1995

Bit-Sequences: A New Cache Invalidation Method in Mobile Environments

Jin Jing

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Abdelsalam Helal

Rafael Alonso

Report Number:
95-076

BIT-SEQUENCES: A NEW CACHE INVALIDATION METHOD IN MOBILE ENVIRONMENTS

Jin Jing
Ahmed Elmagarmid
Abdelsalam Helal
Rafael Alonso

CSD TR-95-076
December 1995

Bit-Sequences: A New Cache Invalidation Method in Mobile Environments

Jin Jing, Ahmed Elmagarmid, Abdelsalam Helal
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA
{jing,ake,helal}@cs.purdue.edu

Rafael Alonso
Matsushita Information Technology
Laboratory, 2 Research Way
Princeton, NJ 08540 USA
alonso@research.panasonic.com

Abstract

In this paper, we address the problem of cache invalidation in mobile environments. We present *Bit-Sequences*, a new cache invalidation algorithm in which a periodically-broadcast invalidation report is organized as a set of binary bit sequences with an associated set of timestamps. A scalable version of the Bit-Sequences algorithm for large databases, called *Multi-Level Bit-Sequences*, is also discussed. As demonstrated through simulation experiments, the Bit-Sequences algorithm performs consistently well under conditions of variable update rates/patterns and client disconnection times. Furthermore, the size of the invalidation report in this algorithm is relatively small and is independent of the number of data items to be invalidated.

1 Introduction

In a mobile computing environment, caching of frequently-accessed data items will be an important technique that will reduce contention on the narrow bandwidth wireless channels. However, cache invalidation strategies will be severely hampered by the disconnection and mobility of clients [7]. It is difficult for a server to send invalidation messages directly to mobile clients that have cached the data items to be updated, since these clients often disconnect to conserve battery power and are frequently on the move. On the other hand, the narrow bandwidth wireless network will be clogged if massive numbers of clients attempt to query a server to validate cached data. Therefore, the existing caching algorithms employed in the traditional client-server architecture where the locations and connections of clients do not change may not be readily applicable to mobile environments.

In [7], Barbara and Imielinski provided an alternate approach to the problem of invalidating caches in mobile environments. In this approach, a server periodically broadcasts an *invalidation report* in which the data items that have been changed are indicated. Rather than querying a server directly regarding the validation of cached copies, clients listen to these invalidation reports over wireless channels. This approach is attractive in the mobile environment because (1) a server need not know the location and connection status of its clients and (2) the clients need not establish an "uplink" connection to a server to invalidate their caches.

One interesting new issue, however, arises when clients use the invalidation report to invalidate their caches. This issue is the *false invalidation* of client caches. Since an invalidation report can include only limited or partial information regarding data changes or the current database state, caches that are actually valid may be invalidated. For example, when a client has disconnected for a substantial period of time, the report may not cover all updates to data items that have occurred since the time of disconnection. In this case, the client may invalidate a cache which is actually valid. Once a cache is invalidated, the client must seek verification or updating of the cache from the data server.

Consider the Broadcasting Timestamps (TS), Amnesic Terminals (AT), and Signatures (SIG) algorithms presented in [7]. In the TS algorithm, the invalidation report includes only the information regarding the data items that have been updated within the preceding w seconds. The report includes the names of these items and the timestamps at which they were updated. The invalidation report in the AT algorithm includes the identifiers of data items that were updated during the last broadcast period L. In both algorithms, clients must invalidate their entire cache when their disconnection period exceeds a specified length (w seconds for TS and L seconds for AT). In the SIG algorithm, the report contains a set of combined signatures of data items.[1] The structure and size of these signatures are designed to diagnose up to f differing items. If more than f different items (it does not matter whether these items had been cached or not) were updated in the data server since the combined signatures were last cached, most items cached by the clients will be invalidated by the SIG algorithm, although many are in fact valid. An explanation of this observation is offered in Appendix A. Thus, these existing algorithms are *effective* for a client in terms of a low ratio of false invalidation (to total invalidation), only if the client has not disconnected for a period that exceeds an algorithm-specified parameter (e.g., w or L) or if the number of updated items during the period is not greater than an algorithm-specified parameter (e.g., f).

In a mobile environment, it is most desirable to provide a cache invalidation algorithm whose performance will not be severely affected by such changing workload parameters as client disconnection times and update rates. Mobile units will frequently be "off", with substantial periods of disconnection intended for battery conservation. Within these substantial disconnection periods, the number of items to be updated by a server may be unpredicatable. A high rate of false invalidation will usually reduce the cache hit rate in query processing, resulting in increased traffic on the narrow wireless networks and decreased query processing throughput.

This paper introduces a new cache invalidation algorithm, which we call *Bit-Sequences* (BS). In this algorithm, the invalidation report consists of a set of binary bit sequences with an associated set of timestamps. As in other invalidation-report based algorithms, a server periodically broadcasts invalidation reports and clients listen to them over wireless channels to invalidate their caches.

[1] Signatures are checksums computed over the value of data items in the database. A combined signature is the Exclusive OR of a set of individual signatures. Each combined signature, therefore, represents a subset of data items. In the SIG algorithm, m combined signatures are computed such that an item i is in the set of combined signature S_j $(1 \leq j \leq m)$ with probability $\frac{1}{f+1}$ where f is the number of differing items up to which the algorithm can diagnose.

One salient property that the algorithm has is that changing workload parameters, such as client disconnection times and update rates/patterns, have less adverse impact on the effectiveness of this algorithm than on other algorithms.

We also discuss a *scalable* BS algorithm called *Multi-Level Bit-Sequences* for large databases. The algorithm can scale to large databases for the "information feed" application domain with skewed access pattern without the use of large invalidation report. The scalability feature is important because the large invalidation report will waste precious wireless bandwidth and increase latency for client cache verification.

To study the effects of disconnection time and update pattern on the performance of various proposed algorithms, we have implemented a simulation model of a client-server system that supports mobile hosts over wireless communication channels. In this paper, we use this simulation model to compare the false invalidation ratio and buffer hit ratio between the proposed BS algorithm and the SIG algorithm. This analysis is intended to assess the extent to which the performance of the proposed algorithm can surpass that of existing algorithms for the conditions of changing workload parameters.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the Bit-Sequences algorithm. Section 4 discusses the scalable BS algorithm with regard to large databases. In Section 5, we compare the Bit-Sequences algorithm with the SIG algorithm through a simulation. Concluding remarks are offered in Section 6.

2 Related Research

Many caching algorithms have been recently proposed for the conventional client-server architecture in which the positions and wired-network connections of clients are fixed. A comprehensive discussion and comparison of caching algorithms in the conventional client-server architecture can be found in [10]. The issue of false invalidation does not exist in this architecture because, as shown in the algorithms discussed in [10], either the server can directly invalidate client caches or clients can query the server for the validation of their caches. In both cases, only obsolete caches will be invalidated.

Recently, the notion of using a repetitive broadcast medium in wireless environments has been investigated. The property of data broadcast program which provides improved performance for non-uniformly accessed data was investigated in [2]. The authors in [2] also addressed the impact of data broadcast on the client cache fetch and replacement policies. The mobile computing group at Rutgers has investigated techniques for indexing broadcast data [14, 15]. The main motivation of this work has been to investigate ways to reduce battery power consumption at the clients for the access of broadcast data. In our approach, the invalidation report is organized in a bit indexing structure in order to save the space of broadcast channels. An approach of broadcasting data for video on demand has been addressed in [17]. The approach, called pyramid broadcast, splits an object into a number of segments of increasing sizes. To minimize latency the first segment is broadcasted more frequently

than the rest. An adaptive scheme of broadcasting data was described in [13]. The adaptability is achieved by varying the frequency of broadcast of individual data items according to the frequency of requests.

In [7], issues of cache invalidation using a broadcast medium in a wireless mobile environment were first introduced. The SIG, TS, and AT algorithms that use periodically broadcast invalidation reports were proposed for client caching invalidation in the environment. It has been shown in [7] that, when the update rate is low, the SIG algorithm is best suited for clients that are often disconnected, while the TS algorithm is advantageous for clients that are connected most of time. The AT algorithm is best suited to update intensive scenarios, provided that the clients are awake most of time. However, the *effectiveness* of these algorithms in terms of low false invalidation ratio depends heavily on such workload parameters as client disconnection time and the number of items updated during these disconnections.

