Characterizing the Temporal and Semantic Coherency of Broadcast-based Data Dissemination

Evaggelia Pitoura[1*], Panos K. Chrysanthis[2**], and Krithi Ramamritham[3]

[1] Department of Computer Science, University of Ioannina, Greece
pitoura@cs.uoi.gr
[2] Department of Computer Science, University of Pittsburgh, USA
panos@cs.pitt.edu
[3] Department of Computer Science and Engineering, IIT, Bombay, India
krithi@cse.iitb.ac.in

Abstract. In this paper, we develop a general theory of temporal and semantic coherency for an extended client/server architecture in which the server broadcasts items of interest to a large number of clients without a specific client request. Such architectures 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 applications, for example in publish-subscribe systems. We introduce various forms of temporal and semantic coherency applicable to such architectures and present a framework to precisely define protocols for enforcing them.

1 Introduction

While traditionally data is delivered from servers to clients on demand, a wide range of emerging data-based applications can benefit from a broadcast mode for data dissemination. In such applications, the server repetitively broadcasts data to a client population without explicit requests. 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.

For example, in electronic commerce applications, such as auctions, it is expected that a typical auction might bring together millions of interested parties even though only a small fraction may actually offer bids. Updates based on the bids made must be disseminated promptly and consistently. Fortunately, the relatively small size of the database, i.e., the current state of the auction, makes broadcasting feasible. But, the communication bandwidth available for a client to communicate with servers is likely to be quite restricted. Thus, an attractive approach is to use the broadcast medium to transmit the current

* Work supported in part by IST-2001-32645
** Work supported in part by NSF award ANI-0123705

state of the auction while allowing the clients to communicate their updates (to the current state of the auction) using low bandwidth uplinks with the servers. Broadcast-based data dissemination is also likely to be a major mode of information transfer in mobile computing and wireless environments [9, 1, 6]. Many such systems have been proposed [17, 11] and commercial systems such as Vitria [19] already support broadcasting. Other applications of broadcasting, include stock trading, next generation road traffic management systems and automated industrial plants [21].

Motivation. The problem addressed in this paper is motivated by such applications. In particular, we are concerned with providing readers with *consistent* (semantically coherent) and *current* (temporally coherent) data. By semantic coherency we mean the consistency properties of the data, e.g., *did the data items read by a client transaction result from a serializable execution of the update transactions at the server?* By temporal coherency we mean currency related properties, e.g., *when were the data items read by a client transaction current at the server?* Without both semantic and temporal coherency, users may be making decisions based on data which even if consistent, may be outdated. Given the limited amount of bandwidth available for clients to communicate with the broadcast server in such environments, achieving semantic and temporal coherency efficiently is a challenging research issue.

Several protocols have been proposed with the goal of achieving consistency and currency in broadcast environments. In [1], the authors discuss the tradeoffs between currency of data and performance issues when some of the broadcast data items are updated by processes running on the server. However, the updates do not have transactional semantics associated with them either at the server or at the clients. Herman et. al. [8] discuss transactional support in the Datacycle architecture, which is also an asymmetric bandwidth environment. Realizing that serializability as the correctness criterion may be expensive, and perhaps unnecessary in such environments, various protocols [16, 2, 13, 12, 10] attempt to cater to less demanding correctness requirements. However, the exact semantic and temporal coherency properties associated with them is not always clear. This lack of clarity has motivated the work reported in this paper. So, instead of proposing specific protocols for broadcast databases, as has heretofore been the case, we develop a unified framework for correctness in broadcast-based data dissemination environments.

Overall Goals and Contributions. Our framework allows us to characterize different semantic and temporal coherency properties of client transactions in broadcast-based data dissemination environments:

- We introduce the *currency interval* of an item in the readset of a transaction as the time interval during which the value of the item is valid. Based on the currency intervals of the items in the readset of a client transaction, we develop the notion of temporal spread and lag of the readset and two notions of currency (overlapping and oldest-value) through which we characterize the temporal coherency of the readset of a transaction.

– We identify five notions of semantic coherency of transactions' readsets. These are all variations or weakening of the standard serializability criterion.

Along the way we show what type of temporal and semantic coherency properties are guaranteed by the various protocols introduced in the literature and indicate how they can be extended to provide different properties.

