Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments

Jin Jing
Mobile Communication Operations
Intel Corporation
2511 NE 25th Avenue
Hillsboro, OR 97124
Jin_Jing@ccm.jf.intel.com

Ahmed Elmagarmid [*]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA
ake@cs.purdue.edu

Abdelsalam (Sumi) Helal
MCC
3500 West Balcones Center Dr.
Austin, Texas 78759-6509
helal@mcc.com

Rafael Alonso
David Sarnoff Research Center
CN 5300
Princeton, NJ 08543
ralonso@sarnoff.com

Abstract

In this paper, we present *Bit-Sequences* (BS), an adaptive cache invalidation algorithm for client/server mobile environments. The algorithm uses adaptable mechanisms to adjust the size of the invalidation report to optimize the use of a limited communication bandwidth while retaining the effectiveness of cache invalidation. The proposed BS algorithm is especially suited for dissemination-based (or "server-push" based) nomadic information service applications. The critical aspect of our algorithm is its self-adaptability and effectiveness, regardless of the connectivity behavior of the mobile clients. The performance of BS is analyzed through a simulation study that compares BS's effectiveness with that of a hypothetical optimal cache invalidation algorithm.

1 Introduction

In mobile and wireless environments, caching of frequently-accessed data is critical for reducing contention on the narrow bandwidth channels. Classical cache invalidation strategies in these environments are likely to be severely hampered by the disconnection and mobility of clients. It is difficult for a server to send invalidation messages directly to mobile clients because they often disconnect to conserve battery power and are frequently on the move. For the client, querying data servers through wireless up-links for cache invalidation is much slower than wired links because of

[*]The work by Elmagarmid is supported by grants from the Intel, Bellcore, and IBM Corporations, and a Purdue Reinvestment garnt.

the latency of wireless links. Also, the conventional client/server interactions cannot scale to massive numbers of clients due to narrow bandwidth wireless links.

In [6], 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 changed data items are indicated. Rather than querying a server directly regarding the validation of cached copies, clients can listen to these invalidation reports over wireless channels. The broadcast-based solution is attractive because it can scale to any number of mobile clients who listen to the broadcast report.

However, a major challenge for broadcast-based solutions is to optimize the organization of broadcast reports. In general, a large report can provide more information and is more effective for cache invalidation. But a large report also implies a long latency for clients while checking the report, given a limited broadcast bandwidth. The Broadcasting Timestamp (TS) [6] is a good example of an algorithm that limits the size of the report by broadcasting the names and timestamps only for the data items updated during a window of w seconds (with w being a fixed parameter). Any client who has been disconnected longer than w seconds cannot use the report before establishing an up-link for cache verification. Unfortunately, the *effectiveness* (reliability) of the report under TS cannot be guaranteed for clients with unpredictable disconnection time. The effectiveness can be measured by the number of cached data items whose status can be accurately verified by the report.

In general, there is a tradeoff between the size and the effectiveness of broadcast reports. The tradeoff is particularly subtle for the clients which cannot continuously listen to the broadcast. In this paper, we address the report size optimization problem. That is, given an effectiveness requirement, how can we optimize the report structure? We present three optimization techniques.

First, for applications where cached data items are changed less often on the database server, we use the bit-sequence naming technique to reference data items in the report. In the bit-sequence naming, each bit in a bit-sequence (or bit-vector) represents one data item in the database. Second, instead of including one update timestamp for each data item, we use an *update aggregation* technique to group a set of data items and associate the set with only one timestamp in the report. The client disconnected after the timestamp can use the bit-sequence to identify the updated items. Third, we use a *hierarchical structure of bit-sequences* technique to link a set of bit-sequences so that the structure can be used by clients with different disconnection times. In this paper, we present a new algorithm called *Bit-Sequences* (BS) that use these three techniques.

The proposed BS algorithm can be applied in applications where the frequently cached and referenced data items are predictable. In these applications, both the servers and the clients do not need to frequently synchronize the mapping of bits in the sequence (or vector) to the names of data items in the database. The static mapping does not have to be explicitly included in the report. The bits in the sequence are used to represent those data items in the database that are frequently cached and referenced by the majority of clients. The BS algorithm can also be used in applications where clients can continuously listen to the broadcast report for cache invalidation or the static bit mapping is not possible. In these applications, a dynamic mapping from data items to bits is explicitly included in the report along with the bit sequence structure.

The main contributions of this paper include the following:

1. when a static bit mapping scheme is implicitly assumed, the BS algorithm can approach the "optimal" effectiveness for all data items indicated in the report regardless of the duration of disconnection of the clients. However, such optimization can be achieved only at the cost of about 2 binary bits for each item in the report

2. the BS algorithm can also be applied to optimize other broadcast-based cache invalidation algorithms in which the dynamic bit mapping has to be included explicitly. The optimization reduces the size of the report by about one half while maintaining the same level of effectiveness for cache invalidation.

The remainder of the paper is organized as follows. Section 2 describes the Bit-Sequences (BS) algorithm. Section 3 discusses the relationship between invalidation effectiveness and bit mapping in the use of the BS algorithm. In section 4, we examine, through simulation experiments, how the BS algorithm compares to the "optimal" algorithm for cache invalidation. Section 5 discusses related research. Concluding remarks are offered in Section 6.

2 The Bit-Sequences Algorithm

2.1 Caching Management Model

A mobile computing environment consists of two distinct sets of entities: *mobile hosts* and *fixed hosts* [10, 6, 4]. Some of the fixed hosts, called *Mobile Support Stations* (MSSs), are augmented with a wireless interface in order to communicate with the 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 connections. There is a set of database servers; each covers one or more cells.

