places on the broadcast at time t is the value of x produced by all transactions committed at the server by the beginning of the current currency interval (this may not be the current value of x at the server). Let c be the duration of the currency interval. For instance, c may be equal to the duration of a broadcast cycle. In this case, the value broadcast for each item during the cycle is the value of the item at the server at the beginning of the cycle. In periodic update scheduling, we may need to keep two versions for some items.

For a uniform treatment of latest value and periodic update scheduling, we assume that a latest value schedule is a periodic update schedule with $c = 0$.

2.3 The Readset of a Transaction

The *readset* of a transaction R denoted $RS(R)$ is the set of items it reads. In particular,

Definition 1 (Readset) *The readset of a transaction R, denoted $RS(R)$, is the set of ordered pairs of data items and their values that R read. The readset always includes only one pair (item, value) per item. In the case in which a transaction reads the same item more than once, the readset includes the value read last.*

A database state is typically defined as a mapping of every data to a value of its domain.

Definition 2 (Database State) *A databases state, denoted DS, is a set of ordered pairs of data items and their values.*

In a database, data are related by a number of integrity constraints that express relationships of values of data that a database state must satisfy.

Definition 3 (Consistent Database State) *A database state is consistent if it does not violate the integrity constraints [8].*

Definition 4 (Consistency of a Subset of a Database State) *A subset of a database state is consistent if it is a subset of a consistent database state [23].*

A readset is a subset of database state. We define two properties for the readset:

- *currency*: how current are the values in the readset with respect to the values at the server
- consistency: whether the readset is a subset of a consistent database state, i.e., whether it satisfies the integrity constraints.

In periodic update scheduling, the content of the broadcast at each currency interval is a subset of the database state at the server as of the beginning of the interval. We would like to ensure that the content of the broadcast at each currency interval is consistent. Note, that in the case of latest value scheduling, this trivially holds. One way to ensure consistency is by enforcing the content of each currency interval to be a subset of a database state produced from the serializable execution of a number of transactions. This is the case with strict schedules. A schedule is *strict* if the commit order of transactions is compatible with the serialization order. Thus, if at each currency interval, the values broadcast are the values produced by all transactions committed at the server by the beginning of the currency interval, then the content of the broadcast during each interval is a subset of the database state produced from the serializable execution of these transactions. Thus:

Proposition 1 *In the case of strict schedules, the content of the broadcast at each currency interval is consistent.*

There is a trade-off between currency and consistency. The largest the currency interval, the worst the currency; the largest the probability to read all data from the same currency interval and thus from the same consistent database state.

Let t_{begin} and t_{commit} be the time R performs its first read operation and the time it commits respectively.

Definition 5 (Lifetime) *The* lifetime *of R, denoted $lifetine(R)$ is the time interval $[t_{begin}, t_{commit}]$.*

3 Models of Consistency

In general, a client transaction R reads values that correspond to different currency intervals and thus its readset may not be consistent even in the case of strict schedules.

Definition 6 (Readset Consistency Requirement) *The* consistency *requirement for a read only transaction R is expressed as follows: $RS(R) \subseteq DS$, where $RS(R)$ is the readset of R and DS is a consistent database state.*

Various consistency guarantees stronger than correspondence to a consistent database state have been defined based on serializability [10, 15, 25]. All guarantee that a read-only client transaction sees a consistent database state. In Table 1, we survey such criteria. For each definition of consistency,

C4: All read-only trans are serialized with the set of server trans and observe a serial order of server trans that agrees with the order in which the server trans committed	Strict Consistency [10]
C4-I: individual trans version of C4	
C4-S: subset of read-only trans version of C4	
C3: All read-only trans are serialized with the set of all server trans	Strong Consistency [10] Serializability [25]
C3-I: individual trans version of C3	Weak Consistency [10]
C3-S: subset of read-only trans version of C3	Strong Consistency [15] (a strong read-only trans is serialized with all server trans and the other strong
C2: All read-only trans are serialized with the set of server trans that produced values that are seen (either directly or indirectly) by them	
C2-I: individual trans version of C2	Update Consistency [10] Weak Consistency [12] External Consistency [25]
C2-S: subset of read-only trans version of C2	
C1: The values read by read only trans correspond to a consistent database state	Weak Consistency [15] Consistency [25]
C0: No consistency requirements	No Consistency [15] Opportunistic [2]

Table 1: Definitions of consistency

there are three versions. The first is the strongest one and requires serializability of *all* read-only client transactions with server transactions. This means that there is a global serialization order including all read-only client transactions and a subset of server transactions. The second version requires serializability of each read-only client transaction *individually*. The last version requires serializability of some *subset* of client read-only transactions. This subset may for example consist of all transactions at a given client site.

