Scalable Invalidation-Based Processing of Queries in Broadcast Push Delivery*

Evaggelia Pitoura
Department of Computer Science
University of Ioannina
GR 45110 Ioannina, Greece
Phone : + 30 (651) 97 311
Fax : + 30 (651) 48 131
email: pitoura@cs.uoi.gr

Abstract

Recently, push-based delivery has attracted considerable attention as a means of disseminating information to large client populations in both wired and wireless environments. In this paper, we address the problem of ensuring the consistency and currency of client's read-only transactions in the case of updates. Our approach is based on broadcasting invalidation reports and is scalable, i.e., its performance and associated overhead is independent of the number of clients. Caching is also considered to improve the latency of queries. Preliminary performance results show the relative advantages of the proposed techniques.

Keywords: broadcast, push-based delivery, transaction management

1 Introduction

In traditional client/server systems, data are delivered on demand. A client explicitly requests data items from the server. Upon receipt of a data request, the server locates the information of interest and returns it to the client. This form of data delivery is called *pull-based*. In wireless computing, the stationary server machines are provided with a relative high-bandwidth channel which supports broadcast delivery to all mobile clients located inside the geographical region it covers. This facility provides the infrastructure for a new form of data delivery called *push-based* delivery. In push-based data delivery, the server repetitively broadcasts data to a client population without a specific request. Clients monitor the broadcast and retrieve the data items they need as they arrive on the broadcast channel.

Push-based delivery is important for a wide range of applications that involve dissemination of information to a large number of clients. Dissemination-based applications include information feeds such as stock quotes and sport tickets, electronic newsletters, mailing lists,

*University of Ioannina, Computer Science Department, Technical Report No: 98-021. Submitted for publication.

road traffic management systems, and cable TV. Important are also electronic commerce applications such as auctions or electronic tendering. Finally, information dissemination on the Internet has gained significant attention (e.g., [9, 25]). Many commercial products have been developed that provide wireless dissemination of Internet-available information. For instance, the AirMedia's Live Internet broadcast network [3] wirelessly broadcasts customized news and information to subscribers equipped with a receiver antenna connected to their personal computer. Similarly, Hughes Network Systems' DirectPC [23] network downloads content directly from web servers on the Internet to a satellite network and then to the subscribers' personal computer.

The concept of broadcast data delivery is not new. Early work has been contacted in the area of Teletext and Videotex systems [5, 24]. Previous work also includes the Datacycle project [10] at Bellcore and the Boston Community Information System (BCIS) [12]. In Datacycle, a database circulates on a high bandwidth network (140 Mbps). Users query the database by filtering relevant information via a special massively parallel transceiver. BCIS broadcast news and information over an FM channel to clients with personal computers equipped with radio receivers.

Recently, broadcast has received considerable attention in the area of mobile computing because of the physical support for broadcast in both satellite and cellular networks. Broadcast delivery in mobile wireless computing poses a number of difficulties. Mobile clients are resource-poor in comparison to stationary servers. Energy conservation is a major concern. The communication environment is asymmetric, in that there is typically more communication capacity from servers to clients than in the opposite direction.

In this paper, we address the problem of preserving the consistency of clients' queries, that is clients' read-only transactions, when the values of the data that are being broadcast are updated at the server. Our approach is based on broadcasting invalidation reports. Broadcasting additional information in the form of a serialization graph is also exploited to increase the concurrency of the scheme in the expense of additional processing at both the client and the server. Consistency is preserved without contacting the server thus the approach is scalable; i.e., performance is independent of the number of clients. This property makes the approach appropriate for highly populated service areas. The proposed techniques are extended to support caching at the client. Preliminary performance results show the relative advantages of the techniques presented in terms of the percentage of acceptable queries.

The remainder of this paper is organized as follows. In Section 2, we introduce th problem of supporting consistent read-only transactions in the presence of updates. In Section 3, we present two invalidation-based schemes: one based on simply broadcasting invalidation lists and one that in addition broadcasts serialization information. In Section 4, we consider client disconnections, i.e., non-continuous access to the broadcast, and discuss possible extensions. Caching is considered in Section 5, while in Section 6, we present preliminary performance results. In Section 7, we discuss related work and in Section 8, we offer our conclusions.

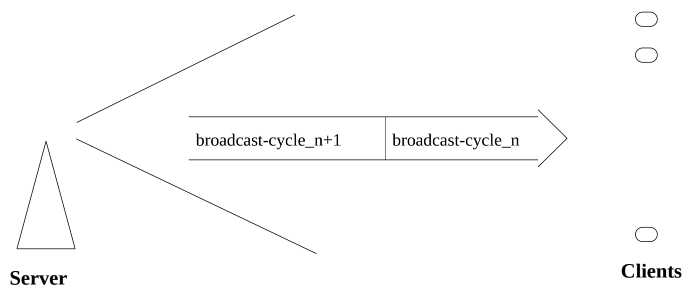

Figure 1: Broadcast-based data delivery

2 Processing Queries at the Client

2.1 The Model

The server periodically broadcasts all data items to a large client population. Each period of broadcast is called a broadcast *cycle* or *bcycle*, while the content of the broadcast is called a *bcast*. Each client listens to the broadcast and fetches data as they arrive; clients cannot make any direct requests for data (Figure 1). 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.