Each server can only service users who are currently located in its coverage. A large number of mobile hosts reside in each cell; issuing queries requesting 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 periodically broadcasts invalidation reports. To answer a query, the client on a mobile host listens 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 returns the result immediately. Invalid caches must be refreshed via a query to the server.

2.2 Optimization Techniques

In the Bit-Sequences (BS) algorithm, three techniques are used to optimize the size of the report structure while retaining the invalidation effectiveness:

- *bit-sequence naming*,
- *update aggregation*, and
- *hierarchical structure of bit-sequences.*

To reference data items in the database, a technique called *bit-sequence naming* is applied in the BS algorithm. The server broadcasts a set of bit sequences. Each *bit* in a bit-sequence represents a data item in the database. The position of bits decides the indexes of numbered data items. For example, the nth bit in a size N of sequence represents data item d_n. Therefore, the naming space for N items is reduced to N bits from $Nlog(N)$ bits. It should be noted that without the order information, at least $log(N)$ bits are needed to identify an item in a set of size N. The bit-sequence naming can be applied when both client and server agree upon the mapping of bits to the names of data items in the server database. The client can find the data item that each bit represents in its cache based on the position of the bit in the sequence.

To indicate the update status of data items, another technique called *update aggregation* is used. In the broadcast report, each sequence is associated with only one timestamp. A bit "1" in a sequence means that the item represented by the bit has been updated since the time specified by the timestamp. A bit "0" means that that item has not been updated since the specified time. Note that the timestamp is not necessarily the exact time when the items represented by "1" bits were updated. Instead, the timestamp specifies the time when all these items were updated. This technique, therefore, helps reduce the report size by associating a single timestamp to a set of updated items rather than a timestamp to each item. For example, for a size m sequence, $32*(m-1)$ bits are saved, assuming that each timestamp is represented by 32 bits (4 bytes or a DWORD type variable). A client that disconnected after the timestamp can use such information in the sequence to make invalidation decision.

The update aggregation not only reduces the size of the report, but also decreases the invalidation precision of cache invalidation. For example, a sequence with a three day-old timestamp may not be very useful for the client who disconnected just three hours ago. Many updates indicated in the sequence actually happened before the client disconnected. If the client uses the bit sequence, many valid data items will be falsely invalidated. To adapt to variable disconnected clients, a technique called *hierarchical structure of bit sequences* is applied. In this technique, $log(N)$ bit sequences with different timestamps and sizes are linked together for N data items covered in the report. From the set of bit sequences, each client uses one bit sequence with the timestamp which is equal to or most recently predates the disconnection time of the client for cache invalidation.

The total size of these bit sequences can only be $2N + b_T log(N)$ bits (where b_T is the size of each timestamp). In this hierarchical structure, the highest-ranking sequence in the structure has 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 latest $N/2$ 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 recently updated. 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 recently updated. 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 recently changed. The timestamp of the sequence indicates the time after which the item was updated.

2.3 The Algorithm

Now, we will describe the BS algorithm that applies the optimization techniques described above. For simplicity, we assume that there are N data items in the database where N is the n power of 2, that is, $N = 2^n$ for some integer n. We also assume that each item is statically (or implicitly) mapped to one bit in the highest sequence (note that the 1-to-1 mapping is actually not necessary in the use of the BS algorithm; we will elaborate on this later). 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.

Each client checks the invalidation and uses a bit sequence among the sequence set with the most recent timestamp that is equal to or predates the disconnection time of the client in order 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 report will not be used, and the client has to establish an up-link for cache invalidation.

Example 2.1 *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*

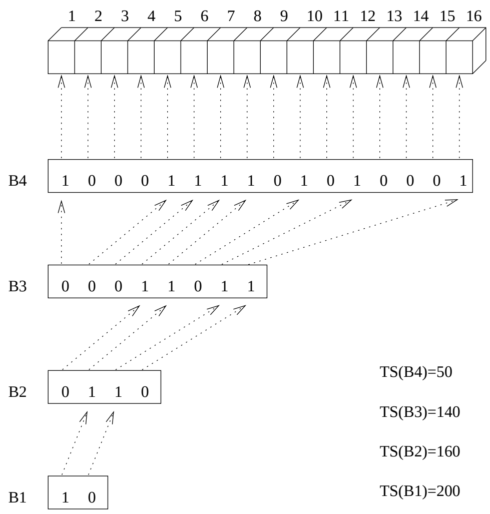

Figure 1: A Bit-sequences example

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. Initially, all bits in each bit sequence are reset to "0" and the timestamp of each bit sequence is also reset to "0". The highest bit sequence B_n will contain $N/2$ "1" bits only after at least $N/2$ data items have been updated since the initial (starting) time. The server will send the broadcast report periodically after the initial time. From the initial time to the time when $N/2$ data items are updated, the highest bit sequence contains more "0" bits than "1" bits (recall that "0" bits mean that the data items represented by these bits have not been updated since the time specified by the timestamp; clients will keep cached data items indicated by "0" bits as valid data items). After more than $N/2$ data items have been updated, the $N/2$ recently updated data items will be indicated by the bits in the highest bit sequence B_n. Therefore, after the time when more than $N/2$ data items are updated, B_n always contains $N/2$ "1" bits.

To construct the BS structure, the server should keep a *update linked list* which contains $N/2$ recently updated data items in a update timestamp order (or all updated data items if less $N/2$ data items have been updated since the initial time). The $N/2$ bits for these data items in B_n will be set to "1" (the "1" bits are less than $N/2$ before $N/2$ bits have been updated since the initial time). The next bit sequence B_{n-1} will contain $N/4$ "1" bits for $N/4$ recently updated data items (or half of "1" bits of B_n for these recently updated data items if less than $N/2$ data items have been updated since the initial time). Each bit sequence is attached to a timestamp that is the latest time since those items indicated by "1" bits have been updated.