Note that if all read-only client transactions are C4-I consistent then C4 and C4-S consistency also hold. The global serialization order is compatible with the commit order of server transactions and the order in each C4-I schedule since the read-only transactions do not conflict with each other.

The database state DS of Definition 6 seen by a read-only client transaction is not necessarily a database state that appears at the server as the following example shows. We use $||$ to denote the beginning of each currency interval. The notation w_i, r_i and c_i stands respectively for the write, read and commit operation of transaction T_i.

Example 1 *Assume two server transactions T_1 and T_2, one client read-only transaction T_R and the following schedule:*

$r_R(x) \;||\; w_1(x)c_1w_2(y)c_2 \;||\; r_R(y)c_R$

R sees the results of T_2 but not of T_1. □

The example above also shows that the assumption that the schedule of all server transactions is strict does not mean that the schedule including one or more read-only client transactions is also strict. The commit order of server transactions T_1 and T_2 ($T_1 \rightarrow T_2$) is compatible with the serialization order of the schedule of server transactions. However, the commit order of the schedule including the read-only transaction T_R is not compatible with the serialization order ($T_2 \rightarrow T_R \rightarrow T_1$).

4 Models of Currency

Each client sees a snapshot of the database state. We are interested in how current this state is. We first define the currency of a single item read by a client.

Definition 7 *(**Currency of an Item**) The currency of an item x in the readset of a transaction R, denoted Currency(x, R), is the commit time of the transaction that wrote the value of x read by R.*

Let DS_t be the database state at the server at time t. In the case that the client sees an actual database, we define strict currency as follows

Definition 8 (Strict Readset Currency) *In the case that the readset RS(R) of a read-only transaction R corresponds to an actual database state DS_t at the server at some time t ($RS(R) \subseteq DS_t$) the strict currency of transaction R, denoted Strict_Currency(R), is t.*

R sees data values as they existed at the server at some previous given point t. In other words, R sees the results of all transactions committed prior to t and not after t; that is R is a *t-vintage* [15] transaction. Let $t_0 \in lifetime(R)$. We define currency relative to t_0.

Definition 9 (Relative Strict Readset Currency) *In the case that the readset RS(R) of a read-only transaction R corresponds to an actual database state, strict currency of R with respect to time $t_0 \in lifetime(R)$, denoted $R\_Strict\_Currency(R, t_0)$, is the time lag: $t_0 - Strict\_Currency(R)$.*

We say that R is *strictly current* with respect to time t_0 if $R\_Strict\_Currency(R, t_0) = k + d$ where (a) if $c > 0$, k is such that let $mc \leq t_0 - d < (m+1)c$, for some m, $k = t_0 - d - mc$ and (b) if $c = 0$ (latest value update scheduling), $k = 0$. That is when R is strictly current with respect to t_0, R sees data as they existed at time t_0 at the server subject to communication delays (d) and the duration (c) of the currency interval. A transaction R sees the latest values when $t_0 = t_{commit}$, that is $R\_Strict\_Currency(R, t_{commit}) = k + d$. The best such currency is attained when $c = 0$, that is with latest value scheduling.

If the readset of a transaction R is not a subset of a single database state DS at the server, then R may not be a *t-vintage* transaction. Instead it can be a *t-bound* [15] transaction: a transaction that sees the results of all transactions committed prior to t but also of some transactions committed after t. In this case, we can only provide currency guarantees per item.

Definition 10 (Oldest-Value Readset Currency) *The oldest-value currency of R, denoted OV_Currency(R), is the largest t such that:*

$$\forall\ (x, v) \in RS(R),\ Currency(x, R) \geq t$$

Definition 11 (Oldest-Value Relative Readset Currency) *The oldest-value currency of R with respect to time t_0 of R is the time lag: $t_0 - OV\_Currency(R)$.*

Similarly we can define a transaction as being oldest-value current with respect to time t_0. Oldest-value currency reduces to strict currency when the readset of a transaction corresponds to an actual database state.