We assume that all updates are performed at the server. Clients access data from the broadcast in a read-only manner. Any updates are applied at the server and disseminated from there. Providing transaction support tailored to read-only transactions is important for many reasons. First, a large number of transactions in dissemination systems are read-only. Then, even if we allow update transactions at the client, it is more efficient to process read-only transactions with special algorithms. That is because consistency of queries is ensured without contacting the server. This is important because even if a backchannel exists from the client to the server, this channel typically has small communication capacity. Furthermore, since the number of clients supported is large, there is a great chance of overwhelming the server with clients' requests. In addition, avoiding contacting the server decreases the latency of client transactions.

Clients do not need to continuously listen to the broadcast. Instead, they tune-in to read specific items. 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 item of interest appears on the channel. Alternatively, the broadcast can be self-descriptive, in that, some form of directory information is broadcast along with data. In this case, the client first gets this information from the broadcast and use it in subsequent reads. Techniques for broadcasting index information along with data are given in [14, 15, 13].

The smallest logical unit of a broadcast is called a *bucket*. Buckets are the analog to blocks for disks. Each bucket has a header that includes useful information. The exact content of the bucket header depends on the specific broadcast organization. Information in the header usually includes the position of the bucket in the broadcast cycle as an offset from the beginning of the broadcast cycle as well as the offset to the beginning of the next broadcast cycle. The offset to the beginning of the next broadcast cycle can be used by the client to determine the beginning of the next broadcast cycle when the size of the broadcast is not fixed. Data items correspond to database records (tuples). We assume that users access data by specifying the value of one attribute of the record, the search key. Each bucket contains several items.

2.2 Updates and Broadcast

We assume that the server broadcasts the content of a database. A database consists of a finite set of data items. A database state is typically defined as a mapping of every data to a value of its domain. Thus, a databases state, denoted DS, can be defined as a set of ordered pairs of data items in D and their values. In a database, data are related by a number of restrictions called integrity constraints that express relationships of values of data that a database state must satisfy. A database state is consistent if it does not violate the integrity constraints [8].

While data items are being broadcast, transactions are executed at the server that may update the values of the items broadcast. We assume that the contents of the broadcast at each cycle is guaranteed to be consistent. In particular, we assume that the values of data items that are broadcast during each broadcast cycle correspond to the state of the database at the beginning of the broadcast cycle, i.e., the values produced by all transactions that have been committed by the beginning of the cycle. Thus, a read-only transaction that reads all its data within a single cycle can be executed without any concurrency overhead at all. We make this assumption for clarity of presentation, we later discuss how it can be raised.

Since the set of items read by a transaction is not known at static time and access to data is sequential, transactions may have to read data items from different broadcast cycles, that is values from different database states. As a very simple example, say T be a transaction that corresponds to the following program:

```
if a > 0 then read b else read c
```

and that b and c precede a in the broadcast. Then, a client's transaction has to read a first and wait for the next cycle to read the value of b or c.

We define the *span* of a transaction T, $span(T)$, to be the maximum number of different broadcast cycles from which T reads data. The above example shows that the order in which transactions read data affects the response time of queries. A form of transaction optimization that orders requests for data based on the order according to which they are broadcast can be employed to keep the transaction's span small.

Since client transactions read data from different cycles, there is no guarantee that the values they read are consistent. We define the *readset* of a transaction T, denoted $Read\_Set(T)$, to be the set of items it reads. In particular, $Read\_Set(T)$ is a set of ordered pairs of data items and their values that T read. Our correctness criterion for read-only transactions is that each transaction reads consistent data. Specifically, the readset of each read-only transaction must form a subset of a consistent database state [21]. We assume that each server transaction preserves database consistency. Thus, a state produced by a serializable execution (i.e., an execution equivalent to a serial one [8]) of a number of server transactions produces a consistent database state. The goal of the methods presented in this paper is to ensure that the readset of each read-only transaction corresponds to such a state.

To guarantee correctness, additional control information is broadcast along with data. Processing of control information is required at both the client and the server. The server computes and broadcasts this information during each broadcast cycle. The client reads this information from the broadcast channel and interpret it appropriately. The size of the control information is an important measure of the efficiency of a transaction processing scheme, since transmitting control information consumes bandwidth. Furthermore, the volume of the broadcast data affects the response time of client transactions. Since access to data is sequential, the larger the volume of the broadcast, the longer the clients need to wait until the data of interest appear on the channel. Another requirement is minimizing the overhead of processing this information at both the server and at the clients.