To support long client disconnections, an idea of adapting the window size of the TS algorithm was discussed in [6, 8]. The approach in [6, 8] adjusts the window size for each data item according to changes in update rates and reference frequencies for the item. This is different from our proposed approach which does not need the feedback information about the access patterns from clients. In the adaptive TS approach, a client must know the exact window size for each item before using an invalidation report. These sizes must therefore be contained in each report for the client to be able to correctly invalidate its caches. [2] However, no detailed algorithm was presented in [6, 8] to show how the window size information is included in the invalidation report. For this reason, we will not compare this approach with our approach in this paper.

The work in [11] discusses the data allocation issues in mobile environments. The algorithms proposed in [11] assume that servers are *stateful* since they know about the state of the clients' caches. The algorithms use this information to decide whether a client can hold a cache copy or not to minimize the communication cost in wireless channels. In contrast, servers in our algorithm (as well as the TS, AT, and SIG algorithms) are *stateless* since they do not have the state information about clients' caches.

3 The Bit-Sequences Algorithm

3.1 Caching Management Model

A mobile computing environment consists of two distinct sets of entities: *mobile hosts* and *fixed hosts* [12, 7, 5]. Some of the fixed hosts, called *Mobile Support Stations* (MSSs), are augmented with a wireless interface to communicate with mobile hosts, which are located within a radio coverage area called a *cell*. A mobile host can move within a cell or between two cells while retaining its network

[2] Consistency problem might arise if the window size for a data item is not included in an invalidation report. Consider that the window size is decreased during a client disconnection period. After the client wakes up, the absence of information regarding the new window size may cause it to falsely conclude the data item is still valid.

connections. There is a set of database servers each covering a cell.

Each server can only service users which are currently located in its cell. A large number of mobile hosts resides in each cell; all issue queries which take the form of simple requests to read the most recent copy of a data item. We assume that the database is updated only by the servers. The database consists of N numbered data items (or pages): $d_1, d_2, ..., d_N$ and is fully replicated at each data server. The data item (or page) is the basic update and query unit by the server and client.

Each server will periodically broadcast invalidation reports. To answer a query, the client on a mobile host will listen to the next invalidation report and use the report to conclude whether its cache is valid or not. If there is a valid cached copy that can be used to answer the query, the client will return the result immediately. Invalid caches must be refreshed via a query to the server.

3.2 The Algorithm

In the Bit-Sequences (BS) algorithm, the server broadcasts a set of bit sequences. Each sequence consists of a series of binary bits and is associated with a timestamp. Each bit represents a data item in a database. A bit "1" in a sequence means that the item represented by the bit has been updated since the time specified by the timestamp of the sequence. A bit "0" means that that item has not been updated since that time.

The set of sequences is organized in a hierarchical structure. The highest-ranking sequence in the structure has a number of N bits which corresponds to N data items in the database. That is, each item is represented by one bit in this sequence. As many as half the bits ($N/2$) in the sequence can be set to "1" to indicate that up to the last $N/2$ items have been changed recently (initially, the number of "1" bits may be less than $N/2$). The timestamp of the sequence indicates the time after which these $N/2$ items have been updated. The next sequence in the structure will contain $N/2$ bits. The kth bit in the sequence corresponds to the kth "1" bit in the highest sequence (i.e., both represent the same data item). In this sequence, $N/2^2$ bits can be set to "1" to indicate the last $N/2^2$ items that were updated recently. The timestamp of the sequence indicates the time after which these $N/2^2$ items were updated. The following sequence, in turn, will contain $N/2^2$ bits. The kth bit in the sequence corresponds to the kth "1" bit in the preceding sequence. In the current sequence, $N/2^3$ bits can be set to "1" to indicate the last $N/2^3$ items that were updated recently. The timestamp of the sequence indicates the time after which these $N/2^3$ items were updated. This pattern is continued until the lowest bottom sequence in the structure is reached. This sequence will contain only 2 bits; these correspond to the two "1" bits in the preceding sequence. Of the two bits in the lowest sequence, one can be set to "1" to indicate the last item that was changed recently. The timestamp of the sequence indicates the time after which the item was updated.

For simplicity, we assume that the number of data items, N, is the n power of 2; i.e., $N = 2^n$ for some integer n. Let B_n denote the highest sequence, B_{n-1} the next sequence, ..., and B_1 denote the lowest sequence, where $n = log(N)$. The timestamp of bit sequence B_k is represented by $TS(B_k)$.

The total number of bits in B_k is denoted by $\mid B_k \mid$ and the total number of "1" bits in B_k by ΣB_k.

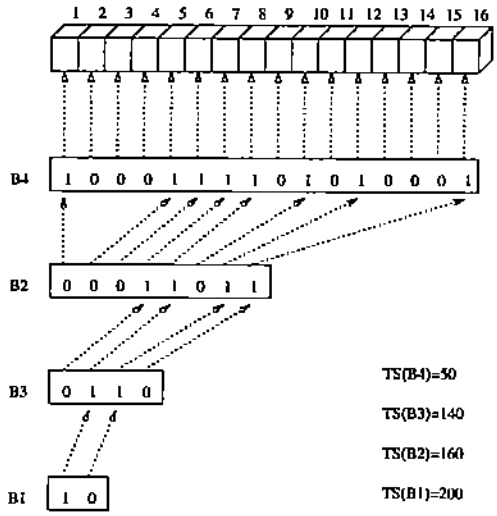

Figure 1: A Bit-Sequences Example

Clients must check the invalidation report before they can use their caches for query processing. If there is a bit sequence among the sequence set with the most recent timestamp that is equal to or predates the disconnection time of the client, the sequence will be used to invalidate its caches. The data items represented by these "1" bits in the sequence will be invalidated. If there is no such sequence (i.e., the disconnection time precedes the timestamp of the highest sequence), the entire cache in the client will be invalidated.

Example 3.1 *Consider a database consisting of 16 data items. Figure 1 shows a Bit-Sequences (BS) structure reported by a server at time 250. Suppose that a client listens to the report after having slept for 80 time units. That is, the client disconnected at time 170 (=250-80), which is larger than $TS(B_2)$ but less than $TS(B_1)$. The client will use B_2 to invalidate its caches. To locate those items denoted by the two "1" bits in B_2, the client will check both B_3 and B_4 sequences, using the following procedure. To locate the second bit that is set to "1" in B_2, check the position of the second "1" bit in B_3. We see that the second "1" bit in B_3 is in the 5th position; therefore, check the position of the 5th "1" bit in B_4. Because B_4 is the highest sequence and the 5th "1" bit in B_4 is in the 8th position, the client concludes that the 8th data item was updated since time 170. Similarly, the client can deduce that the 12th data item has also been updated since that time. Therefore, both the 8th and 12th data items will be invalidated.*

Server Bit-Sequences Construction Algorithm: For each invalidation report, the server will construct a Bit-Sequences (BS) structure based on the update timestamps of data items. To construct the structure, the server should know the data items that are updated and the timestamps for these updates. An *update linked list* can be used to maintain the information about the update timestamps.

Each data item can be denoted by a node in the list. The node should include the following fields: (a) the index number of the data item (note that data items are numbered consecutively); (b) the update timestamp; (c) the pointer to the next node; and (d) the 1-bit position of the data item in a bit sequence.

All the nodes are linked by the pointer fields in the decreased order of update timestamps. That is, the first node in the update linked list denotes the data item that is most recently updated; the second node denotes that data item that is next recently updated; and so on. When a data item is updated, the node denoting the item is moved to the head of the update linked list. To quickly locate the node in the list for a data item, an additional index, called the *item-node index*, that maps a data item to its node in the update linked list can be used. Using the update linked list and the item-node index, the server constructs the Bit-Sequences structure by the following procedure (initially, all bits in B_k are reset (i.e., "0") and $TS(B_k) = 0$ for all k, $0 \leq k \leq n$):

1. If the update timestamp of the 1th node is larger than zero, then construct B_n:

   A. while ($i \leq N/2$ and the update timestamp of the ith node is larger than zero) do:
   /* initially, $i = 1$ */

      set the jth bit in B_n to "1" where j is the index number of the ith node; $i = i + 1$.