The above requirements focus on a single client transaction. In general, multiple client transactions may originate from the same or different client hosts. We would like to ensure that if the execution of a read-only transaction R_2 follows the execution of a read-only transaction R_1, then the values read by R_2 are more current than the values read by R_1.

Let e be a distinct event in the lifetime of a transaction, for instance e may be the first read operation or the commit point. Let $time(e, R)$ be the time that the event e of transaction R occurs.

Definition 12 (Precedes) *Transaction R_1 precedes R_2 with respect to event e, if $time(e, R_1) < time(e, R_2)$.*

Then, we say that

Definition 13 (Temporal Order) *The temporal order between transactions R_1 and R_2 with respect to event e is preserved if whenever one, say R_1, precedes the other with respect to event e, $Currency(R_1) \leq Currency(R_2)$.*

Proposition 2 *If R_1 is strictly current with respect to $time(e, R_1)$ and R_2 is strictly current with respect to $time(e, R_2)$ then the temporal order between R_1 and R_2 with respect to event e is preserved.*

5 Currency Control

It is easy to see that:

Proposition 3 *If a read-only transaction R reads items from the broadcast channel (without any additional information), then RS(R) is oldest-value current with respect to the time that transaction R performed its first read operation.*

To achieve better currency, the server must inform the client of updates by broadcasting a report that includes information about the items that have been updated. This report may be sent immediately after the update is performed or periodically. In the former case, the client must listen to the broadcast continuously. In the latter case, the report may be broadcast at the beginning of each interval or at other points. Let us assume that the report includes a list of the items that have been updated and that it is broadcast at the beginning of each currency interval.

Theorem 1 (The Invalidation Theorem) *Let $(x, u) \in RS(R)$, x be read at time t_x and $t_{xd} = t_x$ - d. Let IR_m be the report that was broadcast at time m and IR be the set of all such reports that were broadcast during the lifetime of R, $IR = \{IR_m : m + d \in lifetime(R)\}$. R is strictly current with respect to time t_k, iff:*

(a) if $t_{xd} \leq t_k$ then $x \notin IR_m$, $t_{xd} < m \leq t_k$, and

(b) if $t_{xd} > t_k$ then $x \notin IR_m$, $t_k < m < t_{xd}$

Proof. In the Appendix.

An interesting case is when $t_k = t_{commit}$. Then, to get strict currency with respect to commitment, an item read at t_x must not appear in any subsequent report. Another interesting case is when $t_k = t_{begin}$, i.e., the client gets the view as of its first read. Then each item read at time t_k must not have been updated prior to t_k and after t_{begin}.

Corollary 1 *To achieve oldest value currency, it suffices to ensure condition (a) of the Invalidation Theorem.*

Invalidation Reports. Consider a report that includes just a list with the (identifiers of the) items that have been updated. We describe next a protocol to achieve strict currency with respect to t_k based on the Invalidation Theorem. The client tunes in and reads the invalidation reports. Two lists are maintained, an *Invalidation_Set* list that includes the items that appeared in an invalidation report broadcast after t_k and the *Read_Set* list that includes the items read so far. The protocol runs in two phases. Up to time t_k, when a client reads a new invalidation report IR_m ($m < t_k$), it checks whether an item in *Read_Set* appears in IR_m and aborts R, if it does. After t_k, for each item read, the client checks whether it appears in the *Invalidation_Set*, and aborts R, if it does.

Propagation Reports. Another possibility is for the report to include both the items updated and their new values. We call such reports *propagation reports*. An item may be re-read from the propagation report.

Algorithm 1 *Let $(x, u) \in RS(R)$, x be read at time t_x and $t_{xd} = t_x$ - d. Let PR_m be the propagation report that was broadcast at time m and PR be the set of all such reports that were broadcast during the lifetime of R, $PR = \{PR_m : m + d \in lifetime(R)\}$. The following protocol ensures that R is strictly current with respect to time t_k.*

(a) *If an item is read prior to t_k and it appears in any PR_m after t_{xd} and prior to t_k ($t_{xd} < m \leq t_k$) replace the item in the readset (i.e., re-read the item) with the value of the item that appears in the latest such propagation report (i.e., the report with the largest m).*

(b) *if an item is read after t_k and appears in any PR_m broadcast prior to t_{xd} and after t_k ($t_k < m < t_{xd}$) abort R.*

Although, strict currency is achieved, re-reading items may violate the semantics of a client program. Consider for example the following client transaction program: if $a > 0$ read(b) else read(c). What happens when the value of a was originally positive, b was read, then a was re-read and its value was found to be negative?