3 Invalidation-Based Query Processing

3.1 Invalidation-Only Scheme

Each bcast is preceded by an invalidation report in the form of a list that includes all data items that were updated during the previous broadcast cycle. For each active read-only transaction R, the client keeps a set $RS(R)$ of all data items that R has read so far. At the beginning of each broadcast cycle, the client tunes in and reads the invalidation report. A read transaction R is aborted if an item $x \in RS(R)$ was updated, that is if x appears in the invalidation report. Clearly,

Theorem 1 *The invalidation only method produces correct read-only transactions.*

Proof. Let c_c be the cycle during which a committed read-only transaction R performed its last read operation and DS^{c_c} be the database state broadcasted at cycle c_c. Then, the values read by R correspond to the database state DS^{c_c}. For the purposes of contradiction, assume that a value of a data item x read by R corresponds to a database state broadcasted at a previous cycle, then an invalidation report should have been transmitted for x and thus R should have been aborted. $\square$

As indicated by the proof above, in the invalidation-only scheme, a read-only transaction R reads the most current values, that is the values produced by all transactions committed at the beginning of the broadcast cycle at which R commits. The increase in the size of the broadcast is equal to $u\,d$, where u is the number of items that were updated and d is the size of the *key*.

Bounded-Inconsistency

One approach to increase concurrency and reduce the overhead of transmitting and processing control information is to provide queries that can tolerate a degree of inconsistency.

There are many definitions of inconsistency and correspondingly different techniques for enforcing them. We describe next two of them for illustration.

One formal characterization of inconsistency is provided by *epsilon-serializability* (ESR) [20, 19]. In epsilon-serializability, each read-only transaction has an import-limit that specifies the maximum amount of inconsistency that it can accept. ESR associates an amount of inconsistency with each inconsistent state defined as its distance from a consistent state. It has meaning for any state that processes a distance function.

Let R be a read-only transaction and c_0 be the cycle at which R starts. The import limit for R can be quantified on a per data item basis. Let $x \in RS(S)$, then the inconsistency associated with x can be defined as the distance of the current value of x and the value of x at c_0 say x_0. R can tolerate reading x_0, and thus import inconsistency equal to this distance, if the distance is within a specified import limit. The inconsistency imported by R depends on the number of concurrent updates, i.e., the number of server transactions that commit while the read-only transaction R is in progress. One way to support this form of imported inconsistency is to extend the validation report for a data item x to include the number of transactions that have updated x during the previous broadcast cycle.

There are other ways to quantify import inconsistency [4]. For example, for a data item x that takes numerical values, instead of transmitting an invalidation report each time it is updated, we may transmit an invalidation report only when the difference of its new value from the old one falls outside a specified range of values.

3.2 Serialization-Graph Testing

If additional information is broadcasted, it should be possible to accept more read-only transactions. To this end, we develop a serialization graph testing (SGT) method. The serialization graph for a history H, denoted $SG(H)$, is a directed graph whose nodes are the committed transactions in H and whose edges are all $T_i \rightarrow T_j$ $(i \neq j)$ such that one of T_i's operations precedes and conflicts with one of T_j operations in H [8]. According to the serialization theorem, a history H is serializable iff $SG(H)$ is acyclic. We assume that each transaction reads a data item before it writes it, that is, the readset of a transaction includes its writeset. Then, in the serialization graph, there can be two types of edges $T_i \rightarrow T_j$ between any pair of transactions T_i and T_j: *dependency* edges that express the fact that T_j read the value written by T_i and *precedence* edges that express the fact that T_j wrote an item that was previously read by T_i.

In brief, the SGT method works as follows. Each client maintains a copy of the serialization graph locally. The serialization graph at the server includes all transactions *committed* at the server, while, in addition, the local copy at the client includes any active read-only transactions that were issued at this site. At each cycle, the server broadcasts any updates of the serialization graph. Upon receipt of the updates, the client integrates them into its local copy of the graph. A read operation at a client is executed only if it does not create a cycle in the local serialization graph.

Implementation of the SGT Method

In the serialization graph testing (SGT) method, the server broadcasts at the beginning of each bcast the following control information:

- the *difference from the previous serialization graph*
  In particular, the server broadcasts for each transaction T_i that was committed during the previous cycle, a list of the transactions with which it conflicts, i.e., it is connected through a direct edge.

- an *augmented invalidation report*
  The report includes all data written during the previous broadcast cycle along with an identification of the first transaction that wrote each of them during the bcycle.

In addition, the content of the broadcast is augmented so that the identification of the last transaction that wrote a data item is broadcast along with the item.