Paper Organization. In Section 2, we introduce the broadcast model, give background definitions, and state the assumptions underlying our framework. In Section 3, after presenting our temporal coherency model, we present currency control protocols that satisfy different currency properties. In Section 4, we first define various models of semantic coherency that provide clients with transaction consistent database states and then present consistency control protocols based on the Read-Test theorem. We also present various propositions that relate certain types of temporal and semantic coherency. Finally, in Section 5, we present our conclusions.

2 The Model

We consider an extended client/server architecture in which a server broadcasts data items from a database to a large client population without a specific request from any client. Each client listens to the broadcast and fetches data items as they arrive. This way data can be accessed concurrently by any number of clients without the performance degradation that would result if the server were to transmit to individual clients. However, each client's access to the data is strictly sequential, since clients need to wait for the data of interest to appear on the channel. The broadcast may be periodic or aperiodic. Without loss of generality, we assume periodic broadcast.

2.1 Model Assumption

Data dissemination protocols take into account the asymmetry that exists between the communication capacity from the server to the clients, the typically meager communication capacity of the backchannel from the client to the server, and the scalability issues arising at the servers. The asymmetry is the result of the huge disparity in 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. For scalability reasons, it is important to limit the number of explicit client requests seen by the server as well as individualized processing of each client request at the server. These reasons along with the need to decrease the latency of client transactions justify the use of client-side protocols for achieving client-specific semantic and temporal coherency requirements. These considerations motivate the following model assumptions:

1. The server is stateless: it does not maintain any client-specific information.

2. All updates are performed at the server and up-to-date data is disseminated from the server.
3. To get semantic and temporal coherency related information, clients do not contact the server directly instead such information is broadcast along with the data.

2.2 Update Period

Assume that a read operation on an item x is initiated at time instance t_r. The client must wait for x to appear on the channel, say, at time instance t_x. When we talk about the time of a read operation we refer to t_x (i.e., the time instance the item is actually read) rather than to t_r. The value of x that the client reads is the value placed on the channel at $t_x - d$, where d is the communication delay.

Which value of an item x is broadcast at a time instance depends on the update broadcast protocol. We consider a periodic update protocol with an *update cycle period* or *update frequency* p_u. With this protocol, the data values at the broadcast are updated every p_u time units. Items broadcast during the same update period are said to belong to the same update cycle. We use the notation $begin\_cycle(t)$ for a time instance t to denote the beginning of the update cycle period that includes t. The value of an item that the server puts on the broadcast at time t_x is the value of x at the server database at time instance $begin\_cycle(t_x)$ which may not be its current value at the database, if x was updated between $begin\_cycle(t_x)$ and t_x. An update cycle period equal to 0, means that for each item the server broadcasts the most up-to-date value. In this case, $begin\_cycle(t) = t$.

Thus, the value that the client reads at t_x from the broadcast is the value at the server database at $begin\_cycle(t_x - d)$. For simplicity, we will assume in the rest of this paper that the communication delay is negligible (i.e., $d = 0$).

In the following, we shall use the term cycle, to denote the update cycle. Usually, the update cycle period is considered equal to the period of the broadcast.

2.3 Preliminary Definitions

Next, we formally define the broadcast content and the transaction readset as subsets of a database state. A database state is typically defined as a mapping of every data item of the database to a value in its domain.

Definition 1. (Database State) *A databases state, denoted DS, is a set of (data item, value) pairs. The database state at time instance t is denoted as $DS(t)$.*

Let BS_c be the set of (data item, value) pairs that appear on the broadcast during the cycle that starts at time instance c. BS_c is a subset of a database state, in particular $BS_c \subseteq DS(c)$.

We use R to denote a client read-only transaction. Let t_{begin_R} and t_{commit_R} be the time R performs its first read operation and the time it commits, respectively.

Definition 2. (Lifetime) *The* lifetime *of R, denoted* lifetime(R) *is the time interval* $[t_{begin_R}, t_{commit_R}]$.

Definition 3. (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 includes at most one (data item, value) pair per data item. If a transaction reads the same item more than once, the readset includes the value read last. A readset is a subset of database state.*

In general, a transaction R may read items from different cycles. In particular: $RS(R) \subseteq \cup_{c \in [t_1, t_2]} BS_c$, where $t_1 = begin\_cycle(t_{begin_R})$ and $t_2 = begin\_cycle(t_{commit_R})$. This says that the readset of R is a subset of all the data items that are broadcast during the cycles that overlap with the execution of R.

3 Temporal Coherency

3.1 A Temporal Coherency Framework

We first define the currency of a single item read by a client.