In the update linked list, each data item can be denoted by a node. 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 decreasing order of update timestamps. That is, the first node in the update linked list denotes the data item that was most recently updated; the second node denotes that data item that was 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 $TS(B_0) \leq T_l$, no data cache need to be invalidated. Stop;
2. if $T_l < TS(B_n)$, the entire cache is invalidated. Stop;
3. Locate the bit sequence B_j with the most recent timestamp that is equal to or predates the disconnect 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. □

2.4 Invalidation Precision

In the Bit-Sequences (BS) algorithm, a client will use a bit sequence, say B_k $(1 \leq k \leq n)$, to invalidate its cache if the client started disconnection at a time which is equal to or larger than $TS(B_k)$ but smaller than $TS(B_{k-1})$ (we assume $TS(B_0)$ is equal to ∞). By the definition of Bit-Sequences, we know that as many as ΣB_k data items that are indicated by "1" bits in B_k have 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 that are actually valid, but falsely invalidated.

However, the real number of falsely invalidated data items will actually depend on several factors, such as the last time the cached data were validated, and the query/update patterns, etc.

To see how the disconnection/update pattern impacts the false invalidation, assume that the client started the disconnection at time T_d (i.e., the last time when the cached data were validated)

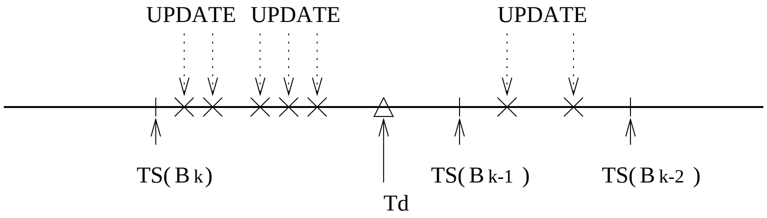

Figure 2: Updates vs. disconnection scenario 1

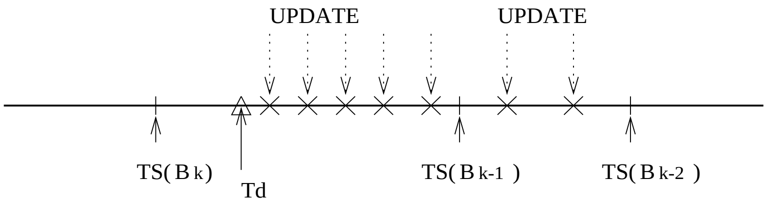

Figure 3: Updates vs. disconnection scenario 2

where $TS(B_k) < T_d < TS(B_{k-1})$. 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_d *and* (2) the client validated these cached data after the updates (or before its disconnection). Figure 2 shows the scenario. On the other hand, if the client disconnected at the time T_d before the $\Sigma B_k/2$ data items were updated, then these invalidated data items are actually obsolete ones, and no data item is falsely invalidated because they were updated between T_d and $TS(B_{k-1})$. Figure 3 gives the scenario.

Therefore, we expect that the actual rate of false invalidation to be quite low. The simulation study in the late section will verify this observation.

3 Effectiveness vs. Bit Mapping

As we have defined earlier, the effectiveness of a report can be measured by the number of cached data items that can be accurately verified for a client by the use of the report. Different requirements for the effectiveness can be achieved by different bit mappings in the BS algorithm. In this section, we will discuss two types of bit mapping schemas that define two different effectiveness requirements. We will also analyze the report size for each mapping scheme.

Update Window Scheme (Dynamic Bit Mapping):

Like the TS algorithm described in [6], bits in the report can only represent data items that have been updated within the preceding w second window. In this case, the bit mapping to data items is dynamically changed for each report, and the dynamic mapping has to be explicitly included in each report so that clients know the dynamically changed mapping relationship. Since the mapping contains only the update information for the last w seconds, the effectiveness of the report will depend on the types of clients. In fact, the report with the dynamic mapping, similar to the TS algorithm, is effective only for clients who disconnected less than w seconds ago, but not for clients who disconnected more than w seconds ago.

The bit mapping in a report can also indicate only the last f updated data items. In this case, the effectiveness of the report is similar to that in the SIG algorithm in [6] (see the analysis included in Appendix A). The analysis shows that the SIG algorithm is effective only if the number of updated data items does not exceed f, an algorithm parameter). Because the data items within the f number window may change from time to time, it is imperative that a dynamic bit mapping scheme should be explicitly included to map names of data items to bits in the report.

In the TS algorithm, each report includes the name and the update timestamp of each data item. Therefore, the size of each report is $k * (b_T + log(N))$ bits where k is the number of data items in the report, b_T is the bits of each timestamp, and the $log(N)$ is the bits of each name of data item (assuming that N is the maximum number of data items in database servers and that each name is coded by $log(N)$ binary bits). In comparison, the BS algorithm with dynamic (explicit) bit mapping uses approximately $k*(2+log(N))+b_T*log(k)$ bits. That is, $k*log(N))$ bits are used for the dynamic bit mapping explicitly included in the report, $2k$ for log(k) bit sequences (the highest sequence has k bits and the lowest has only 2 bits), and $b_T * log(k)$ for $log(k)$ timestamps of the bit sequences. In practice, each timestamp or data item can be represented by a long integer of 4 bytes. The total number of bits used by the TS algorithm is about $(32+32)*k$ or $2*32*k$ and the number of bits used by the BS algorithm is around $2*k + 32*k + 32*log(k)$. For a large k, the latter is about half of the former.