Each client tunes in at the beginning of the broadcast to obtain the control information. Upon receipt of the graph, the client updates its local copy of the serialization graph SG to include any additional edges and nodes. Let SG^i be the subgraph of SG that includes only the transactions that were committed during cycle i. An interesting property is that:

Claim 1 *There cannot be any incoming edges to transactions in SG^i from transactions committed in subsequent cycles $m > i$.*

This is true since all transactions in SG^i are committed prior to any transactions committed in subsequent cycles.

The client also adds precedence edges for all its active read-only transactions as follows. Let R be such an active transaction and RS^i be the set of items that R has read so far. For each item x in the invalidation report such that $x \in RS^i$, the client adds a precedence edge $R \rightarrow T_f$, where T_f is the first transaction that wrote x during the previous cycle. Although R conflicts with all transactions that wrote x during the previous cycle, it suffices to add just one edge to T_f since:

Claim 2 *Let $x \in RS(R)$ and SG_a be the graph that includes edges $R \rightarrow T$ for each T that wrote x during bcycle $i-1$ and SG_f the subgraph of SG that includes only an edge $R \rightarrow T_f$, where T_f is the first transaction that wrote x during the bcycle $i-1$. SG_a has a cycle if and only if SG_f has a cycle.*

Proof.
($\Leftarrow$) If SG_f has a cycle then SG_a has a cycle since SG_f is a subgraph of SG.
($\Rightarrow$) Let SG_a have a cycle. Assume for the purposes of contradiction that SG_f is acyclic. Then, the cycle of SG_a must include an edge that does not belong to SG_f. This must be an edge $R \rightarrow T'$, where T' is a transaction other than T_f that wrote x. Since T' wrote x after T_f, there is an edge $T_f \rightarrow T$. Thus, there is a path $R \rightarrow T_f \rightarrow T'$ in SG_f. By similar arguments, we can replace any edge in a cycle of the SG_a that does not exist in SG_f by a corresponding path. Thus, SG_f also has a cycle, a contradiction. □

When R reads an item y, a dependency edge $T_l \rightarrow R$ is added in the local serialization graph, where T_l is the last transaction that wrote y. The read operation is accepted, only if no cycle is formed. It can be shown with an argument similar to the one in the previous claim that it suffices to add just an edge $T_l \rightarrow R$ instead of adding edges $T' \rightarrow R$ from all transactions T' that wrote y.

Claim 3 *Let $y \in RS(R)$ and SG_a be the graph that includes edges $T \rightarrow R$ for each T that wrote y and SG_l be the subgraph that includes only an edge $T_l \rightarrow R$, where T_l is the last transaction that wrote y. SG_a has a cycle if and only if SG_l has a cycle.*

We will prove that the SGT method detects all cycles that include a read-only transaction R. We will use the following lemma:

Lemma 1 *Let o be the broadcast cycle during which R performs its first read operation.*

(a) During broadcast cycle m, the only types of cycles that include R are of the form $R \rightarrow T_{i_1} \rightarrow T_{i_2} \rightarrow \ldots T_{i_k} \rightarrow R$, where for any consecutive $T_{i_m} \in SG_k$ and $T_{i_{m+1}} \in SG_l$, it holds $o \leq k \leq l$.

(b) The SGT algorithm detects all such cycles.

Proof.
(a) For R to be involved in a cycle, R must have both an incoming and an outgoing edge. An outgoing edge $R \rightarrow T_{i_1} \in SG_n$ is to a transaction T_{i_1} that overwrote an item read by R, thus $n \geq o$. From Claim 1, the edges among subgraphs of transactions committed at different broadcast cycles go from transactions committed at previous cycles to transactions committed at subsequent cycles, thus $k \leq l$. The outgoing edge $T_{i_k} \rightarrow R$ is from the last transaction T_{i_k} that wrote an item read by R.
(b) To prove that the SGT algorithm detects all such cycles, we must show that: (i) the local serialization graph at the client includes all such cycles and, (ii) the SGT algorithm detects them. (i) Since the server broadcasts all edges and nodes involving server transactions, it suffices to show that all incoming and outgoing edges to R are included in the local graph. From Claims 2 and 3, it suffices to include edges that involve T_f and T_l, as the SGT does. (ii) Since the graph at the server is acyclic, cycles may be created only when an incoming or outgoing edge to R is added. We claim that such cycles may be formed only when an incoming edge to R is added, that is only when an item is read. Assume for the purposes of contradiction that the addition of an outgoing, i.e., precedence, edge $R \rightarrow T_j$ can create a cycle. Such edges are added at the beginning of each broadcast cycle k, for each transaction $T_j \in SG_k$, where T_j is the first transaction that overwrote an item previously read by R. For a cycle to be formed, there must also be an edge $T_i \rightarrow R$, where from part (a), $T_i \in SG_m$ with $m \geq k$, that means that R read an item from the last transaction T_i that wrote this item during broadcast cycle m, which is impossible since no values produced during broadcast cycles $m \; geq \; k$ have been read yet. □

Theorem 2 *The SGT method produces correct read-only transactions.*

Proof. From Lemma 1, the SGT algorithm detects all cycles that involve R, thus from a direct application of the serialization theorem, R is serializable with all server transactions, thus R reads consistent data and is correct. □

Regarding the currency of read-only transactions, each read-only transaction R that performs its first read at c_0 reads values that correspond to a database state between the state at the beginning of broadcast cycle c_0 and the current database state.