Definition 4. (Currency Interval of an Item) *CI(x, R), the currency interval of x in the readset of R is $[c_b, c_e)$ where c_b is the time instance the value of x read by R was stored in the database and c_e is the time instance of the next change of this value in the database. If the value read by R has not been changed subsequently, c_e is infinity.*

The above time instances (c_b and c_e) correspond to *transaction* time [18]: the time that the value is stored in the database. They could as well correspond to *valid* time, i.e., the time when the value becomes effective in reality, or to some *user-defined* notion of time. The rest of this section holds for all such interpretations of time.

Based on the CIs of the items in the readset we define properties that characterize the currency of the readset. First, based on whether or not there is a time instance when the values read by R are concurrently current at the server, we define two different forms of currency of the readset: *overlapping currency* and *oldest-value currency.*

Definition 5. (Overlapping Currency of a Transaction) *A transaction R is overlapping current if and only if there is an interval of time that is included in the currency interval of all items in R's readset: $\cap_{(x,u) \in RS(R)} CI(x, R) \neq \emptyset$. Let this overlap correspond to the interval $[c_b, c_e)$, then $Overlapping\_Currency(R) = c_e^-$ (where t^- refers to the time instance just before t) if c_e is not infinity, else $Overlapping\_Currency(R) = current\_time$.*

If a transaction R is overlapping current and the intersection is the interval $[c_b, c_e)$, then its readset RS(R) is a subset of an actual database state $DS(t)$ at the server ($RS(R) \subseteq DS(t)$) for all $t \in [c_b, c_e)$.

If a transaction is not overlapping current, we characterize its currency based on the oldest value read by R which is an indication of the outdatedness of the data.

Definition 6. (Oldest Value Currency of a Transaction) *The oldest-value currency of a transaction R, denoted $OV\_Currency(R)$, is equal to c_e^-, where c_e is the smallest among the endpoints of the $CI(x, R)$, for every x, $(x, u) \in RS(R)$.*

For overlapping current transactions, oldest-value currency reduces to overlapping currency, that is, if R is overlapping current then $OV\_currency(R) = Overlapping\_Currency(R)$.

In general, if a transaction is not overlapping current, its readset may not be a subset of a single database state DS at the server. In this case, we are interested in minimizing the discrepancies in the currency of the items in the readset. This is captured through the notion of temporal spread.

Definition 7. (Temporal Spread of a Readset) *Let min_{c_e} be the smallest among the endpoints and max_{c_b} the largest among the begin-points of the currency intervals of items in the readset of a transaction R. The temporal spread of the readset of R, $temporal\_spread(R)$, is equal to (max_{c_b} - min_{c_e}), if $max_{c_b} > min_{c_e}$ and zero otherwise.*

Temporal spread measures the discrepancies among the values read by a transaction; the larger the spread, the more distant in time the different server states seen by the transaction. For overlapping current transactions, the temporal spread is zero.

Example 1. (Fig. 1) Let data items x_1, x_2, x_3 and x_4 be in the readset of a transaction.

(i) Let R_1 be a transaction with $CI(x_1, R_1) = [2, \infty)$, $CI(x_2, R_1) = [4, 8)$, $CI(x_3, R_1) = [5, 10)$ and $CI(x_4, R_1) = [2, 18)$. Then, R_1 is overlapping current $Overlapping\_Currency(R_1)$=$8^-$ (the temporal spread is 0).

(ii) Let R_2 be another transaction with $CI(x_1, R_2) = [2, \infty)$, $CI(x_2, R_2) = [4, 8)$, $CI(x_3, R_2) = [5, 10)$ and $CI(x_4, R_2) = [9, 13)$. In this case, R_2 is not overlapping current, but is oldest-value current with $OV\_Currency(R_2) = 8^-$, meaning that the oldest value read by R_2 corresponds to 8^-; all other values remain valid even after 8. Temporal_spread$(R_2) = 9 - 8 = 1$, that is the discrepancy between the database states seen by R_2 is 1.

(iii) Finally, let R_3 be a transaction R_3 with $CI(x_1, R_3) = [2, \infty)$, $CI(x_2, R_3) = [4, 10)$, $CI(x_3, R_3) = [5, 10)$ and $CI(x_4, R_3) = [15, 18)$. $OV\_Currency(R_3) = 9$ (i.e., R_3 reads less out-dated than R_2), however R_3's temporal spread is equal to $15 - 10 = 5$, which is larger than the temporal spread of R_2.