Autoprefetch and Multiversioning. Instead of propagating values of updated items, we can have a hybrid scheme that combines invalidation reports and propagation. One such method is called *auto-prefetch* and can be used to achieve consistency with respect to the commit point. In this case, every time an item in the *Read_Set* list appears in an invalidation report, the transaction is not aborted but it is instead marked invalidated. A read operation is re-issued for all invalidated items. The items are re-read from the broadcast channel, the next time they appear. Another method to attain consistency with respect to a time instance t_k prior to commitment is instead of broadcasting one value per item, to broadcast multiple versions, i.e., the values that the item had at some previous currency intervals. In this case, when the value of an item appears in an invalidation report, we re-read from the broadcast the version of the item that corresponds to time t_k.

6 Consistency Control

We will show first that we only need to check for violations of consistency only when a read-only transaction reads a new item.

Theorem 2 (Read-Test Theorem) *A consistency criteria may be violated only when a a read-only transaction reads a new data item if and only if the schedule of server transactions is strict.*

Proof. In the Appendix.

From the Read-Test theorem we get the following corollary about the type of the read-test.

Corollary 2 *Read-Test: Let R reads x from T and let $Follow_R$ = {T': T' overwrote an item previously read by R}. The read operation of transaction R succeeds iff there is no path from any transaction T' that overwrote an item previously read by any transaction $T' \in Follow_R$.*

In general, we need to broadcast enough information so that the client can check whether such a path occurs. The type of information depends on the type of consistency criteria enforced.

Enforcing Consistency Definitions. The Read-Test takes specific forms depending on the type of consistency we want to enforce. Let us first consider the individual-transaction version of the criteria of Table 1.

Corollary 3 *Let R reads x from T and let Follow_R = {T': T' overwrote an item previously read by R}.*

C4-I test *Let $t_{min} = min_{T' \in Follow\_R}(commit\_timestamp(T'))$ and $t_T = commit\_timestamp(T)$. The C4-I criterion is satisfied iff $t_T < t_{min}$ (then there is no path from any T' to T).*

C3-I test *The C3-I criterion is satisfied iff there is no path from any T' to T in the serialization graph that includes server transactions.*

C2-I test *The C2-I criterion is satisfied iff there is no path from any T' to T including only dependency edges in the serialization graph that includes server tarnsactions.*

An interesting case of the subset version of the consistency criteria is the one that requires serializability of all transactions at the same client. The only difference from the individual-transaction case is that the path of Corollary 2 besides server tranactions may also include other read-only client transactions.

Lemma 1 *The C4-I test is sufficient to get all versions of C4.*

Proof. In the Appendix.

If we want to enforce global serializability, that is to find a global serialization order for all client and server transactions the only enforcible criteria is C4. Otherwise, we can not ensure that the serialization orders assumed at different clients are compatible.

Theorem 3 (Global Consistency Theorem) *The only enforcible criteria that includes read-only transactions at more than one client, if we assume no communication between clients or from the server to the client, is C4.*

Proof. In the Appendix.

Consistency and Currency. Consider the C4 criterion. Let R be a client transaction that is C4 consistent. In paricular, let $t_{min} = min_{T' \in Follow\_R}(commit\_timestamp(T'))$ and $t_{max} = max_{T:R\ reads\ x\ from T}(commit\_timestamp(T))$. From the $C4-I$ test, we can see that R is serializable after the transaction T with timestamp t_{max} and before the transaction T' with timestamp t_{min}, that is $RS(R) \subseteq DS_t$, where $t_{max} < t < t_{min}$.

Proposition 4 *If C4 holds, then each read-only transaction sees an actual database state (t-vintage transaction).*

For the other criteria, we do not necessarily get t-vintage transactions as example 1 shows.

7 Case Studies

We list below a number of protocols proposed in the literature along with the type of consistency and currency that each provides. For a complete treatment refer to [21].