Space Efficiency

Instead of keeping a complete copy of the serialization graph locally at each client, by Lemma 1, it suffices to keep for each query R only the subgraphs SG_k with $k \geq c_o$, where c_o is the bcycle when the first item read by R is invalidated, i.e., overwritten. Thus, if no items are updated, there is no space or processing overhead at the client. Furthermore, at most S subgraphs are maintained, where S is the maximum transaction span of the queries at the client. By Lemma 1, we may also keep only the outgoing edges from R; there is no need to store the incoming edges to R.

However, the volume of the control information that is broadcasted is considerable. Let tid be the size of a transaction identifier, c be the maximum number of transactions committed during a broadcast cycle, and N be the maximum number of operations per transaction at the server. We assume that transaction identifiers are unique within each broadcast cycle, thus it suffices to allocate $log(c)$ bits per transaction identifier. Then, when there is a need to distinguish between transactions at different cycles, we also broadcast a version number indicating the broadcast cycle at which the transaction was committed. The size of the invalidation report is: $u(i + log(c))$. Since, there are at most N operations per transaction, each transaction may participate in at most N conflicts with other transactions. Thus, the difference from the previous graph has at most $c\,N$ edges. The total size of the difference is: $cN(2log(c)+2v)$, assuming that along with each transaction we also broadcast the broadcast cycle at which it was committed. Finally, the size of the broadcast is further augmented, since along with each item, we must also broadcast the identifier of the last transaction that wrote it.

4 Discussion

4.1 Disconnections

Listening to the broadcast consumes energy. In addition, access to the broadcast data may be monetarily expensive. Thus, mobile clients may voluntary skip listening for a number of broadcast cycles. Besides this voluntary form of disconnection, client disconnections are very common when broadcast data are delivered wirelessly. Wireless communications face many obstacles because the surrounding environment interacts heavily with the signal, thus in general wireless communications are less reliable and deliver less bandwidth than wireline communications. Thus, a desirable requirement from a broadcasting scheme is to allow clients to continue their operation after periods during which the clients miss listening to the broadcast signal.

The techniques presented require active clients to monitor the broadcast continuously. In the invalidation-only scheme, a client has to tune-in at each and every cycle to read the invalidation report. Otherwise, it cannot ensure the correctness of any active read-only transaction. Similarly, the SGT method does not tolerate any client disconnection. If a client misses a broadcast cycle, it cannot anymore guarantee serializability. Thus, any active read transactions must be reissued anew. An enhancement of the scheme to increase tolerance to disconnections would be to broadcast along with items version numbers. Then, a read operation could be accepted as long as its version number was smaller than the version of the last broadcast that the transaction has listen to. This guarantees that the client has all the information required for cycle detection.

In all the schemes, periodic retransmission of control information would increase their tolerance to intermittent connectivity. For instance, an invalidation report of the items updated during the last w bcycles may be broadcasted to allow clients to resynchronize.

4.2 Extensions

Instead of the invalidation reports being broadcasted at the beginning of each broadcast cycle, such reports can as well be broadcasted at other pre-specified intervals h, $h \leq T$, where T is the length of the broadcast. In this case, the invalidation report must include all items updated during h. The values broadcast correspond to the values produced by all transactions committed by the beginning of the current interval.

Another way to extend the scheme is to increase its granularity. For example, invalidation reports may include buckets instead of items. A bucket is considered updated when any of the items that it includes has been updated. Instead of maintaining for each transaction R the set of items it has read, the set of buckets is maintained. Then, a query is aborted if one of the buckets it has read is subsequently updated. This scheme may lead to aborting queries that normally should not have been aborted, since the invalidation of a bucket does not necessarily means that the specific item read has also been updated. However, it accepts only correct queries and imposes less overhead.

5 Caching

To reduce latency in answering queries, clients can cache items of interest locally. Caching reduces not only the latency but also the span of transactions, since transactions find data of interest in their local cache and thus need to access the broadcast channel for a smaller number of cycles. When broadcast items are updated, the value of cached items become stale. We assume that each page, i.e., the unit of caching, corresponds to a bucket, i.e., the unit of broadcast.

There are various approaches to communicating updates to the client. The two basic techniques are invalidation and propagation. For invalidation, the server sends out messages to inform the client of which pages are modified. The client removes those pages from its cache. For propagation, the servers sends the updated values. The client replaces its old copy with the new one.