We now examine how the currency of the readset relates to the transaction's lifetime. To this end, we define *transaction-relative currency* which relates the currency of the readset of a transaction R with some time instance during its lifetime.

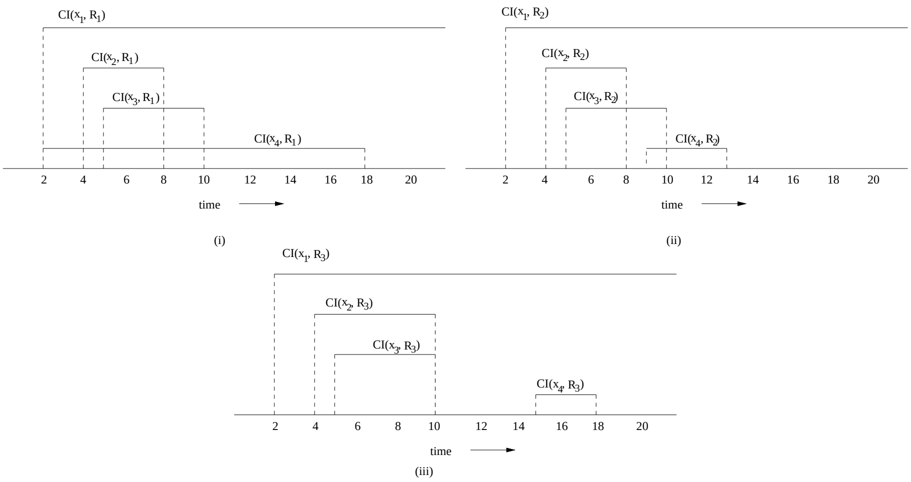

Fig. 1. Example 1

Definition 8. (Transaction-Relative Currency) *A transaction R is* relative overlapping current *with respect to some time t, if $t \in CI(x, R)$ for all x read by R. A transaction R is* relative oldest-value current *with respect to some time t, if $t \leq OV\_Currency(R)$.*

For a given R, three possibilities of t are: $t \geq t_{commit_R}$, $t_{begin_R} \leq t < t_{commit_R}$, and $t < t_{begin_R}$. Correspondingly, we may have t to refer to the beginning of the respective cycle, that is, $t \geq begin\_cycle(t_{commit_R})$, $begin\_cycle(t_{begin_R}) \leq t < begin\_cycle(t_{commit_R})$, and $t < begin\_cycle(t_{begin_R})$.

Definition 9. (Temporal Lag) *Let t_c be the largest $t \leq t_{commit_R}$, with respect to which R is relative (overlapping or oldest-value) current, then $temporal\_lag(R)$ $= t_{commit_R} - t_c$.*

Temporal lag indicates the currency of the values read; temporal lag of zero, means that the transaction reads items current at its commit time. The smaller the temporal lag and the temporal spread, the higher the *temporal coherency* of a read transaction. Thus, R exhibits the best temporal coherency when it is overlapping relative current with respect to t_{commit_R} (then both the time lag and the temporal spread are zero).

Example 2. (Fig. 2) If the lifetime of transaction R_1 in Example 1 is: (i) $[3, 7]$, then R_1 has temporal lag equal to 0, (ii) $[4, 12]$, then R_1 has temporal lag equal to 4 $(12 - 8)$, (iii) $[10, 19]$, then R_1 has temporal lag equal to 11 $(19 - 8)$. That is, although in all cases R_1 is overlapping current (i.e., the temporal spread is zero), R_1 in case (i) reads more current data than in case (ii) and this in turn is more current than in case (iii) (with respect to its lifetime).

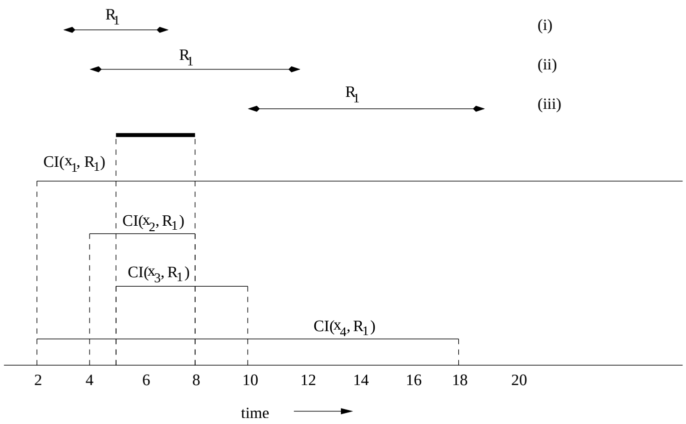

Fig. 2. Example 2

3.2 Achieving Temporal Coherency

Let us first see what kind of currency is attained when no additional control information is broadcast. The following proposition shows that we cannot guarantee anything better than oldest-value currency, in particular (proof in [14]):