   B. assign the update timestamp of the ith node to $TS(B_n)$. /*when $i < N/2$,$TS(B_n) = 0$*/

   C. for $i = 1$ to N do:
   /* update the 1-bit position of node in the update linked list; initially, $j = 1$ */

      if the ith bit (i.e., the ith data item) is set to "1" in B_n, then (a) locate the node for the ith data item in the update linked list using the item-node index; (b) set the value "j" into the 1-bit position of the node; $j = j + 1$.

2. If $\Sigma B_{k+1} \geq 2$, then construct B_k for all k ($0 \leq k \leq n - 1$):

   A. while ($i \leq \Sigma B_{k+1}/2$) do: /* initially, $i = 1$ */

      set the jth bit in B_k to "1" where j is the 1-bit position of the ith node; $i = i + 1$.

   B. assign the update timestamp of the ith node to $TS(B_k)$.

   C. for $i = 1$ to ΣB_k do:
   /* update the 1-bit position of node in the update linked list; initially, $j = 1$ */

      if the ith bit (i.e., the ith data item) is set to "1" in B_k, then (a) locate the node for the ith data item in the update linked list using the item-node index; (b) set the value "j" into the 1-bit position of the node; $j = j + 1$. □

Note that, in the above algorithm, we use a dummy bit sequence B_0. The size and "1" bit number of the sequence are always equal to zero. However, the server will include the timestamp of the sequence $TS(B_0)$ into each invalidation report. The timestamp indicates the time after which no data item has been updated.

Client Cache Invalidation Algorithm: Before a client can use its caches to answer the queries, the client shall wait for the next invalidation report that includes the Bit-Sequences structure, and

then execute the following procedure to validate its caches. The input for the algorithm is the time variable T_l that indicates the last time when the client received a report and invalidated its caches.

1. if $TS(B_0) \leq T_l$, then no cache is to be invalidated and stop;
2. if $T_l < TS(B_n)$, then the entire cache is invalidated and stop;
3. Locate the bit sequence B_j with the most recent timestamp that is equal to or predates the disconnection time T_l, i.e., B_j such that $TS(B_j) \leq T_l$ but $T_l < TS(B_{j-1})$ for all j $(1 \leq j \leq n)$;
4. Invalidate all the data items represented by the "1" bits in B_j. To determine the index numbers of these items (i.e., the position of the bit that denotes the data item in B_n), the following algorithm can be used:
   A. mark all the "1" bits in B_j;
   B. if $j = n$, then all the data items that are marked in B_n are to be invalidated and the positions of these "1" bits in B_n are their index number in the database and stop;
   C. for each "1" bit in B_j, mark the ith "1" bit in B_{j+1} if the "1" bit is in the ith position in B_j;
   D. $j = j + 1$ and go back to step B. □

3.3 Discussion

As we have seen in the previous subsection, a client in the BS algorithm may invalidate its entire cache only if its disconnection period started prior to the time specified by the timestamp of the highest bit sequence B_n. In this case, there would be at least N/2 data items that have been updated at the server during the client disconnection period. When the client disconnection period is less than $TS(B_0)$, there are no data items which need be invalidated.

A client will use a bit sequence, say B_k $(1 \leq k \leq n)$, to invalidate its caches if it started its disconnection at a time which is equal to or larger than $TS(B_k)$ but smaller than $TS(B_{k-1})$. By the properties of Bit-Sequences, we know that as many as ΣB_k data items that are indicated by "1" bits in B_k may need to be invalidated. Among the ΣB_k data items, there are at least ΣB_{k-1} data items that have been updated at the server since the client's disconnection (where $\Sigma B_k/2 = \Sigma B_{k-1}$). Therefore, in the worst case, there are at most $\Sigma B_k/2$ data items in the client caches to be falsely invalidated. The real number of falsely invalidated data items will actually depend on several factors such as the last time the invalidated data items were cached, and the query/update patterns etc.

To see how the caching time or query/update pattern impact the false invalidation, assume that the client started the disconnection at time T_l where $TS(B_k) < T_l < TS(B_{k-1})$ (see Figure 2). The worst case in which $\Sigma B_k/2$ data items are to be falsely invalidated can happen if and only if (1) the $\Sigma B_k/2$ data items were updated from $TS(B_k)$ to T_l *and* (2) the client cached these items after the updates (and before its disconnection) at time T_c. Scenario 1 in Figure 2 shows the case. On the other hand, if the client cached these data items at the time T_c that is before $TS(B_k)$, then these invalidated data items are actually obsolete ones and no data item is falsely invalidated. Scenario

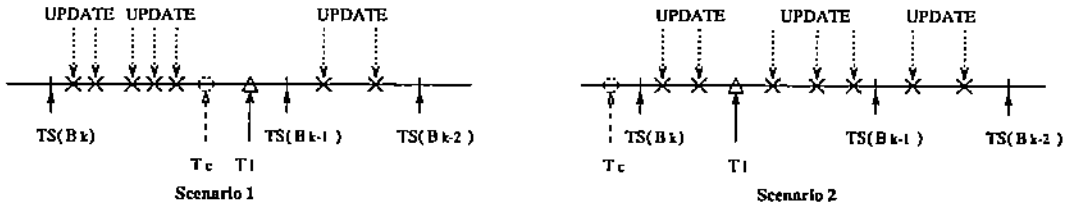

Figure 2: Client Query/Disconnection Scenarios

2 in Figure 2 gives the case. Furthermore, even the client cached some of these data items between $TS(B_k)$ and T_l, these cached items will not be falsely invalidated if they were updated between T_l and $TS(B_{k-1})$.

In contrast, in the AT or TS algorithms, the entire cache will be invalidated if the disconnection time exceeds an algorithm-specified value (w seconds in TS and L seconds in AT), regardless of how many data items have actually been updated during the disconnection period. The actual number of updated data items may be very small if the update rate is not high (the actual number can be approximated by λ_uT, where λ_u is the update rate and T is the disconnection time). In the SIG algorithm, most of the cache will usually be invalidated when the number of data items that were updated at the server during the disconnection time exceeds f. Thus, the false invalidation ratio in the AT, TS, or SIG algorithms will depend heavily on such workload parameters as the update rate and disconnection time.

Let us now consider the report size in each algorithm. In the BS algorithm, the size of the invalidation report can be approximately expressed as a function of the number of data items: 2N + $b_T$log(N), where N is the total number of data items in the database and b_T is the size of a timestamp. By the definition, we know that, for the N data items, there are log(N) bit sequences with respective sizes (in bits) of N, N/2, N/2^2, ..., 2. The total bits for the set of sequences will be no more than 2N. Because each sequence is associated with a timestamp, the total bits for these timestamps will be $b_T$log(N).

In comparison, the report size in the SIG algorithm can be expressed, as in [7], as $6g(f + 1)(ln(1/\delta)+ln(N))$, where g is a parameter used to specify the bound 2^g of the probability of failing to diagnose an invalid cache, δ is a parameter used to specify the bound of the probability of diagnosing valid cache as invalid, and f is the number of different items that the SIG algorithm can diagnose. The report sizes for the TS and AT algorithms can be expressed, as in [7], as $n_w(log(N) + b_T)$ and $n_L log(N)$, respectively, where n_w is the number of data items updated within w seconds and n_L is the number of data items updated within L seconds. As an example, if we set N = 1024, $g = 16$, $\delta = 10^{-7}$, $f = 10$, and $b_T = 512$, the report sizes in the BS, AT, TS, and SIG algorithms will be around 7200 bits, $10n_L$ bits, $522n_w$ bits, and 24000 bits, respectively. That is, the report size in BS is smaller than that in SIG but larger than that in AT and TS.

In summary, the Bit-Sequences (BS) algorithm has the following the key features that distinguish it from the AT, TS, SIG algorithms:

- Changing workload parameters, such as client disconnection time and update rate/pattern, are expected to have less impact on the ratio of false invalidations than with the AT, TS, or SIG algorithm.

- The size of bit sequences is independent of the number of updated items.