Invalidation combined with a form of autoprefetching has been shown to perform well in broadcast delivery [2]. At the beginning of each broadcast cycle (or at other pre-defined points), the server broadcasts an invalidation report, which is a list of pages that have been updated. This report is used to invalidate those pages in cache that appear in the invalidation report. These pages remain in cache to be autoprefetched later. In particular, when the new value of an invalidated page appears in the broadcast, the client fetches the new value and replaces the old one. Thus, a page in cache either has a current value (the one in the current broadcast) or is marked for autoprefetching.

The cache invalidation report is similar to the invalidation report used in our query processing schemes. However, the two reports differ in granularity. The cache invalidation report includes the pages (or buckets) that have been updated, whereas the query-processing invalidation report includes the data items that have been updated. As discussed, the query processing techniques can be modified to work on pages or buckets rather on items. We describe next how the techniques can be extended to work in conjunction with caching, while keeping their granularity at the item level.

For the invalidation only scheme, each read first checks whether the item is in cache. If it is found in cache and the page is not invalidated, then it is read from the cache. Otherwise, the item is read from the broadcast.

To enhance the simple-invalidation scheme, the cache is extended so that along with each value it also includes the bcycle during which the value was inserted in the cache. Let R be a query and u the first bcycle at which an item $x \in RS(R)$ is invalidated. Instead of aborting R, R is marked abort and continues operation as long as old enough values for all future reads can be found in the cache. In particular, R continues its read operations as long as the items it wants to read exist in the cache and have versions $c < u$. We call this method, simple-invalidation with versioned cache.

Theorem 3 *The invalidation only method with versioned cache produces correct read-only transactions.*

Proof. Let R be a committed query and u be the first bcycle at which an item read by R was invalidated. Let DS^{u-1} be the database state broadcasted at bcycle $u-1$. Then, the values read by R correspond to the database state DS^{u-1}. This holds because all the values read by R till bcycle u correspond to DS^{u-1}, then a value is read only if the version in cache is $c < u$. This value is the value the item has at $u-1$ otherwise it should have been invalidated and a new version should have been autoprefetched. □

For the SGT method, the cache must be extended to include for each item the last transaction that wrote it; information that is broadcasted anyway. Each time an item is read from the cache, the same test for cycles as when the item is read from the broadcast is executed.

6 Performance Evaluation

6.1 The Model

Our performance model is similar to the one presented in [1]. The server periodically broadcasts a set of data items in the range of 1 to $BroadcastSize$. We assume for simplicity a flat broadcast organization in which the server broadcasts cyclicly the set of items.

The client accesses items from the range 1 to $ReadRange$, which is a subset of the items broadcast ($ReadRange \leq BroadcastSize$). Within this range, the access probabilities follow a Zipf distribution. The Zipf distribution with a parameter *theta* is often used to model non-uniform access. It produces access patterns that become increasingly skewed as *theta* increases. The client waits $ThinkTime$ units and then makes the next read request.

Updates at the server are generated following a Zipf distribution similar to the read access distribution at the client. The write distribution is across the range 1 to $UpdateRange$.

Server Parameters		**Client Parameters**	
BroadcastSize	1000	ReadRange (range of client reads)	250
UpdateRange	500	theta (zipf distribution parameter)	0.95
theta (zipf distribution parameter)	0.95	Think Time (time between client reads in broadcast units)	2
Offset (update and client-read access deviation)	0 - 250 (100)	Number of Reads per Quey	5 - 50 (10)
ServerReadRange	1000	**Cache**	
N (number of server transactions)	10	CacheSize	125
Offset (update and server-read access deviation)	0	Cache replacemnt policy	LRU
UpdateNumber	20 - 500 (100)		
ReadNumber	4*UpdateNumber		

Figure 2: Parameters

We use a parameter called $Offset$ to model disagreement between the client access pattern and the server update pattern. When the offset is zero, the overlap between the two distributions is the greatest, that is the client's hottest pages are also the most frequently updated. An offset of k shifts the update distribution k items making them of less interest to the client. We assume that during each $bcycle$, N transactions are committed at the server. All server transactions have the same number of update operations and read operations, where read operations is four times more frequent than updates. Read operations at the server are in the range 1 to $BroadcastSize$, follow a Zipf distribution, and have zero offset with the update set at the server.

The client maintains a local cache that can hold up to $CacheSize$ pages. The cache replacement policy is LRU: when the cache is full, the least recently used page is replaced. When pages are updated, the corresponding cache entries are invalidated and subsequently autoprefetched. Table 2 summarizes the model parameters. Values in parenthesis are the default.

6.2 Experimental Results

To evaluate the various schemes, we considered the percentage of transactions that are aborted. First we varied the number of read operations per query (Figure 3). Whereas the SGT method with caching outperforms all other schemes, the invalidation-only scheme with versioned cache seems to offer an attractive alternative for queries with less than 30 reads, thus avoiding the considerable overhead of the SGT method. Caching reduces the number of transactions aborted since it reduces their span and thus the probability of invalidation.