The report size in the SIG algorithm can be expressed, as in [6], as $6g(f+1)(ln(1/\delta)+ln(N))$, where N is the number of data items in the database, 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. Typically, g and δ are set to 16 and 10^{-7}, respectively [6]. For a database with $N = 1024$, the report size will approximately be $2000*f$ (or $6*16*(f+1)*(ln(10^7)+ln(10^3))$), which is much larger than $32*(f+log(k))$ in the BS algorithm with the dynamic (explicit) bit mapping.

Therefore, using the BS algorithm with a dynamic (explicit) bit mapping, the size of the report is close to half of that in the TS algorithm (and much smaller than that in the SIG algorithm) while retaining almost the same level of effectiveness for cache invalidation. The bit saving is achieved by the optimization in the update aggregation and the hierarchical structure techniques. In the next section, the effectiveness of the BS algorithm will be shown–using simulation– to be very close to the effectiveness of a hypothetical "optimal" algorithm.

Hot Set Scheme (Static Bit Mapping):

In the previous section, we used a strict one-to-one mapping to demonstrate how the BS algorithm works. The example in that section uses a static (implicit) bit mapping. In general, the BS algorithm with static mapping is able to maintain the effectiveness of the report for the data items covered to all clients regardless of the length of their disconnection.

In practice, the strict one-to-one mapping may not be feasible because of the large size of the database in the server. On the other hand, it is also not necessary to have the strict one-to-one mapping. In fact, from the clients' point of view, only data items that are cached and often referenced are of interest. For many database and applications, the hot spot (i.e., the set of data items most cached and referenced) is changed less frequently. For these applications, the BS algorithm with the static mapping can be applied to reference these data items and can adapt to the changes of the disconnect times of the clients and variable update rates in database servers.

In the BS algorithm with static bit mapping, the size of the report can be expressed as a function of the number of data items: 2k + $b_T$log(k), where k is the total number of data items covered in the report and b_T is the size of a timestamp. For a large k, the size approaches $2k$ (and less than $3k$). For example, when k=1,000 and $b_T = 32$, the size is about 2300 bits (= 2*1,000 + 32*10). Therefore, the effectiveness of the report is achieved at the cost of about 2 bits (less than 3 bits) for each data item covered in the report.

Hybrid BS Scheme (Static and Dynamic Bit Mappings):

Another way to improve the effectiveness without increasing the size of report is to use coarse granularity technique. That is, one bit can be used to represent a block of data items rather than a single data item. If any item in the clock is updated, the bit that represents the block is set to "1". Clients have to invalidate all cached data items represented by the coarse granularity bit, if the bit is set to "1". In general, therefore, the coarse granularity technique is suitable for the data set that is less frequently changed.

The effectiveness of coarse granularity technique can be improved further when a hybrid BS scheme of bit mappings is used. Specifically, the BS with dynamic bit mapping can be used to indicate the updated data items in the last w seconds while the BS with static and coarse granularity bit mapping can be used to cover all other hot data items (i.e., those data items that are mostly cached and referenced often, except the recently updated data items that are included in the dynamic BS scheme). The advantage of using the hybrid scheme is that the coarse granularity bits in the static BS scheme will not be set to "1" immediately even though some items in the data set (indicated by the bits) have recently been updated. In other words, clients do not need to invalidate most of the data items covered by the coarse granularity bits even though some data items covered in the coarse bit set have been updated recently.

The hybrid BS scheme with the coarse granularity technique has been studied and analyzed in [14]. The hybrid BS scheme enables the static BS scheme to cover more data items without increasing the size of the report and to be effective for all clients regardless of their disconnect times. A similar hybrid approach has also been described in [17]. The approach in [17] uses an up-link message to check the status of cached data items other than the recently updated data items. In contrast, the hybrid BS scheme uses the static BS scheme in the broadcast report to check the status of these items.

In summary, the effectiveness of the report can be affected by different bit mapping schemas and the techniques used in the BS algorithm can be used to optimize the size of the report for different effectiveness requirements.

4 Performance Analysis

The performance analysis presented here has been designed to show how the invalidation precision of the BS algorithm can approach that of an "optimal" algorithm where all invalidated data items are actually stale ones. Recall that the techniques used in the BS algorithm may invalidate some items that are actually valid in order to reduce the size of the report. We compared the BS and the hypothetical "optimal" algorithms under different workload parameters, such as disconnect time, query/update pattern, and client buffer size, etc.

The performance metrics in the study is the cache hit ratio of clients after reconnection. The cache hit ratio is computed by dividing the sum of the queries that are answered using client caches by the sum of the total queries in the simulation. These queries include only the first *ten* queries after each wake-up, as we are interested in how the cache hit ratios would be affected by the use of invalidation report after reconnection.

For simplicity, we model a single server system that services a single mobile client. The assumption is reasonable because we only measure the cache hit ratios of clients after wake-up and the broadcast report can be listened by any number of clients for cache invalidation. 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. We assume that the cache pool in the server is large enough to hold the entire database. The size of client cache pools is specified as a percentage of the database size. The cache pools are filled fully with cached data before the disconnection of the client. Each mobile host generates a single stream of queries after reconnections. After the stream of queries, the client may be disconnected with an exponentially distributed time.

We compare the BS based algorithms with an "optimal" cache invalidation algorithm that has no false invalidation to cached data. The "optimal" algorithm assumes that clients precisely invalidate only staled cache data without delay. The algorithm is not implementable practically, as it requires an infinite bandwidth between clients and servers for non-delayed invalidation. We call this algorithm as BASE algorithm and use it as a basis in order to gain an understanding of performance in a simplified simulation setting and as a point-of-reference for the BS algorithm. We compare the BS and BASE algorithms with variable disconnections and access patterns.