4 The Scalable Bit-Sequences Algorithm

In the basic BS algorithm, each bit is used to represent a data item in the database. The invalidation report size, which is represented by $2N + b_T log(N)$, is approximately lineally proportional to the number of items in the database. To limit the size of invalidation report for large databases, at the simplest level, we can uniformly increase the granularity of each bit in the invalidation report instead of increasing the number of bits. For example, for an 8GB database, each bit in the report can represent an 8MB data block in the database. Both clients and servers can still use 8K page size as their access granularity. A bit in the report will be set to "1" if and only if any page in the block represented by the bit has been updated. A client will invalidate a cached data page if the block that contains the page has been marked by "1" bit in a corresponding bit sequence. We call this scaled algorithm as the *Coarse Granularity BS* (CG-BS) algorithm.

A second method is to group different data pages into one data block according to the relative frequency of update. For example, rarely updated data pages can be grouped in one data block and each frequently updated data page itself constitutes one data block. Each data block is represented by one bit in the invalidation report. The *Weighted Granularity BS* (WG-BS) scheme would scale well to a large database if the update pattern in the server is relatively "static".

Intuitively, we believe that both scaled BS algorithms described above still perform better than the SIG algorithm for a client if more than f, say m, data pages have been updated during the disconnection period of the client. The reason is that most caches in the client will be invalidated in the SIG algorithm if m ($> f$) data pages (it does not matter whether the m pages are cached or not) have been updated in the server. On the other hand, for the scaled BS algorithms, if these m pages happen to be within one block, the client needs only to invalidate all the cached pages that are in the same block. Cached data pages from other blocks will not have to be invalidated. Generally, the false invalidation rate depends on which blocks the updated data pages and cached data pages reside at. Similarly, both scaled BS algorithms has a low false invalidation rate compared to the AT or TS algorithm for long disconnected clients.

However, both the CG-BS and WG-BS methods have some disadvantages for some application domains like those in "information feed" environments. [3] In an "information feed" application, the

[3] In an "information feed" environment, although some information such stock prices during trading hours may change constantly, the cache approach is still very useful if a small percent deviation of the true values in the clients may be acceptable. In this situation, the "quasi-copy" scheme proposed in [3] can be applied. The server can send invalidation information only for the data whose changes have exceeded the threshold specified by the "quasi-copy"

data server produces (or updates) data that would be consumed by other clients. Both updates and queries have the same "hot spot" pages (or the shared locality) in a specific time period. Furthermore, the "hot spot" pages may dynamically be changed. For example, traffic data pages will be the "hot spots" (updated by servers and consumed by clients frequently) only in the traffic rush hour, while the pages for some stock prices would become "hot" only during the heavy trading period of these stocks. In the CG-BS algorithm, different stock prices may be grouped in one data block. If only a few hot stocks are heavily being traded, the clients that have cached many slowly changed stocks may be falsely trigged to establish an uplink connection for cache verification. The WG-BS algorithm may not adapt well to this dynamically changed "hot spots" of updates and queries.

4.1 The Multi-Level Bit-Sequences Algorithm

We now describe a Multi-Level Bit-Sequences (ML-BS) algorithm that is expected to adapt well to an "information feed" environment with dynamically changed and skewed "hot spots" of updates and queries. In the algorithm, the database is abstractly viewed as a hierarchical data structure. Suppose that the database has N^{p+1} data pages ($p \geq 0$). In the highest level, the database is equally divided into N data *p-blocks*. In the next level, each data p-block is equally divided into N data *(p-1)-blocks*. This pattern is continued until the lowest level is reached. In the lowest level, each data 1-block is divided into N data 0-*blocks* (i.e., data pages). It is assumed that a data page is the basic access unit for updates and queries in both the server and client sides.

An invalidation report consists of one 0-level BS (0L-BS) structure, one 1-level BS (1L-BS) structure, ... , and one p-level BS (pL-BS) structure. Each kL-BS structure ($0 \leq k \leq p$), which is the exact same as that described in Section 3 for the BS algorithm except the granularity of each bit, consists of $log(N)$ bit sequences with an associated set of timestamps. The highest-ranking sequence in each kL-BS structure has N bits, while the lowest-ranking sequence has 2 bits. In the 0L-BS structure, each bit represents one data page (i.e., one data 0-block) in a data 1-block. Each bit in the kL-BS structure represents one data k-block in a data (k+1)-block ($0 \leq k \leq p$ and the (p+1)-block is the database). The bit is set to "1" only if any data page in the data k-block has been updated since the time specified by the sequence timestamp.

In each invalidation report, the 0L-BS structure is always used to represent the data pages in the "hot" data 1-block. The 1L-BS structure is used to represent the data 1-blocks in the "hot" data 2-block, and so on. The absolute starting address (or name) of the data k-block represented by each kL-BS structure is also included in the invalidation report. The clients will use the address to figure out whether a cached page belongs to the data k-block.

Example 4.1 *Suppose that the database with* 8^3 *data pages is organized in a three-level granularity data structure (see Figure 3). In the three-level BS algorithm, an invalidation report will include*

scheme. For example, the server indicates the "update" of a stock price in the invalidation report only if the change of the stock price is more than 0.5 percent of its last broadcast value.

one 0L-BS, one 1L-BS and one 2L-BS structures. The highest-ranking bit sequence in each structure has 8 bits, while the lowest-ranking bit sequence has only 2 bits. In the 0L-BS structure, each bit represents a data page (i.e., a data 0-block) in a block (i.e., a data 1-block) of 8 pages and set to "1" only if the page has been updated since the time specified by the sequence timestamp. In the 1L-BS structure, a bit that represents a data 1-block in a data 2-block of 8 data 1-blocks is set to '1' only if any page in the 1-block has been updated since the time specified by the sequence. In the 2L-BS structure, a bit that represents a data 2-block is set to '1' only if any page in the 2-block has been updated since the time specified by the sequence timestamp.

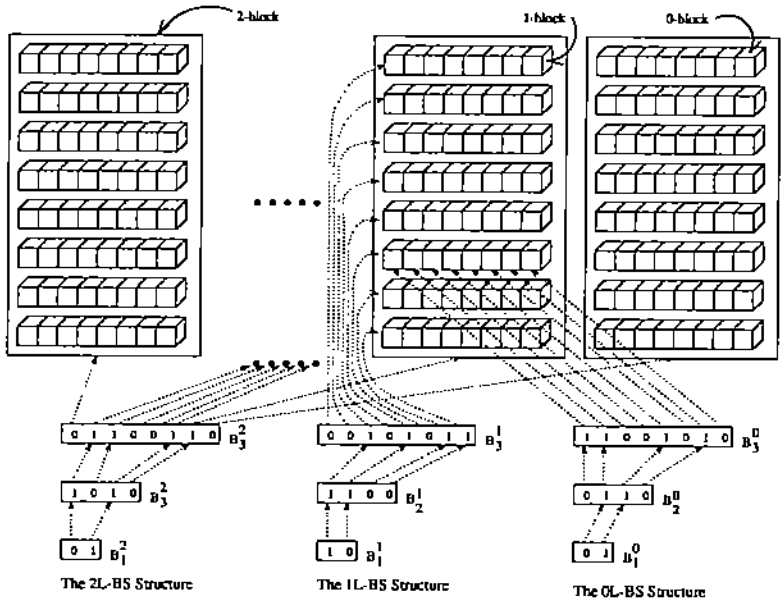

Figure 3: A Multi-Level Bit-Sequences Example

Server Multi-Level Bit-Sequences Construction Algorithm Sketch: For each invalidation report, the server will construct a set of multi-level BS structures by the following procedure:

1. *Determine the "hot" data $(k+1)$-block $(0 \leq k \leq p)$:* The "hot" data $(k+1)$-block in which data pages are frequently queried by clients can be dynamically changed time to time. For some "information feed" applications, the server can simply use application semantics (e.g., the traffic report in the rush hour, the weather report in the severe-weather watch period, or the stock quotes in the trading hours) or the rate and pattern of updates in the recent time period to determine the data $(k+1)$-block because both updates and queries share the same "hot spots". In general, however, the server may use the feedback information from clients to adaptively decide the data $(k+1)$-block (the adaptive algorithms similar to those in [6, 13] can be used for the purpose). The absolute address of the "hot" data $(k+1)$-block will be included in the next invalidation report.