Proposition 1. *Let $t_{lastread_R}$ be the time instance R performs its last read. If R reads items from the broadcast channel (without any additional information), then R is relative oldest-value current with respect to $begin\_cycle(t_{begin_R})$, with $temporal\_lag(R) \leq t_{commit_R} - begin\_cycle(t_{begin_R})$ and $temporal\_spread(R) \leq t_{lastread_R}$ - $begin\_cycle(t_{begin_R})$.*

From Proposition 1, we see that: the larger the period of the cycle, the better (smaller) the temporal spread and the worse (larger) the temporal lag. In fact, the best lag is attained when the period of the cycle is zero. In this case, $temporal\_lag(R) \leq t_{commit_R} - t_{begin_R}$ and $temporal\_spread(R) \leq t_{lastread_R}$ - t_{begin_R}.

If a transaction R reads all items from the same cycle, then $temporal\_spread(R)$ = 0, R is overlapping current with respect to $begin\_cycle(t_{commit_R})$ and its $temporal\_lag(R)$ is equal to the update cycle period.

If we want to improve temporal coherency, additional information needs to be made available to clients. Next, we study some protocols that have appeared in the literature and formally express and prove their properties. When we say that a protocol satisfies a property, we mean that all transactions that follow the protocol have the property. The protocols fall in two broad categories: (a) invalidation (which corresponds to broadcasting the endpoints (c_es) of the currency interval for each item) and (b) versioning (which corresponds to broadcasting the begin points (c_bs) of the currency interval for each item).

Invalidation-based Protocols. Invalidation is based on the following theorem (proof in [14]).

Theorem 1. (Updates and Currency) *Let t_k be a time instance. Let $(x, u) \in RS(R)$ and x be read at time instance t_x.*

Condition (1) If $t_x \leq t_k$, then x is not updated in ($begin\_cycle(t_x)$, t_k].
Condition (2a) If $t_x > t_k$ and $begin\_cycle(t_k) = begin\_cycle(t_x)$, then x is not updated in ($begin\_cycle(t_x)$, t_k].
Condition (2b) If $t_x > t_k$ and $begin\_cycle(t_x) > begin\_cycle(t_k)$, then x is not updated in $(t_k$, $begin\_cycle(t_x)]$.

(A) Conditions (1) and (2a) hold for all x in $RS(R)$ if and only if R is relative oldest-value current with respect to t_k.
(B) Conditions (1), (2a) and (2b) hold for all x in $RS(R)$ if and only if R is relative overlapping current with respect to t_k.

An invalidation report is a list with the items that have been updated at the server since the broadcast of the previous invalidation report. Without loss of generality, we assume that invalidation reports are broadcast periodically. This also facilitates selective tuning; a client can estimate the broadcast of the next invalidation report and tune in at the appropriate time. Let p_i be the *invalidation period*; invalidation period equal to 0 means that an invalidation report for an item is broadcast immediately after the item is updated.

For notational convenience, we assume that broadcasting invalidation reports takes negligible time. Let IR_c be the invalidation report that is sent at time instance c, IR_c includes all items that have been updated since the broadcast of the invalidation report IR_{c-p_i}.

For a time instance t, we use the notation $invalidation(t)$ to denote the time instance when the invalidation report covering t is broadcast, that is the time instance $t' \geq t$ when the next invalidation report is broadcast. That is, for $c - p_i < t \leq c$, $invalidation(t) = c$. For $p_i = 0$, $invalidation(t) = t$. In the case in which the period of broadcasting invalidation reports (p_i) is equal to the update period (p_u), invalidation reports are broadcast at the beginning of each cycle, and $IR_{begin\_cycle(t)}$ includes all items that have been updated since the beginning of the previous cycle, and covers all t, $begin\_cycle(t) - p_u < t \leq begin\_cycle(t)$. In particular, $begin\_cycle(t)$ is covered by $IR_{begin\_cycle(t)}$ (that is, $invalidation(begin\_cycle(t)) = begin\_cycle(t)$).

The following property relates invalidation reports and updates (proof in [14]).