Invalidation Method. The invalidation method proposed in [20, 19] and in certification reports [5] (if we consider only read-only transactions) is an application of the Invalidation Theorem. The protocol works as follows. The client maintains for each read-only transaction R a list *Read_List* with the items that were read so far. The server broadcasts a report with the items that were updated since the broadcast of the previous report. If an item read (that is an item in the *Read_Set* list appears in the report, then transaction R is aborted. It is easy to see that this protocol provides strictly consistent schedules with respect to the time of the last item read. It also gives C4 consistency.

Periodic Consistency. Periodic consistency proposed in [2] focus on single read operations, there are no transactional semantics. A combination of invalidation reports with autoprefetch or propagation

reports ensure that each client reads the most current value available. The method provides oldest value currency with respect to the first value read but there is no consistency guarantees.

Versioning. With the versioning method proposed in [19], along with each item, a timestamp or version number is broadcast that corresponds to the currency interval at the beginning of which the item had the corresponding value. Let v_0 be the currency interval at which transaction R performs its first read. For each subsequent read, the read-test checks that the items read have version numbers $v \leq v_0$. If this does not hold, R is aborted. This gives strict currency with respect to the first item read and C4 consistency.

BCC-T1. The BCC-T1 method proposed in [18] provides an implementation of the C4-I test. Each transaction gets a commit timestamp. The commit timestamp of the transaction that last wrote each item is also broadcast along with the item. In addition, an invalidation report is broadcast periodically that includes for each data item x that has been updated since the previous report the pair (x, min_t) where min_t is the smallest commit timestamp among all transactions that wrote x. For each transaction we also maintain the set $Current\_RS(R)$ that includes the (item, value) pairs read by R so far and a counter *count* as follows. When for an item $(x, value) \in Current\_RS(R)$ the pair (x, min_t) appears in an invalidation report, we set *count* equal to min{min_t, count}. For each item read, the Read-Test checks wether the item read has timestamp $t <$ *count*. If this does not hold, R is aborted.

The SGT Method. The SGT method proposed in [20] provides an implementation of the C3-I test. In particular, the server maintains a serialization graph SG with all transactions committed at the server. The server broadcasts periodicaly the serialization graph to the clients. Each clients maintains a local copy of the graph. The Read-Test checks for cycles in the local copy of the graph.

The F-matrix method. The F-matrix proposed in [24] provides an interesting implementation of the C2-I test. Along with each item x, an array C with n elements (where n is the number of items in the database) is broadcast. C[i] provides information regarding the transaction that affected the values of both items x and i.

Multiple Versions. With the multiple version method proposed in [19], instead of broadcasting only the most current value of each item, the values that the item had at the previous k currency intervals are also broadcast. The method provides C4 consistency and strict currency with respect to various points in the lifetime of the transaction.

8 Conclusions and Future Work

In this paper, we proposed a general theory for currency and consistency for an extended client/server environment in which the server broadcasts items of interest to a large number of clients. An interesting feature of our currency and consistency model that is applicable to non-broadcast systems as well is the decoupling of currency and consistency. Currency can be enforced in non-transactional settings. We relate currency and consistency through the definition of strict currency which is the type of consistency we get in transactional systems. Our model provides the necessary tools for arguing about the correctness and other properties of the various protocols. In addition, it provides the basis for new protocols to be advanced. The proposed model can be easily extened for the case of a cache being maintained at the clients. In this case, clients read items from the broadcast channel or from the cache. The theory is directly applicable if the values in cache are maintained current. It can also be extended for deferred cache update policies. An interesting line for future research is applying the theory of readset currency and consistency in the case of warehouses and caches, since both warehouses and caches are subsets of database states.

References

[1] S. Acharya, R. Alonso, M. J. Franklin, and S. Zdonik. Broadcast Disks: Data Management for Asymmetric Communications Environments. In *Proceedings of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD 95)*, June 1995.

[2] S. Acharya, M. J. Franklin, and S. Zdonik. Disseminating Updates on Broadcast Disks. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB 96)*, September 1996.

[3] A. H. Ammar and J. W. Wong. The Design of Teletext Broadcast Cycles. *Performance Evaluation*, 5(4), 1985.

[4] S. Banerjee and V. O. K. Li. Evaluating the Distributed Datacycle Scheme for a High Performance Distributed System. *Journal of Computing and Information*, 1(1), 1994.

[5] D. Barbará. Certification Reports: Supporting Transactions in Wireless Systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 1997.