2. *Construct the update linked list for each "hot" data $(k+1)$-block $(0 \leq k \leq p)$:* Because the kL-BS structure may be used to represent a data $(k+1)$-block dynamically, the update linked

list should be built according to the update timestamps in the corresponding data pages of the $(k+1)$-blocks. Each node in the update linked list denotes a data k-block in the "hot" data $(k+1)$-block. The update timestamp in the node is equal to that of the data page that is most recently updated in the $(k+1)$-block. All the nodes in the update linked list are linked by the pointer fields in the decreased order of update timestamps.

3. *Construct the kL-BS structure ($0 \leq k \leq p$)*: The kL-BS structure is constructed using the update linked list of the "hot" data $(k+1)$-block. The construction procedure is that same as that of the basic BS structure except that each bit in the kL-BS structure represents a data k-block. □

Client Cache Invalidation Algorithm Sketch: After receiving an invalidation report, the client will use the following procedure to validate each cached page:

1. *Locate the kL-BS structure*: determine the smallest k such that the data $(k+1)$-block represented by the kL-BS structure in the invalidation report contains the data page (the address of the data k-block has been included in the report).

2. *Validate the cached page*: use the cache invalidation procedure similar to that in the basic BS algorithm to validate the cached page. The input to the procedure includes the kL-BS structure determined in the Step 1 and the parameter T_l which is the last time when the cache was validated or cached. The cached data page is to be invalidated if the data k-block that contains the page has been marked by "1" bit in a corresponding bit sequence. □

4.2 Optimization for Cold Page Invalidation

In the ML-BS algorithm described above (and the basic BS algorithm as well), a client will validate all the cached pages when it receives an invalidation report. Suppose that the cached pages, which are not denoted by the bits in the page-level 0L-BS structure, are in a k-block denoted by a bit in the kL-BS structure. If any page in the block has been updated since the last cache validation time, then these pages will be invalidated although most of them are actually valid. To reduce the false invalidation chance, the following optimization can be applied:

Broadcast Cache-Freshing Bit-Sequences: In each invalidation report, the server includes an additional Bit-Sequence structure called *Cache-Freshing Bit-Sequences* (CF-BS). The bit granularity in the structure is at the data page level (i.e., each bit represents a data page in a data 1-block and the structure represents the data 1-block). The server changes the 1-block represented by the CF-BS structure periodically. That is, the structure will represent the $(j+1)$th (or the 1st) data 1-block in the database if it represented the jth (or the last) data 1-block in the last invalidation report (assuming all the data 1-blocks in the database are numbered).

Deferred Cache Invalidation for Uncertain Cold Pages: When a client receives an invalidation report, it uses the following rules to decide whether a cached copy is to be validated or not.

1. Always validate the cached pages in the 1-blocks represented by the fine-granularity CF-BS and 0L-BS structures;

2. Always validate the cashed page in the data k-block ($1 \leq k \leq p$) denoted by a bit in the kL-BS structure if the data k-block has not been updated since the last validation of the cached page;

3. Defer to invalidate any other cached page until it is used if the data k-block ($1 \leq k \leq p$) that contains the page has been updated since the last validation of the cached page.

Because each cached copy will not be always validated when the client receives an invalidation report, the client should maintain a validation timestamp for each cached copy. When the cache invalidation algorithm that uses a kL-BS ($0 \leq k \leq p$) or a CF-BS structure is executed, the the input parameter T_l to the algorithm will be the last time when the page was validated or cached. [4] The rule 1 will actually produce a very low false invalidation ratio because of the use of the fine-granularity (i.e., page-level) BS structure, while the rule 2 does not invalidate any cached page (though it does validate the valid cached pages). The cached pages validated in both rules 1 and 2 will receive a latest validation timestamp. The cached pages that have a later validation timestamp will have a lower chance of being falsely invalidated next time because a low ranking bit sequence can be used for a cached copy that has a late invalidation timestamp.

In the rule 3, however, the client will defer to invalidate any other cached page until it is used if the page is to be invalidated by a coarse-granularity kL-BS structure. The delay of invalidation will help to reduce the false invalidation chance for the "cold" cached page. In fact, the "cold" cached page may be less frequently updated or queried than "hot" cached pages. Before the query that uses the the "cold" cached page arrives, there might be many queries executed for "hot" cached pages (therefore, there are many invalidation reports received and cache validations completed). If the client happened to receive a report that contains a CF-BS structure representing the "cold" page's 1-block before the query is executed, then the "cold" page might be validated as a valid copy by the fine-granularity CF-BS structure and receive a later validation timestamp. When the query to the "cold" cached page is executed, the client can use the cache provided that no other pages in the k-block have been updated after the client received the CF-BS structure.

4.3 Discussion

A single coarse-granularity BS structure can limit the size of an invalidation report for a large database. It may, however, incur a high false invalidation chance. The ML-BS method provides a flexible scheme that allows the system to use multiple BS structures to tune the granularity of data blocks for invalidation. The basic idea behind the ML-BS method is to use less bits to represent the data pages that are rarely changed in the database. If each of these data pages is represented by one bit, in most of time these bits either are in "0" status or appear in the top-ranking bit sequences with very small timestamps. In a coarse-granularity BS structure, these pages are actually grouped

[4] In the cache invalidation algorithms described previously, the client will validate all the cached pages when it receives an invalidation report. These algorithms, therefore, only maintain the validation timestamp T_l for the whole cache rather than for each individual cached page.

together and represented by a few bits. The use of the CF-BS structure and the deferred validation for uncertain cold caches will help to reduce the false invalidation chance for these "cold" data pages.

For the "hot" data pages, on the other hand, a fine-granularity (i.e., page-level) BS structure can be used for the validation of cached copies. The server can simply determine these "hot" data pages according to application semantics or the update rate and pattern (in the "information feed" application domain, both the servers and clients will have a shared locality in most of time). The ML-BS method is expected to perform well without the use of large invalidation report for the "information feed" applications with a skewed access pattern .

In some situations, the "hot" data pages may be separated in different data 1-blocks. For example, both the traffic rush hour and the severe-weather watch may happen at the same time, but the data pages for the traffic report and the weather watch reside in two different data blocks. In this case, the first half of bits in the page-level BS structure can be used to represent the traffic data pages, while the second half of bits are used to denote the weather watch data pages. However, the server should include two absolute starting addresses in the invalidation report, one for the traffic pages and another for the weather watch pages. The client can figure out the data page represented by each bit according to the absolute starting address and the position of the bit.

The the ML-BS algorithm can scale to a large database in terms of the size of invalidation report. Suppose that we have an N^{p+1} page database and the highest-ranking bit sequence in each kL BS structure is N bits. Then the size of each invalidation report in the ML-BS algorithm will approximately be $(p+1)S(N)$ where $S(N)$ $(= 2N + b_T log(N))$ is the size of a basic BS structure with N bits in the highest-ranking bit sequence. In comparison, if the basic BS algorithm is used (i.e., one bit for one page), the size is close to $S(N)^{p+1}$ for $N >> 1$.

5 Simulation Study

The performance analysis presented here is designed to compare the effects of different workload parameters, such as disconnection time and query/update pattern, on the relative performance of the basic BS algorithm and the SIG algorithm proposed in [7]. The performance metrics in the study include the false invalidation ratio and the buffer hit ratio in clients. We do not include the ML-BS algorithm in our performance study in this paper because we expect that the ML-BS algorithm can be comparable to the basic BS algorithm in terms of the false invalidation ratio and the buffer hit ratio provided that the server can correctly decide the "hot spots" for the applications with a skewed access pattern. Also, the study does not include the AT and TS algorithms because they are not applicable to long disconnections of clients.

Our model is similar to that employed in [9, 10] but has been extended to support message broadcasting over wireless channels. Our model also simplifies aspects of resource management in both client and server so that no CPU and I/O times are modeled in each. Such a simplification is appropriate to an assessment of the false invalidation ratio and the buffer hit ratio. All simulations

were performed on Sun Sparc Workstations running SunOS and using a CSIM simulation package [16].

5.1 System Configuration

For simplicity, we model a single server system that services multiple mobile clients. At the server, a single stream of updates is generated. These updates are separated by an exponentially distributed update interarrival time. The server will broadcast an invalidation report periodically.