Our model 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 effect of false invalidation on the cache hit ratio of algorithms. All simulations were performed on Sun Sparc Workstations running SunOS and using a CSIM simulation package [15].

4.1 Parameters and Settings

Our model can specify the item access pattern 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 selected for access according to 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.

A *cache coherency* parameter is used to adjust the cache "locality" of queries. The cache coherency specifies how often cached data will be reused. If the parameter is set to 0.5, half of the queries will access cached data and the rest of queries will follow the query access pattern specified by the simulation parameter setting.

Parameter	Setting
Database Size	1,000 data items
Client Buffer Size	5%, 15%, 25%, 35% of database size
Cache Coherency	0, 0.5
Mean Disconnect Time	10,000 to 1,000,000 seconds
Mean Update Arrive Time	1,000 seconds

Table 1: Simulation parameter settings

Parameter	UNIFORM	HOTCOLD
HotUpdateBounds	-	first 20% of DB
ColdUpdateBounds	all DB	remainder of DB
HotUpdateProb	-	0.8
HotQueryBounds	-	first 20% of DB
ColdQueryBounds	all DB	remainder of DB
HotQueryProb	-	0.8

Table 2: Access parameter settings

Table 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. The UNIFORM (query or update) workload is a low-locality workload in which updates or queries are uniformly distributed. The HOTCOLD (query or update) workload has a high degree of locality of updates or queries in which 80% of queries or updates are performed within 20% portion of databases. Different combinations of query and update patterns can be used to model different application access characteristics. For example, the HOTCOLD query and HOTCOLD update workload is intended to model an "information feed" situation where the data server produces data to be consumed by all clients. This situation is of interest because information services are likely to be one of the typical applications in mobile computing environments [10].

4.2 Experiment 1: BS vs. BASE

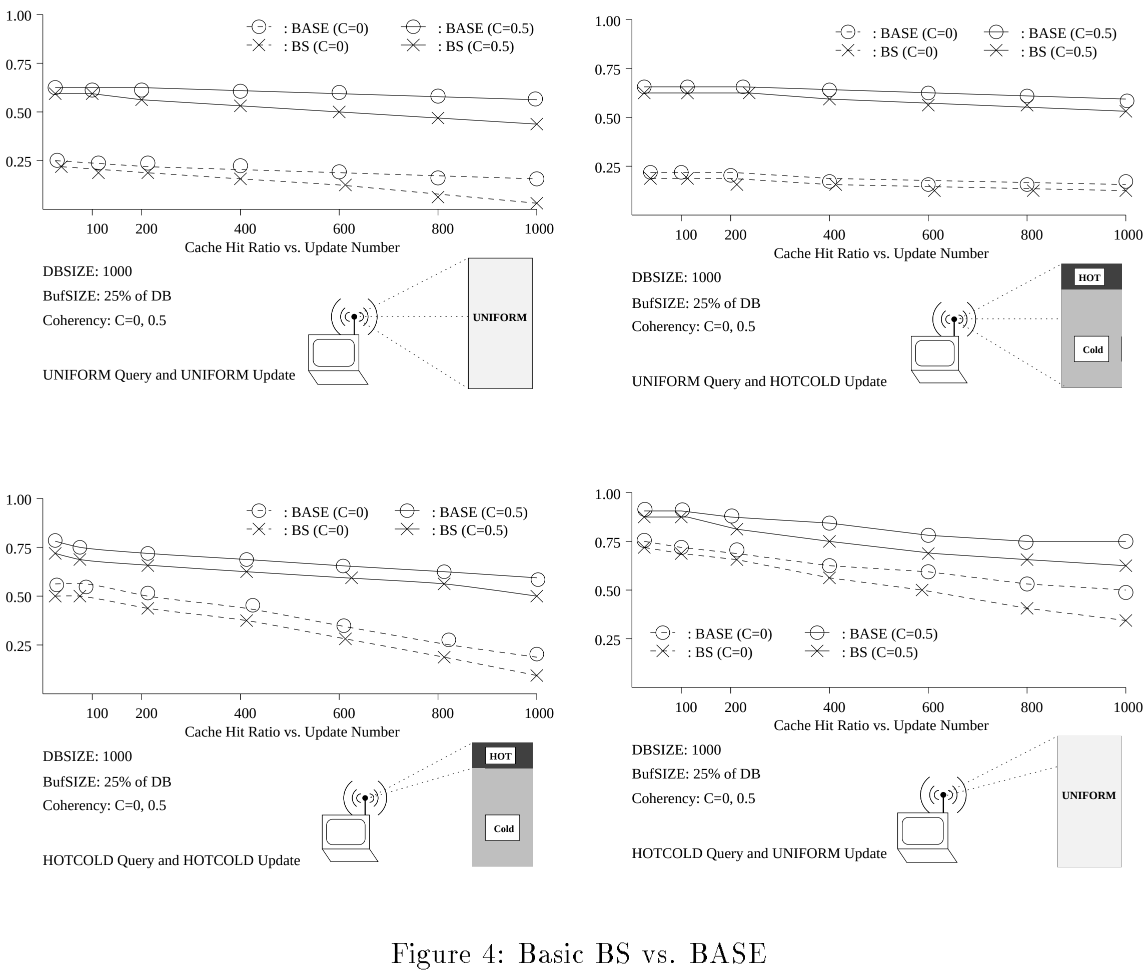

Figure 4: Basic BS vs. BASE

Figure 4 shows the experimental results that illustrate how the cache hit ratios are affected by the false invalidation introduced by the BS algorithm. The cache hit ratios are measured after clients wake-up from disconnection and use the BS structure to validate their cache. The first ten queries