Property 1 (Invalidation). For a data item x, let two time instances t_1 and t_2, $t_2 > t_1$, x is not updated in the time interval $[t_1, invalidation(t_2)]$ iff $x \notin IR_c$, for every c, $invalidation(t_1) \leq c \leq invalidation(t_2)$.

Using Property 1 and Theorem 1, we can design various invalidation protocols and prove their currency properties. In particular, to guarantee that a transaction R is oldest-value (overlapping) current with respect to some time instance, we need to test for each item in its readset whether it is updated in the specific interval defined in Theorem 1. To do so, we read the corresponding invalidation reports specified by Property 1. In the case that the item appears in any such report, the transaction is aborted.

Note that, using Property 1, guarantees oldest-value (overlapping) currency with respect to $invalidation(t_k)$ as opposed to t_k. In other words, the finest granularity that we can get through invalidation reports is at the time the invalidation report is sent; in other words, we can only see the database states $DS(t)$ that correspond to invalidation points. In the special case in which $p_i = p_u$ (that is when invalidation reports are broadcast at the beginning of each cycle), we get currency with respect to *begin_points*.

If invalidation reports are broadcast more often (small p_i), then we need to read more reports. However, we need to listen to the broadcast for less time, since the last report to be read is the report that covers t_2 and how long after t_2 this report will be broadcast depends on p_i.

A specific instance of the invalidation protocol with $t_k = begin\_cycle(t_{begin_R})$ appears in the literature as invalidation lists [13, 12] or certification reports [2] (if we consider only read-only transactions).

A variation of invalidation report is based on re-reading an item that appears in an invalidation report, instead of aborting the issuing transaction. Such a method called *autoprefetch* was proposed in [1]. There, it was used to achieve overlapping currency with respect to $begin\_cycle(t_{commit_R})$.

Versioning. Another basic approach for ensuring overlapping currency is based on versions or timestamps [12]. In particular, with each (data item, value) pair in the broadcast, $(x, u) \in BC_c$, we also broadcast a version number or timestamp, $timestamp(x) = c_b$, where c_b is the time instance the value of x was written in the database. Let $t_0 = begin\_cycle(t_{begin_R})$. When R reads x, if $timestamp(x) > t_0$, R is aborted. It is easy to show (proof in [14]) that

Claim. The versioning protocol ensures that a transaction R is overlapping current with respect to t_0, $t_0 = begin\_cycle(t_{begin_R})$.

The following protocol [12] uses both invalidation reports and versioning. Let IR_{t_i} be the first invalidation report that includes an item previously read by R. Until t_i, R reads items as they appear in the broadcast. After t_i, R checks whether $timestamp(x) > t_i$, if so, R is aborted. It is easy to show (proof in [14]) that

Claim. The versioning protocol with invalidation reports ensures that a transaction R is overlapping current with respect to t_i, where IR_{t_i} is the first invalidation report that includes an item previously read by R.

4 Semantic Coherency

4.1 A Semantic Coherency Framework

The currency properties of a client transaction focus on the timeliness of its readset. In this section, we consider whether the data read by a transaction are semantically correct. We relate semantic coherency with database consistency.

Definition 10. (Database State and its Consistency) *A* database state is consistent *if it does not violate the integrity constraints [3] defined on the database. A* subset of a database state is consistent *if it is a subset of a consistent database state [15].*

Thus, since a readset of a transaction is a subset of a database state, it is consistent if it is a subset of a consistent database state.

A consistent database state need not necessarily be produced by a set of server transactions. Hence, various consistency guarantees stronger than correspondence to a consistent database state have been defined, based on transaction serializability [4, 7, 20]. All (except one) guarantee that a read-only client transaction sees a consistent database state. We discuss them now in increasing order of "strength":

Definition 11. (Degrees of Consistency) *Let R be a read-only transaction, then*

- *R is* **C0** *consistent if no consistency requirements are placed on its readset.* No consistency[4] *[7] and* opportunistic *consistency [1] are examples of occurrence of C0 in the literature.*
- *R is* **C1** *consistent if $RS(R)$ is a subset of a consistent database state. No other serializability-based requirements are placed on this state.* Consistency *[20] and* weak consistency *[7] are examples of occurrence of C1 in the literature.*
- *R is* **C2** *consistent if R is serializable with the set of server transactions that produced values that are seen (either directly or indirectly) by R.* Update consistency *[4, 16],* weak consistency *[5] and* external consistency *[20] are examples of C2 from the literature.*
- *R is* **C3** *consistent if R is serializable with the set of* all *server transactions.* Weak consistency *[4] is an example of C3 from the literature.*
- *R is* **C4** *consistent if R is serializable with the set of all server transactions and observes a serial order of server transactions that agrees with the order in which the server transactions committed.*

A schedule is *rigorous* iff the commit order of transactions is compatible with the serialization order. So, C4 is C3 plus the requirement that schedules be rigorous.