Each mobile client host generates a single stream of queries. The arrival of a new query is separated from the completion of the previous query by either an exponentially distributed think time or an exponentially distributed disconnection time. Our model assumes that each client will enter into a disconnection mode once for every three consecutive queries. In other words, each group of three queries will be separated by a disconnection time, and these three queries are separated from each other by a think time.

We assume that the buffer pool in the server is large enough to hold the entire database. The size of client buffer pools is specified as a percentage of the database size. The buffer pools are managed using an LRU replacement policy.

The network is modeled as a PRE_RES server (i.e., the service discipline is preempt resume, based on priority) with a service rate of *Network Bandwidth*. Invalidation report broadcasts have higher priority than other messages, ensuring that these reports can always be broadcast over the wireless channels with an exact broadcast period specified by the parameter *Broadcast Period*. All other messages are of equal priority and will be served on a first-come first-served basis.

Our model can specify the item access patterns of workloads, thus allowing different client locality types and different server update patterns to be easily specified. For each client, two (possibly overlapping) database regions can be specified. These regions are specified by the *HotQueryBounds* and *ColdQueryBounds* parameters. The *HotQueryProb* parameter specifies the probability that a query will address a data item in the client's "hot" database region. Within each region, data items are chosen through a uniform distribution. For the data server, the *HotUpdateBounds* and *ColdUpdateBounds* parameters are used to specify the "hot" and "cold" regions, respectively, for update requests. The *HotUpdateProb* parameter specifies the probability that an update will address a data item in the update's "hot" database region.

Tables 1 presents the database simulation parameters and settings employed in our simulation experiments. Table 2 describes the range of workloads associated with access patterns considered in this study. These workloads are very similar to those described under the same names in [10]. The UNIFORM workload is a low-locality workload in which caching is not expected to reap significant benefits. The HOTCOLD workload has a high degree of locality of client queries. The FEED workload is intended to model an "information feed" situation where the data server produces data to be consumed by all other clients. This situation is of interest because information services are

Parameter	Setting
Data Item Size	8192 bytes
Database Size	1000 data items
Client Buffer Size	25% of database size
Number of Clients	100 mobile client hosts
Broadcast Period	10 seconds
Network Bandwidth	10000 bits per second
Control Message Size	512 bytes
Mean Think Time	100 seconds
Mean Disconnect Time	200 to 10000 seconds
Mean Update Arrive Time	100 to 1000 seconds

Table 1: Simulation Parameter Settings

Parameter	UNIFORM	HOTCOLD	FEED
HotQueryBounds	-	i to i+9,i=10(k-1)+9, the kth client	1 to 200
ColdQueryBounds	all DB	remainder of DB	remainder of DB
HotQueryProb	-	0.8	0.8
HotUpdateBounds	-	-	1 to 200
ColdUpdateBounds	all DB	all DB	remainder of DB
HotUpdateProb	-	-	0.8

Table 2: Workload Parameter Settings

likely to be one of the typical applications in mobile computing environments [12].

To simulate the SIG algorithm, additional algorithm-specified parameters are needed. The setting for the invalidation report size (*SigReportSize*) in SIG is 24000(bits) which is computed by the formula $6g(f+1)(ln(1/\delta)+ln(N))$ which is given in [7], with N = 1000, $g = 16$, $\delta = 10^{-7}$, and $f = 10$ (i.e., the number of differing items that can be diagnosed by SIG). In our experiments, the signatures in the SIG invalidation report are simulated as well, since we have not used actual data items in these simulations. Each signature is simply represented by an integer variable. The value of the variable will be increased by one whenever one data item in the set represented by the signature is updated. In this way, for a signature S_i, the broadcast copy will be different from the cached copy if any data item in the signature set was updated between the last caching time and the current broadcasting time. The simulated signatures have been idealized to forestall any possibility of incorrectly diagnosing an outdated cache as valid. In real signatures, the probability of incorrectly diagnosing a cache as valid is bounded by the parameter 2^g [7].

The false invalidation ratio is computed by dividing the sum of the false invalidated items by the sum of the total invalidated items in the simulation. The total invalidated data items in a caching algorithm include those data items which are both falsely and legitimately invalidated. The buffer hit ratio is defined as the ratio of the number of queries that are answered using client caches to the

total number of queries.

5.2 Experiment 1: The UNIFORM Workload

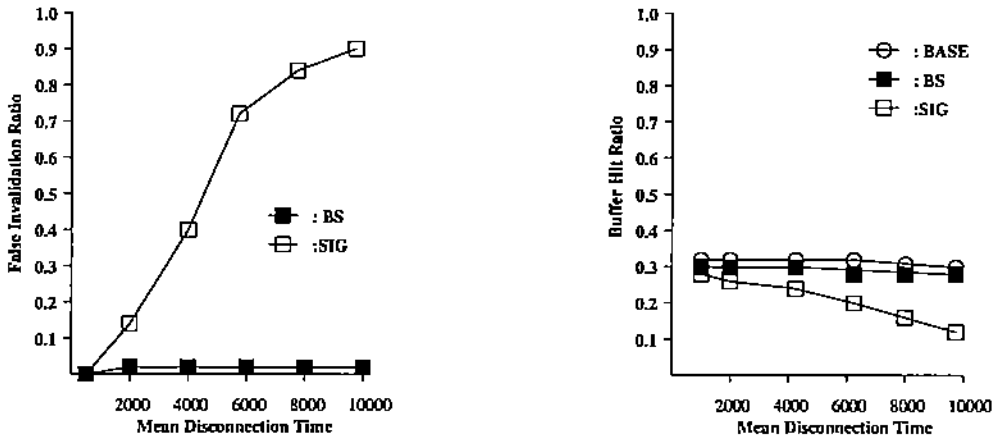

Figure 4: UNIFORM Workload, Update Rate = 0.001

In the UNIFORM workload, as indicated in Table 2, each query reads a data item chosen uniformly from among all the data items in the database. Also, each update request chooses a data item uniformly from all the data items. In this workload, both query and update parameters display no locality of access.

Figure 4 shows the experimental results achieved with the UNIFORM access workload and a mean update interarrival time of 1000 seconds. The false invalidation ratio of the SIG algorithm (with f = 10) increases rapidly when the disconnection time increases. In contrast, the false invalidation ratio of the BS algorithm is very low (less than 0.05 for all the disconnection times from 200 to 10000 seconds). It is observed that the false invalidation ratio of the SIG algorithm increases fast although the multiplication value of the mean disconnection time ($\leq$ 10000) and the mean update rate (= 0.001) in Figure 4 is no larger than the SIG parameter f (= 10). This behavior can be explain as follows. In our experiments, both disconnection time and update interarrival time are exponentially distributed. For a particular client, the disconnection time may exceed the mean disconnection time and the number of updated data items during the disconnection period may also be larger than the mean number of updated data items. Once this happens, a large number of data items will be falsely invalidated (the reason is discussed in Appendix A). This number may be much larger than the actual number of updated data items will significantly increase the false invalidation ratio (Notice that the ratio is computed by taking the sum of the false invalidated items and the sum of the total invalidated items in the simulation, and dividing the first sum by the second one). If we assume the disconnection time and the update rate are constant distributions, the false invalidation ratio in Figure 4 should be very close to 0 because the multiplication value of the disconnection time (= 0 to 10000) and the update rate (= 0.001) for each client should be no larger than the parameter f

(= 10) of SIG.

Figure 4 also shows the buffer hit ratio of an "idealized" cache invalidation algorithm called BASE. The "idealized" algorithm always invalidates stale cached data and never falsely invalidate any valid cached data. It is shown that the BS algorithm is almost as good as the BASE algorithm in terms of buffer hit ratio.