Then, we considered the number of updates (Figure 4). In this case, the invalidation-only scheme with versioned cache outperforms all other schemes for a large number of updates (over 1/4 of the $BroadcastSize$). This is because the possibility of cycles in the serialization graph increases with the number of operations at the server. In general, the SGT methods are less attractive than the invalidation-only methods when there is a lot of activity in the server. Thus, while for a small number of operations at the server the SGT methods more than double the number of queries that are accepted, when the number of operations at the server increases, the increase of the accepted transaction decreases to 10%

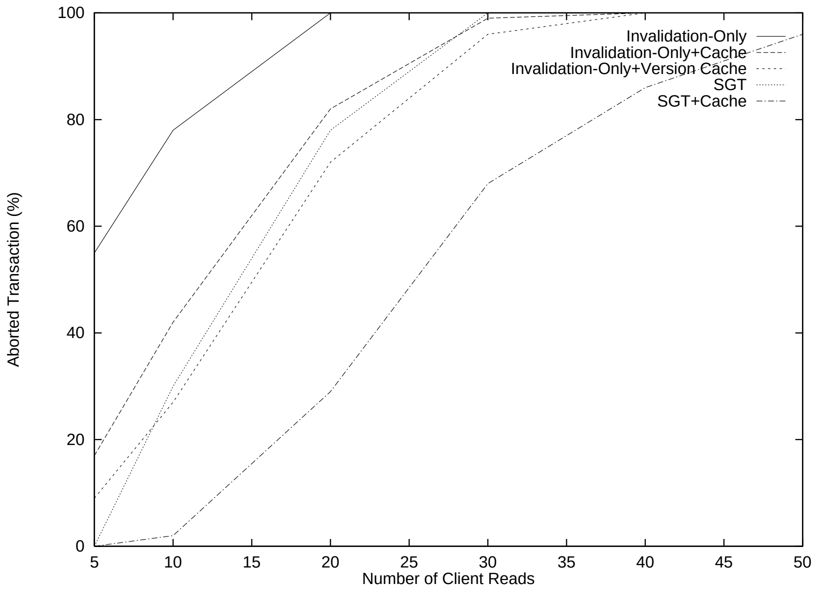

Figure 3: Number of Reads per Query

(Figure 4).

Finally, we considered the overlap between the client read and the server update pattern. As expected, when the overlap is the maximum, that is the client's hot data are those that are most frequently updated, all schemes have the highest abort rates. When the overlap is small (less than 50%), the SGT methods accept all transactions.

7 Related Work

Recently, there has been considerable interest on broadcast delivery (for a review, see for example [11] and Chapter 4 of [18]). Updates have been mainly treated in the context of caching. In this case, clients maintain a local cache of the data of interest. Invalidating cache entries by broadcast is the focus of much current research including [7], [2], and [16]. Updates are considered in terms of local cache consistency; there are no transaction semantics. In [17], we have first introduced the problem of maintaining the consistency of queries. In the current paper, we present specific invalidation-based techniques, prove their correctness, relate them to caching and present preliminary performance results.

A weaker alternative to serializability for transactions in broadcast systems is proposed in [22]. In this work, read only transactions have similar semantics with weak transactions in the conflict-serializability approach. However, the emphasis is on developing and formalizing a weaker serializability criterion rather than on protocols for enforcing them. Finally, broadcast in transaction management is also used in the certification-report method [6]. Read-only transactions in the certification-report method are similar to read-only transactions in the invalidation-only method. However, in the certification-report method, data delivery is on demand, the broadcast medium is mainly used by the server to broadcast concurrency control information to its clients.