Each criterion in the above list strengthens the previous criterion. Let us examine this list working backwards: C4 demands the serializability of all server transactions and R. In addition, it demands that the serialization order be consistent with the commit order. C3 is derived by dropping from C4 the requirement dealing with the serialization order having to agree with the commit order. So,

[4] We have retained the original terms, even though some terms like "strong" and "weak" have been used by different authors to mean different things. By formulating a framework for discussing correctness requirements, our hope is to shed some light on their precise meanings.

C3 simply demands the serializability of all server transactions and R. C2 is derived from C3 by demanding the serializability of a read-only transaction R with just those server transactions from which R reads data directly or indirectly. C1 drops the requirement of serializability from C2, being satisfied with the readset simply being consistent. C0 drops even the consistency requirement from C1.

4.2 Relation to Temporal Coherency

In the following, we adapt the definition of the currency interval to reflect that the server broadcast items from a transactional database system. The temporal coherency framework holds for this adapted definition as well.

Definition 12. ((Transactional) Currency Interval of an Item) *CI(x, R), the currency interval of x in the readset of R is $[c_b, c_e)$ where c_b is the commit time of the transaction that wrote the value of x read by R and c_e is the commit time of the transaction that next changes this value. If the value read by R has not been changed subsequently, c_e is infinity.*

Our only assumption about concurrency control at the server is that server schedules are serializable and that only committed values are broadcast. It can be proved ([14]) that:

Proposition 2. *If a client transaction R is overlapping current, then R is C1 consistent.*

Overlapping current transactions using the transactional definition of the currency interval correspond to the c_e*-vintage* transactions of [7]. A transaction satisfies the t-vintage requirement iff it reflects the updates of all transactions committed before time instance t and does not reflect any updates due to any transaction committed after t. Similarly, a concept related to oldest-value current transactions are t*-bound* transactions [7]. A t-bound transaction sees the results of all transactions committed prior to time instance t but also of some transactions committed after t. Thus, we may say that a transaction R with $OV\_Currency(R) = t_o$, is t_o-bound.

4.3 Achieving Semantic Coherency

Proposition 2 shows that to attain C1 consistency, it suffices to attain overlapping currency. However overlapping currency is not sufficient to attain stricter notions of consistency even when a transaction reads all items from the same cycle (proof in [14]):

Proposition 3. *If a client transaction reads all items from a single cycle, it is C1, but not necessarily C2.*

The Read-Test. As the following theorem shows, in order to efficiently check for serializability-based consistency (that is, C2, C3, or C4 consistency), the server schedules should be rigorous. The theorem states (proof in [14]) that when server schedules are rigorous, checks of consistency violations can be done when a new item is read by a client transaction. This simplifies the construction of protocols for attaining semantic coherency.

Theorem 2. (Read-Test Theorem) *It suffices to check for C2, C3, or C4 consistency violations when a read-only transaction reads a new data item if and only if the schedule of server transactions is rigorous.*

The proof of the Read-Test theorem also shows that if the server schedule is not rigorous, then any of the C4, C3, and C2 consistency may be violated anytime a server transaction T_f writes an item that was previously read by a client transaction R and there is some path from T_f to a server transaction T from which R read an item. This can happen even after R has completed its operations.

Note that even when the server schedules are rigorous, C3 is not equivalent to C4. For example, let T_0, T_1, T_2 be server transactions and T_R be a client transaction. Consider the schedule $w_0(y)\ c_0\ r_R(y) \ ||\ w_1(y)w_2(x)c_1c_2 \ ||\ r_R(x)c_R$, where $||$ denotes the beginning of a new cycle. The commit order (which is compatible with the serializability) order at the server is: $T_0 \rightarrow T_1 \rightarrow T_2$. The serializability order at the client it is: $T_0 \rightarrow T_2 \rightarrow T_R \rightarrow T_1$, which means that T_R is C3 but not C4.

From Theorem 2, we get that:

Corollary 1. *If the server schedule is rigorous and R reads all items from the same cycle, then R is C4.*

Thus, there is a trade-off between temporal and semantic coherency. The larger the length of the cycle (i.e., the update frequency) the worst the temporal spread, but the stronger the consistency attained.