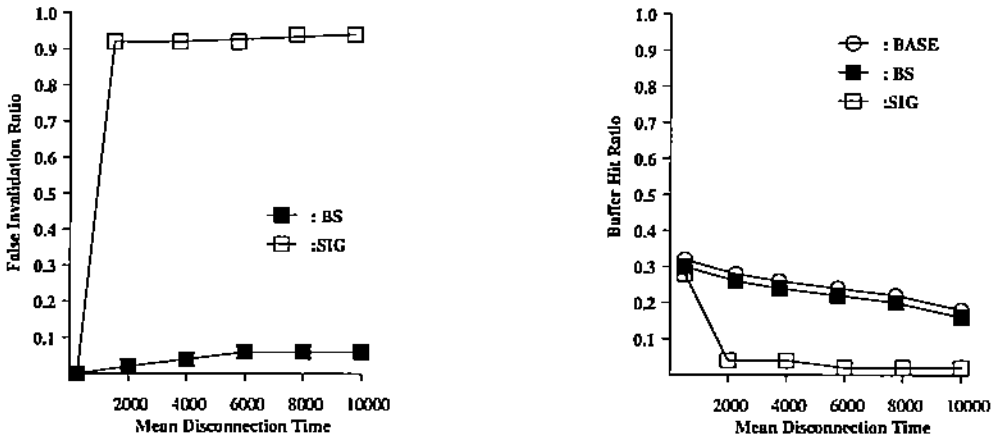

Figure 5: UNIFORM Workload, Update Rate = 0.01

In this experiment, we also examine the effect of high update rate on the false invalidation ratio by setting the mean update interarrival time to 100 seconds. Figure 5 shows the experimental results. The false invalidation ratio of the BS algorithm is still very low even under the condition of high update rate, while the false invalidation ratio of the SIG algorithm reaches 0.9 rapidly at the disconnection time 2000. Due to the high false invalidation rate, the buffer hit ratios are very low in SIG. The experimental result indicates that the BS algorithm can adapt to changing update rates much better in terms of low false invalidation ratio than the SIG algorithm.

5.3 Experiment 2: The HOTCOLD Workload

In the HOTCOLD workload, each update request chooses a data item uniformly from all data items. Each client has a 10-item hot query region to which 80% of its requests are directed; the other 20% of its requests are directed to elsewhere in the database. Thus, this workload represents a situation in which client queries favor disjoint local hot regions of the database and significant benefits can be expected from caching.

Figure 6 depicts the experimental results results achieved with the HOTCOLD workload and a mean update interarrival time of 1000 seconds. A comparison with Figure 4 and Figure 6 indicates that the buffer hit ratios of the BS and SIG algorithms in the HOTCOLD experiment are obviously higher than those in the UNIFORM experiment. The increases of the hit ratios are due to the high locality of the query access in the HOTCOLD workload.

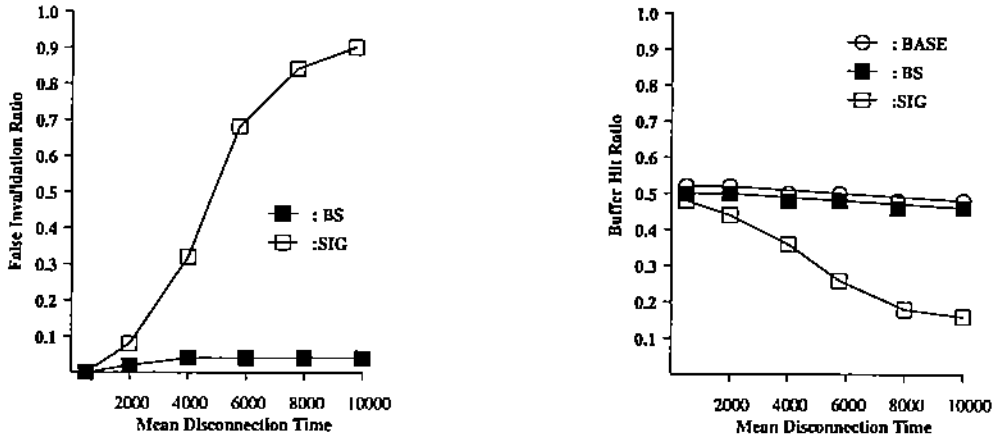

Figure 6: HOTCOLD Workload, Update Rate = 0.001

The buffer hit ratio of the SIG algorithm in Figure 6 drops faster than that of the BS algorithm when the disconnection time increases, Because of the differences of false invalidation ratios.

5.4 Experiment 3: The FEED Workload

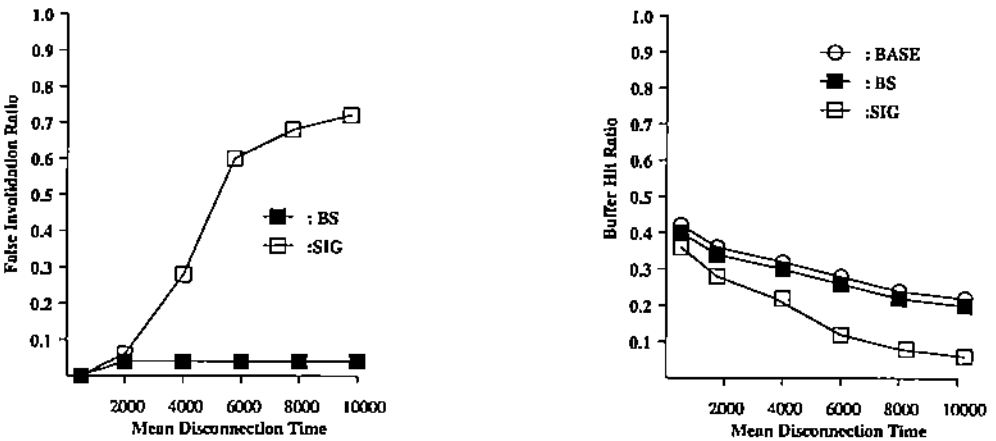

Figure 7: FEED Workload, Update Rate = 0.001

In an "information feed" workload, the data server produces data to be consumed by other clients. This is intended to approximate an environment, such as stock trading, where a database of stock prices might be maintained by an information feed and then accessed heavily by other clients. In this workload, 80% of the server updates are directed to 20% of the database; 80% of the client queries are also directed to this region.

Figure 7 indicates the experimental results achieved with the FEED workload and a mean update interarrival time of 1000 seconds. Once again, the false invalidation ratio of the BS algorithm remains

very low even when the disconnection time increases. Comparing Figure 4 and 7, we note that the increasing rate of the false invalidation ratio of the SIG algorithm in this experiment is lower than that in the UNIFORM experiment. This is because in this experiment both updates and queries are directed to the same "hot" region. The frequently cached items (i.e. queried items) are also the frequently updated data items. So, the false invalidation ratio of the SIG algorithm decreases in this workload. Consequently, the hit ratio of the SIG algorithm in this experiment is higher than that in the UNIFORM experiment.

5.5 Summary

The experiments presented in Sections 5.2 through 5.4 showed the false invalidation ratio and the buffer hit ratio of the SIG, and BS algorithms under different disconnection times, update rates, and access workloads. In general, changing these workload parameters have little impact on the false invalidation ratio of the BS algorithm. The buffer hit ratio of BS is very close to that of the "idealized" cache invalidation algorithm (which has no false invalidation to cached data). In the environment of a high locality of queries, the false invalidation ratio will have a significant impact on the buffer hit ratio.

Although all the experimental results were given under 3 hours (10000 seconds) disconnection times, The mean false invalidation rates for the BS algorithm are expected not very high even when the disconnection times are longer than 3 hours. A client will invalidate its entire caches only if more than half number of data items have been updated in the server since its last validation time. We only presented the experimental results in the range of 3 hours disconnection times because this range provides a clear and useful comparison between the SIG and BS algorithms.

In this paper, we have not discussed the performance of these algorithms for a situation in which clients are often connected. This aspect has not been addressed as in this paper we are more interested in the false invalidation aspect of these algorithms and its effect on their performance. The false invalidation problem becomes serious only when these algorithms are placed under conditions of long client disconnection times and changing update and query rates. When clients are often connected, both TS and AT algorithms are expected to perform well, because of the relatively small size of invalidation reports.

6 Conclusions

In this paper, we have introduced a new cache invalidation algorithm, called Bit-Sequences (BS), in which a periodically broadcast invalidation report is organized as a set of binary bit sequences with an associated set of timestamps.