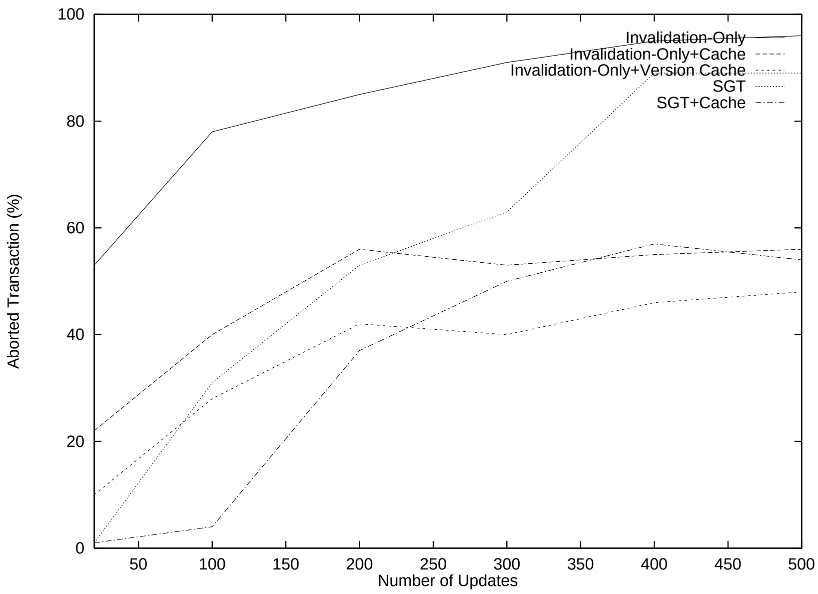

Figure 4: Number of Updates

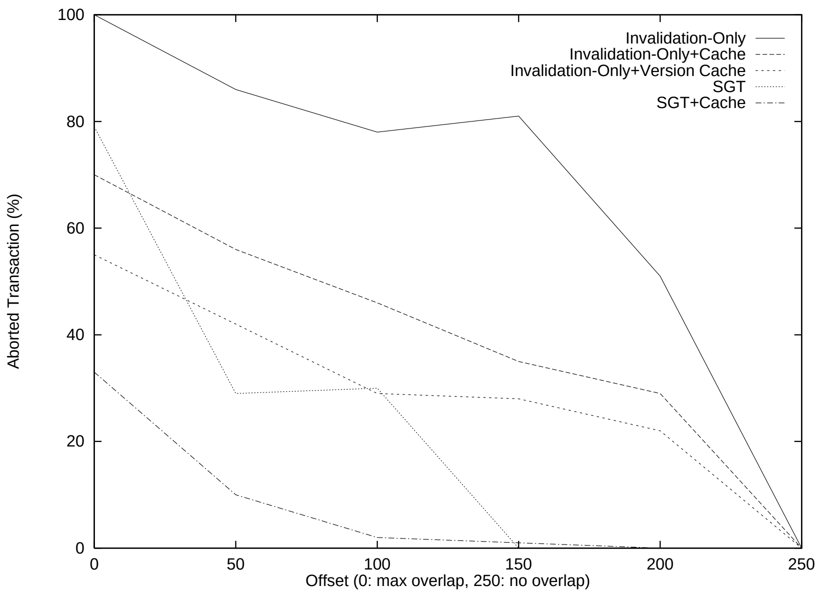

Figure 5: Offset, update and read access deviation

8 Conclusions and Future Work

We have presented a set of invalidation-based methods that provide support for consistent queries. The methods are scalable in that their performance is independent of the number of clients. The methods were presented for a flat broadcast organization, in which all items are broadcasted with the same frequency. One possible extension is to consider a broadcast-disk organization [1], where specific items are broadcasted more frequently than others, i.e., are placed on "faster disks". An interesting problem related to the query processing methods is determining the optimal frequency for transmitting control information. We are also investigating the deployment of multiversion schemes for increasing the concurrency of queries.

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. Reprinted in Mobile Computing, Imielinski and Korth, Eds., Kluwer Academic Publishers, 1996.

[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] AirMedia. AirMedia Live. www.airmedia.com.

[4] R. Alonso, D. Barbará, and H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM Transactions on Database Systems*, 15(3):359–384, September 1990.

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

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

[7] 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.

[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 196.

[10] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, and A. Weinrib. The Datacycle Architecture. *Commuinications of the ACM*, 35(12), December 1992.

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

[12] D. Gifford. Polychannel Systems for Mass Digital Communication. *Communications of the ACM*, 33(2), 1990.

[13] T. Imielinski and J. C. Navas. Geographic Addressing, Routing, and Resource Discovery with the Global Positioning System. *Communications of the ACM*, 1997. To appear.

[14] T. Imielinski, S. Viswanathan, and B. R. Badrinanth. Energy Efficient Indexing on Air. In *Proceedings of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD 94)*, pages 25–36, 1994.

[15] T. Imielinski, S. Viswanathan, and B. R. Badrinanth. Power Efficient Filtering of Data on Air. In *Proceedings of the the 4th International Conference on Extending Database Technology*, March 1994.

[16] 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.

[17] E. Pitoura. Supporting Read-Only Transactions in Wireless Broadcasting. In *Proceedings of the DEXA98 International Workshop on Mobility in Databases and Distributed Systems*, August 1998.

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

[19] C. Pu and A. Leff. Replica Control in Distributed Systems: An Asynchronous Approach. In *Proceedings of the ACM SIGMOD*, pages 377–386, 1991.

[20] K. Ramamritham and C. Pu. A Formal Characterization of Epsilon Serializability. *IEEE TRansactions on Knowledge and Data Engineering*, 7(6):997–1007, 1995.

[21] 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.

[22] J. Shanmugasundaram, A. Nithrakasyap, J. Padhye, R. Sivasankaran, M. Xiong, and K. Ramamritham. Transaction Processing in Broadcast Disk Environments. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures.* Kluwer, 1997.

[23] Hughes Network Systems. DirectPC Homepage. www.direcpc.com, 1997.

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

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