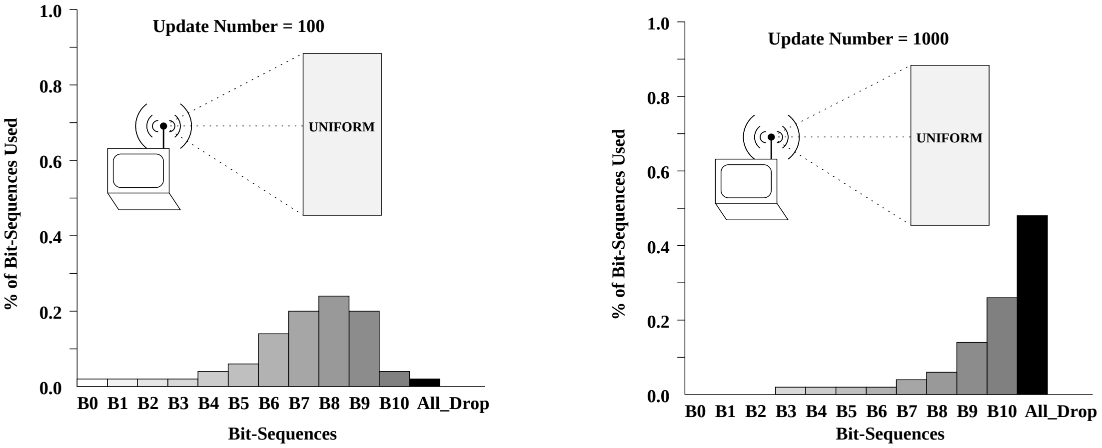

Figure 5: Percentages of Bit-sequences used

after wake-up are counted for the computation of cache hit ratios, so the measurement can correctly reflect the effect of false invalidation introduced by the BS algorithm on the cache hit ratios. As a point of reference, the cache hit ratios for BASE algorithm where there is no false invalidation of cached data are shown in the Figure 4. The cache hit ratios for both algorithms are measured under two different cache coherency parameters. Solid lines depict the results when the parameter is equal to 0.5 while dashed lines represent results when the parameter is equal to 0. These experiments assume that clients hold 25% of database items in their cache before disconnections.

In Figure 4, the horizontal axis represents the mean number of updates during the client disconnect period while the vertical axis is the cache hit ratio. Because a constant mean update arrival time of 1,000 seconds is used in these experiments, the mean update number is equivalent to the disconnect time of the client (i.e., the mean disconnection time of client is equal to the multiplication of the mean update number and the mean update arrive time.). For example, the mean update number 1,000 implies the mean 1,000,000 seconds disconnection of client. Therefore, the results in these experiments indicate the relationship between the cache hit ratios and the disconnections of clients.

The comparison between the BASE and BS algorithms indicates that the basic BS algorithm adapts well to variable disconnections, update rates, and access patterns. In most cases, the cache hit ratios of the BS algorithm are close to those of the BASE algorithm. However, the difference of cache hit ratios between two algorithms increases when the number of updates (or the disconnection) increases (e.g., the UNIFORM query and UNIFORM update case in Figure 4). The reason for this increase is that a higher ranking bit sequence will be used for cache invalidation for a longer disconnected client (with a large number of updates in servers during the disconnection). Figure 5 shows the percentages of bit sequences used by clients for two different disconnect times (or two different update numbers) in the UNIFORM update and query case. A large update number (i.e., long disconnection) means that a high ranking bit sequence is used. The higher bit sequence contains more sequent bits and in the worst case, about half of the items represented by these bits may be falsely invalidated. The high false invalidation, therefore, introduces low cache hit ratios.

We note that the difference between the BASE and BS algorithms in the HOTCOLD update cases is not as large as that in the UNIFORM update cases with the same update number. This is because the actual number of different data items that are updated in HOTCOLD update is smaller than the number in UNIFORM update, although the total number of updates are the same in both cases. A small number of updated data items implies that a low level bit sequence is used in cache invalidation, as shown in Figure 5.

4.3 Experiment 2: BS vs. BASE (Varying Cache Sizes)

Figures 6 and 7 show the cache hit ratio as a function of client cache size that changes from 5% to 35% of the database size. The cache hit ratio is measured with two different update numbers: 100 and 1,000. These updates are performed in the data servers during the disconnection of the mobile clients.

As shown in Figure 6, the cache hit ratio increases with the increase of the cache size. Especially, when the coherency parameter is 0, the rate of increase of the cache hit ratio becomes close to that of the cache size. When the coherency parameter becomes 0.5, the rate of increase declines slightly. This is because half of the queries are not affected by the increased cache size. Recall that when the coherency parameter is 0.5, half of the queries always hit the client caches. Also, the rate of increase of the cache hit ratio when the update number is 1,000 is a slightly smaller than when the update number is 100. This is because frequent updates reduce the chances of cache hits.

Figure 7 shows that, for the HOTCOLD query pattern, the cache hit ratio drops fast with the decrease of cache size. The reason is that for the HOTCOLD query pattern, 80% of the queries will be executed in the 20% region (hot pages) of the database. When the cache size is less than 20% of database size, many of these 80% queries can not be answered by cached data items. This is the reason that the cache hit ratio drops quickly when the cache size is less than 20% and increases slowly when the cache size is larger than 25%.

In summary, large-size client cache helps increase the cache hit ratio for the UNIFORM query access. The increase rate of the cache hit ratio for the HOTCOLD query pattern is slowed down after the cache size is larger than the size of hot portion of the database.