We have shown that the BS algorithm adapts itself dynamically as the update rate/pattern varies because the bit sequence B_k $(1 \leq k \leq n)$ in the invalidation report will always carry the information

about the $N/2^{n-(k-1)}$ data items that have been recently updated. Clients using the bit sequence B_k to invalidate their caches will be guaranteed that, among the data items that are marked in ΣB_k for invalidation, half ($\Sigma B_k/2$) have actually been updated at the data server. A client that invalidates its entire cache is guaranteed that $\Sigma B_n/2$ data items have been updated. This adaptability enables the BS algorithm to reduce the false invalidation ratio at client sites.

We have also described a scalable BS algorithm, called the Multi-Level BS (ML-BS) algorithm, for large databases without the use of large invalidation report. The algorithm is applicable to the "information feed" application domain in which the clients have a dynamically changed and shared locality of accesses. In the ML-BS algorithm, each bit with coarse-granularity in the kL-BS structure ($k > 0$) is used to denote a group of rarely (or slowly) changed data pages, while each bit with fine-granularity in the 0L-BS structure is used to represent one "hot" data page. If the clients can always use the fine-granularity 0L-BS structure for the validation of "hot" data pages, the false invalidation ratio will be expected to be very low.

In a mobile environment, it is highly desirable to provide a cache invalidation algorithm that adapts dynamically as workload parameters change unexpectly. This adaptability is motivated by the observation that the frequent disconnection and movement of mobile clients renders the system workload highly unpredicatable. A flexible algorithm will be less severely impacted by these changing workload parameters. In this paper, it has been shown that this adaptability can be improved by utilizing the update information in the server. Ideally, a caching algorithm should be capable of adapting much better if the information from clients such as the query patterns and client disconnection time can be utilized. We plan to investigate the development of such adaptive algorithms on the basis of feedback regarding query patterns and client disconnection time in the future work.

References

[1] Sas user's guide. SAS Insititute Inc., Cary, NC, 1989.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, San Jose, California, 1995.

[3] 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.

[4] R. Alonso and H. Korth. Database issues in nomadic computing. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 388–392, 1993.

[5] B. R. Badrinath, A. Acharya, and T. Imielinski. Structuring distributed algorithms for mobile hosts. In *Proc. of the 14th International Conference on Distributed Computing Systems*, Poznan, Poland, June 1994.

[6] D. Barbará and T. Imielinksi. Adaptive stateless caching in mobile environments: An example. Technical Report MITL-TR-60-93, Matsushita Information Technology Laboratory, 1993.

[7] D. Barbará and T. Imielinksi. Sleepers and workaholics: Caching strategies for mobile environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1–12, 1994.

[8] D. Barbará and T. Imielinksi. Sleepers and workaholics: Caching strategies for mobile environments (extended version). *MOBIDATA: An Interactive journal of mobile computing*, 1(1), Nov. 1994. Available through the WWW, http://rags.rutgers.edu/journal/cover.html.

[9] M. J. Carey, M. J. Franklin, M. Livny, and E. Shekita. Data caching tradeoffs in client-server dbms architectures. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1991.

[10] M. J. Franklin. Caching and memory management in client-server database systems. *Ph.D. Thesis*, 1993. University of Wisconsin-Madison.

[11] Y. Huang, P. Sistla, and O. Wolfson. Data replication for mobile computers. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Minneapolis, Minessota, 1994.

[12] T. Imielinski and B. R. Badrinath. Wireless mobile computing : Challenges in data management. *Communication of ACM*, 37(10), 1994.

[13] T. Imielinski and S. Vishwanath. Adaptive wireless information systems. In *Proceedings of SIGDBS (Special Interest Group in DataBase Systems) Conference*, Tokyo, Japan, 1994.

[14] T. Imielinski, S. Vishwanath, and B. R. Badrinath. Energy efficient indexing on air. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Minneapolis, Minessota, 1994.

[15] T. Imielinski, S. Vishwanath, and B. R. Badrinath. Power efficient filtering of data on the air. In *Proceedings of the International Conference of EDBT (Extending DataBase Technology)*, 1994.

[16] H. Schwetman. Csim user's guide (version 16). MCC Corporation, 1992.

[17] S. Vishwanath and T. Imielinski. Pyramid broadcasting for video on demand service. In *Proceedings of the IEEE Multimedia Computing and Networks Conference*, San Jose, California, 1995.

Appendix A: Probability Analysis of the False Alarm in SIG:

In the SIG algorithm presented in [7], there is the probability of diagnosing a cache (of data item) as invalid, when it is not. When the number of data items that have been updated since the last combined signatures were cached is equal to or less than f, the probability (which was given in [7]) is:

$p_f = Prob[X \geq Kmp] \leq exp(-(K-1)^2 m\frac{p}{3})$

where f is the number of different items up to which the algorithm is designed to diagnose, $1 \leq K \leq 2$, m is the number of combined signatures determined by $6g(f+1)(ln(1/\delta) + ln(N))$, p ($\approx \frac{1}{1+f}(1-\frac{1}{e})$) is the probability of a valid cache being in a signature that does not match, and X is a binomial variable with parameters m and p.

However, when the actual number n_u of updated data items at server is greater than f, the probability $p_f^{n_u}$ of incorrectly diagnosing a valid cache by the algorithm will be different from p_f. Using the similar analysis procedure as in [7], the probability can be computed as follows. To compute $p_f^{n_u}$, we shall first compute the probability p^{n_u} of a valid cache being in a different signature. For this to happen, the following must be true:

1. This item must belong to the set in the signature. The probability is $\frac{1}{f+1}$ (Notice that in the SIG approach, each signature is corresponding to a set of data items and each set is chosen so that an item i is in the set S_i with probability $\frac{1}{f+1}$).

2. Some item that has been updated since the last time the signature report was cached must be in the set and the signature must be different. The probability will be $(1-(1-\frac{1}{f+1})^{n_u})(1-2^{-g})$ where n_u is the number of data items that have been updated since the last time the signature report was cached, g is the size (in bits) of each signature (Notice that the probability that the two different values of an item have the same signature is 2^{-g}).

Thus, the probability p^{n_u} of a valid cache being in a signature that does not match is:
$p^{n_u} = \frac{1}{f+1}(1-(1-\frac{1}{f+1})^{n_u})(1-2^{-g}) \approx \frac{1}{f+1}(1-(1-\frac{1}{f+1})^{n_u})$
Now we can define a binomial variable X^{n_u} with parameter m and p^{n_u}. Then, the probability of incorrectly diagnosing a valid cache can be expressed as the probability that the variable X^{n_u} exceeds the threshold ($=Kmp$) of the SIG algorithm. That is,
$p_f^{n_u} = Prob[X^{n_u} \geq Kmp]$

n_u	10	20	30	40	50
$p_{10}^{n_u}(m=1500)$	0.00048	0.33112	0.76935	0.87915	0.92678
$p_{20}^{n_u}(m=2900)$	5.55E-16	0.00174	0.07992	0.41943	0.68977

Table 3: The Values of Probability $p_f^{n_u}$ When $f = 10, 20$

Table 3 gives a set of values of the probability which were computed using SAS package [1] for $f = 10, 20$, $m = 1500, 2900$ (where m is computed by $6(f+1)(ln(1/\delta)+ln(N))$ with $N = 1000$ and $\delta = 10^{-7}$), $K = 1.4$ and n_u = 10-50. The results indicate that the probability of incorrectly diagnosing a valid cache increases quickly when n_u grows from 10 to 50.

# of Differing Items	10	20	30	40	50	...
# of Being Invalided($f = 10$)	10	263	810	909	934	...
# of Being Invalided($f = 20$)	10	20	124	462	745	...

Table 4: The Actual Number of Differing Items vs. The Number of Items to be Invalided

To verify our probability analysis, we conducted a set of simulation experiments to demonstrate the relation between the actual number of differing items and the number of items to be invalidated. In the experiments, we used the same parameters as in the probability computation for Table 3. That is, $N = 1000$, $\delta = 10^{-7}$, and $K = 1.4$. The experiments compute combined signatures for two

databases with n_u (=10-50) different items (the total number of items in each database is 1000) and use the SIG algorithm to generate the data items to be invalidated. The simulation results shown in Table 4 also indicate that the number of data items to be invalidated increased quickly when the actual number of differing items exceeds the parameter f. In these results, the set of data items to be invalidated is always a superset of the differing items. □