[6] D. Barbará and T. Imielinski. Sleepers and Workaholics: Caching Strategies in Mobile Environments. In *Proceedings of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD 94)*, pages 1–12, 1994.

[7] S. K. Baruah and A. Bestavros. Real-Time Mutable Broadcast Disks. In *Proceedings of the RTDB*, pages 3–22, 1997.

[8] P. A. Bernstein, V. Hadjilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addisson-Wesley, 1987.

[9] A. Bestavros and C. Cunha. Server-initiated Document Dissemination for the WWW. *IEEE Data Engineering Bulletin*, 19(3), September 1996.

[10] P. M. Bober and M. J. Carey. Multiversion Query Locking. In *Proceedings of the 1992 SIGMOD Conference*, pages 497–510, 1992.

[11] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, and A. Weinrib. The Datacycle Architecture. *CACM*, 35(12), Dec 1992.

[12] A. Chan and R. Gray. Implementing Distributed Read-Only Transactions. *IEEE Transactions on Software Engineering*, 11(2):205–212, 1985.

[13] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users. *ACM TODS*, 24(1), 1999.

[14] M. J. Franklin and S. B. Zdonik. A Framework for Scalbale Dissemination-Based Systems. In *Proceedings of the OOPSLA Conference*, pages 94–105, 1997.

[15] H. Garcia-Molina and G. Wiederhold. Read-Only Transactions in a Distributed Database. *ACM TODS*, 7(2):209–234, 1982.

[16] T. Imielinski, S. Viswanathan, and B. R. Badrinanth. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, May/June 1997.

[17] J. Jing, A. K. Elmargarmid, S. Helal, and R. Alonso. Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments. *ACM/Baltzer Mobile Networks and Applications*, 2(2), 1997.

[18] V. C. S. Lee, S. H. Son, and K. Lam. On the Performance of Transaction Processing in Broadcast Environments. In *Proceedings of the International Conference on Mobile Data Access (MDA'99)*, December 1999. to appear.

[19] E. Pitoura and P. K. Chrysanthis. Exploiting Versions for Handling Updates in Broadcast Disks. In *Proceedings of 25th VLDB*, pages 114–125, 1999.

[20] E. Pitoura and P. K. Chrysanthis. Scalable Processing of Read-Only Transactions in Broadcast Push. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 432–441, 1999.

[21] E. Pitoura, P. K. Chrysanthis, and K. Ramamritham. A Theory of Currency and Consistency in Broadcast Databases (extended version). Technical Report TR: 99-26, Univ. of Ioannina, Computer Science Dept, 1999.

[22] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.

[23] R. Rastogi, S. Mehrotra, Y. Breitbart, H. F. Korth, and A. Silberschatz. On Correctness of Non-serializable Executions. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 97–108, 1993.

[24] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham. Efficient Concurrency Control for Broadcast Environments. In *ACM SIGMOD International Conference on Management of Data*, pages 85–96, 1999.

[25] W. E. Weihl. Distributed Version Management for Read-Only Actions. *ACM Transactions on Software Engineering*, 13(1):56–64, 1987.

[26] J. Wong. Broadcast Delivery. *Proceedings of the IEEE*, 76(12), December 1988.

[27] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, and K. Ramamritham. Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments. In *Proceedings of 3rd IEEE Real-Time Technology and Applications*, 1997.

[28] T. Yan and H. Garcia-Molina. SIFT – A Tool for Wide-area Information Dissemination. In *Proceedings of the 1995 USENIX Conference*, 1995.

Appendix

Proof of Theorem 1

(The Invalidation Theorem) *Let $(x, u) \in RS(R)$, x be read at time t_x and $t_{xd} = t_x$ - d. Let IR_m be the report that was broadcast at time m and IR be the set of all such reports that were broadcast during the lifetime of R, $IR = \{IR_m : m \in lifetime(R)\}$. R is strictly current with respect to time t_k, iff:*

(a) if $t_{xd} \leq t_k$ then $x \notin IR_m$, $t_{xd} < m \leq t_k$, and

(b) if $t_{xd} > t_k$ then $x \notin IR_m$, $t_k < m < t_{xd}$

Proof.