5 Related Research

Many caching algorithms have been proposed recently for the conventional client-server architecture in which the positions and wired-network connections of clients are fixed. A comprehensive discussion

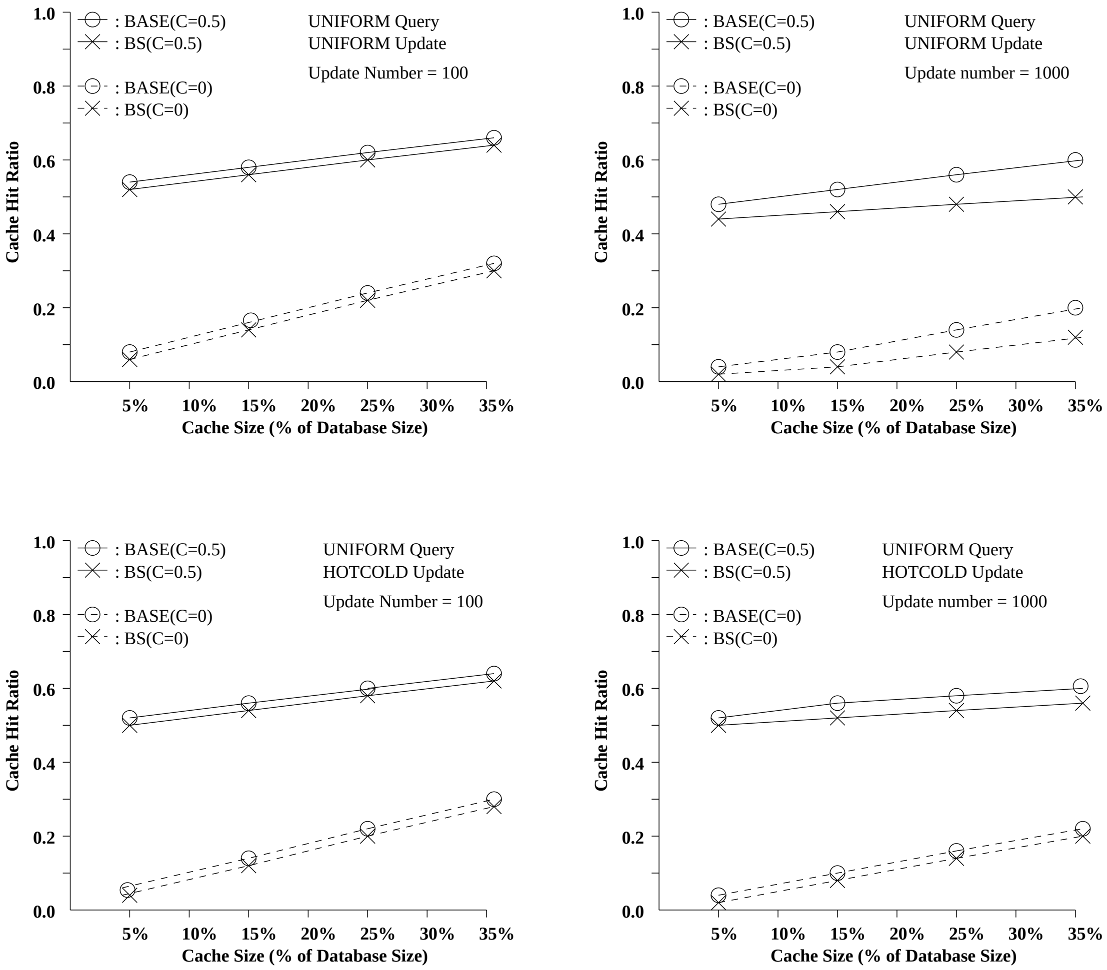

Figure 6: BS vs. BASE: varying cache sizes, UNIFORM query

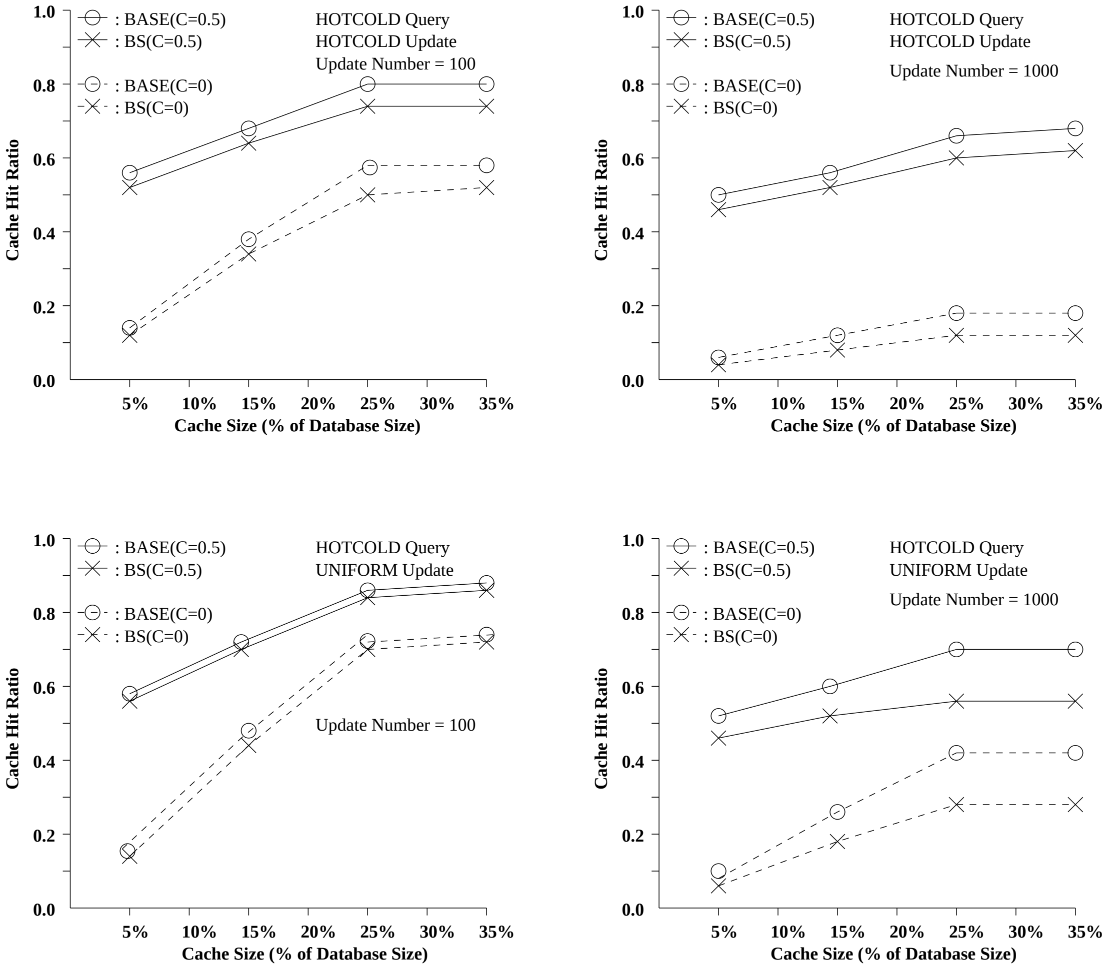

Figure 7: BS vs. BASE: varying cache sizes, HOTCOLD query

and comparison of caching algorithms in the conventional client-server architecture can be found in [8]. The issue of false invalidation does not exist in this architecture because, as shown in the algorithms discussed in [8], 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 would be invalidated.

Recently, the notion of using a repetitive broadcast medium in wireless environments has been investigated. The property of a data broadcast program which provides improved performance for non-uniformly accessed data was investigated in [2]. The authors in [3] also addressed the impact of update dissemination on the performance of broadcast disks. The scheme for update dissemination in [3] considers only continuous connectivity of clients in the broadcast disk environments.

The mobile computing group at Rutgers has investigated techniques for indexing broadcast data [12, 13]. 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 that broadcasts data for video on demand has been addressed in [16]. This 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 [11]. The adaptability is achieved by varying the frequency of the broadcast of individual data items according to the frequency of requests.

In [6], 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. 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 be invalidated when the number of data items that were updated at the server during the disconnection time exceeds f (see the analysis in Appendix A). Thus, the SIG algorithm is best suited for clients that are often disconnected when the update rate is low while the AT and TS algorithm is advantageous for clients that are connected most of the time.

To support long client disconnections, the idea of adapting the window size of the TS algorithm was discussed in [5, 7]. The approach in [5, 7] 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. [1] However, no detailed algorithm was presented in [5, 7] to show

[1]Consistency problems might arise if the window size for a data item is not included in an invalidation report.

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 hybrid BS scheme with the coarse granularity technique have been studied and analyzed in [14]. A simialr hybrid approach has also been described in [17]. The approach in [17] uses an up-link message to check the status of cached data items other than the most recently updated data items. By comparison, the hybrid BS scheme uses the static BS algorithm in the broadcast report to check the status of these items.

The work in [9] discusses the data allocation issues in mobile environments. The algorithms proposed in [9] 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.

6 Conclusions

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

Using simulation, we studied the behavior and performance of BS and have shown how the algorithm adapts itself dynamically as the update rate/pattern varies for the data items covered in the bit mapping. We have also shown how close the invalidation precision of the BS algorithm is compared to that of the "optimal" one (where clients invalidate only the data items that have actually been updated).