Semantic Coherency Protocols. Let us consider what is needed to achieve C2, C3 and C4 consistency, in the case of rigorous schedules (proofs in [14]).

Corollary 2. *Let the server schedule be rigorous, R read x from T, and $Follow_R$ $= \{T_f$: T_f overwrote an item previously read by $R\}$. Then,*

> *<u>C4 Read Test</u> Let $t_{min} = min_{T_f \in Follow\_R}\ (t_{commit_{T_f}})$ and $t_T = t_{commit_T}$, R is C4 iff $t_T < t_{min}$.*
> *<u>C3 Read Test</u> R is C3 iff there is no path from any $T_f \in Follow_R$ to T in the serialization graph of server transactions.*
> *<u>C2 Read Test</u> R is C2 iff there is no path from any T_f to T that includes only dependency transaction edges in the serialization graph of server transactions. Dependent transaction edges are those edges that correspond to transactions that R "directly or indirectly reads from".*

Proposition 4. *If a transaction R is C4 consistent, then R is overlapping current with respect to* t_{min} *of Corollary 2.*

The BCC-T1 method [10] provides an implementation of the C4 test. 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 R, 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 whether the item read has timestamp less than *count*. If this does not hold, R is aborted.

The SGT method [13] provides an implementation of the C3 test. The server maintains a serialization graph SG with all transactions committed at the server. The server broadcasts periodically 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 [16] provides an interesting implementation of the C2 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.

5 Conclusions

In this paper, we have proposed a general theory for characterizing the temporal and semantic coherency of transactions for an extended client/server environment in which the server broadcasts items of interest to a large number of clients. 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 extended 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 to caches. if the values in cache are maintained current. It can also be extended for deferred cache update policies.

References

1. 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.
2. D. Barbará. Certification Reports: Supporting Transactions in Wireless Systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 1997.

3. P. A. Bernstein, V. Hadjilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addisson-Wesley, 1987.
4. P. M. Bober and M. J. Carey. Multiversion Query Locking. In *Proceedings of the 1992 SIGMOD Conference*, pages 497–510, 1992.
5. A. Chan and R. Gray. Implementing Distributed Read-Only Transactions. *IEEE Transactions on Software Engineering*, 11(2):205–212, 1985.
6. 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.
7. H. Garcia-Molina and G. Wiederhold. Read-Only Transactions in a Distributed Database. *ACM TODS*, 7(2):209–234, 1982.
8. G. Herman, G. Gopal, K.C. Lee, A. Weinreb, "The Datacycle Architecture for Very High Throughput Database Systems," Proceedings of the ACM SIGMOD Conference, New York, 1987.
9. 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.
10. V. C. S. Lee, S. H. Son, and K. Lam. On the Performance of Transaction Processing in Broadcast Environments. In *Proccedings of the International Conference on Mobile Data Access (MDA'99)*, 1999.
11. B. Oki, M. Pfluegl, A. Siegel, D. Skeen, "The Information Bus - An Architecture for Extensible Distributed Systems," Proceedings of the SOSP Conference, North Carolina, December 1993.
12. E. Pitoura and P. K. Chrysanthis. Exploiting Versions for Handling Updates in Broadcast Disks. In *Proceedings of 25th VLDB*, pages 114–125, 1999.
13. 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*, 1999.
14. E. Pitoura, P. K. Chrysanthis, and K. Ramamritham. Characterizing the Semantic and Temporal Coherency of Broadcast-Based Data Dissemination (extended version). Technical Report TR: 2002-14, Univ. of Ioannina, Computer Science Dept, 2002.
15. 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.
16. 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.
17. S. Shekar, D. Liu, "Genesis and Advanced Traveler Information Systems (ATIS): Killer Applications for Mobile Computing," MOBIDATA Workshop, New Jersey, 1994.
18. R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 236–246, 1985.
19. White Paper, Vitria Technology Inc. (http://www.vitria.com).
20. W. E. Weihl. Distributed Version Management for Read-Only Actions. *ACM Transactions on Software Engineering*, 13(1):56–64, 1987.
21. P. Xuan, S. Sen, O.J.Gonzalez-Gomez, J. Fernandez and K. Ramamritham, "Broadcast on Demand - Efficient and Timely Dissemination of Data in Mobile Environments," IEEE Real-Time Technology and Applications Symposium, pp. 38-48, June 1997.