Let $(x, u) \in RS(R)$, x is read at time t_x.
Case (a): $t_{xd} \leq t_k$
We must show that this value was produced by the last transaction T committed by t_k that wrote x. Since R read x prior to t_k, transaction T committed prior to t_k. We must show that x was not updated between the time it was read (t_{xd}) and t_k. For the purposes of contradiction, assume that x was updated. Then, x should have been appeared in an IR_m that was broadcast after t_{xd} and prior to t_k, that is $t_{xd} < m \leq t_k$, which violates condition (a) above.
Similarly we can prove Case (b): $t_{xd} > t_k$. □

Proof of Theorem 2

(The Read-Test Theorem) *A consistency criteria may be violated only when a a read-only transaction reads a new data item if and only if the schedule of server transactions is strict.*

Proof.

(1) If the schedule of server transactions is not strict then the testing for consistency when an item is read is not sufficient.

Consider the following schedule that includes four server transactions T_0, T_1, T_2, T_3 and one read-only transaction T_R:
$w_0(y)c_0w_3(z)r_2(z) \mid\mid w_2(a)r_1(a)w_1(x)c_1c_2\ r_R(x)r_R(y) \mid\mid w_3(y)c_3$

The schedule of server transactions is not strict, since the serialization order is $T_0 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$, while the commit order is $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3$. Note that consistency is violated when T_3 writes y. R can not decide whether to commit, since T_3 could write an item and violate consistency at any time. Note that there are only dependency edges among server transactions. □

(2) If the schedule of update transactions is strict, then testing for consistency when a transaction reads a new item is sufficient.

Let SG be the serialization graph that includes both server and client transactions and SSG be the subgraph of SG that includes only server transactions.

We assume that there is no cycle in the SG, thus the only cycle possible includes at least one read-only transaction R and is of the form: $T \rightarrow R \rightarrow T' \rightarrow \ldots\ rightarrow\ T$, transaction T' overwrote an item y previously read by R, and R read an item x from transaction T.

Assume that SG is acyclic at t and that at some point $t' > t$ an operation causes an edge to be added in the SG. Assume, for the purposes of contradictions that this operation is not a read operation of a client transaction.

Case (a) The edge is between two server transactions T_1 and T_2 and was caused by an operation of T_1. Assume for the purpose of contradiction, that a cycle is formed: $T \rightarrow R \rightarrow T' \rightarrow \ldots \rightarrow T_1 \rightarrow T_2 \ldots \rightarrow T$. T_1 is committed after T, since T is a transaction that has already committed (since R read a value produced by T), However T_1 precedes T in the serialization order which violates the assumption that the schedule of server transactions is strict.

Case (b) The edge is between a server transaction T_1 and a client transaction R', this means that T_1 overwrote an item read by R'. Again T_1 precedes T which violates the condition that schedules are strict.
Thus, the only type of edge that can create a cycle is when R reads an item. □

Proof of Lemma 1

Lemma 1: *The C4-I test is sufficient to get all versions of C4.*

Proof.

We will show that the path for R does not include any other read-only client transactions. For the purposes of contradiction let us assume that there is such a path that includes another read-only transaction say R'. Then $T' \rightarrow \ldots T_m \rightarrow R' \rightarrow T_k \ldots \rightarrow T$.
From the Read-Test for R',
$commit\_timestamp(T_m) < commit\_timestamp(T_k)$, (1)
while from the Read-Test for R,
$commit\_timestamp(T') < commit\_timestamp(T)$ (2).
From schedules being strict,
$commit\_timestamp(T') > commit\_timestamp(T_k)$, (3)
and
$commit\_timestamp(T_m) > commit\_timestamp(T')$(4). Then
$commit\_timestamp(T') > commit\_timestamp(T_k)$ (from 3)
$\Rightarrow commit\_timestamp(T') > commit\_timestamp(T_m)$ (from 1)
$\Rightarrow commit\_timestamp(T') > commit\_timestamp(T)$ (from 4)
which violates the Read-Test for R (inequality 2). □

Proof of Theorem 3

(Global Consistency Theorem) *The only enforcible criteria that includes read-only at more than one clients, if we assume no communication between clients or from the server to the client, is C4.*

Proof (sketch)

Lemma 1 shows as that is possible to enforce C4 by enforcing the C4-I test locally at each client.

We need to show that the other criteria are not enforcible. The idea is that at different clients, non-conflicting server transactions may be ordered in a conflicting way. □.