This paper examined how the effectiveness of reports that uses the BS algorithm can be affected by different bit mapping schemas. The BS algorithm with static (implicit) bit mapping was found to support clients regardless of the length of their disconnection times and offers the effectiveness of the report for data items covered in the report at the cost of about 2 bits/item. These bits can be used to cover the data items that are cacheable and most frequently referenced. The BS algorithm with dynamic (explicit) bit mapping was found to offer the same level of effectiveness at the expense of only about half of the report size.

Our study also revealed that the coarse granularity bit technique enables the static BS algorithm to cover more data items without increasing the size of the report. The hybrid BS scheme with the coarse granularity bit technique was also found to improve the effectiveness of the report (i.e., less false invalidation for data items covered by the coarse granularity bits) by including recently updated data items in a dynamic BS scheme in the 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.

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] S. Acharya, M. Franklin, and S. Zdonik. Disseminating updates on broadcast disks. In *Proceedings of VLDB*, Bombay, India, 1996.

[4] 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.

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

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

[7] D. Barbará and T. Imielinski. 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.

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

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

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

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

[12] 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, Minnesota, 1994.

[13] 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.

[14] J. Jing. Data consistency management in wireless client-server information systems. *Ph.D. Thesis*, 1996. Purdue University.

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

[16] 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.

[17] K. Wu, P. Yu, and M. Chen. Energy-efficient caching for wireless mobile computing. In *Proceedings of the IEEE Data Engineering Conference*, 1996.

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

In the SIG algorithm presented in [6], there is the probability of diagnosing a cache (of data item) as invalid when in fact 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 [6]) 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 the 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 [6], the probability can be computed as follows. To compute $p_f^{n_u}$, we 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 corresponds 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 items that have 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 the parameters 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
$p_{10}^{n_u}(m = 1,500)$	0.00048	0.33112	0.76935	0.87915	0.92678
$p_{20}^{n_u}(m = 2,900)$	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 was computed using SAS package [1] for $f = 10, 20$, $m = 1,500, 2,900$ (where m is computed by $6(f+1)(ln(1/\delta) + ln(N))$ with $N = 1,000$ 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 these experiments, we used the same parameters as in the probability computation for Table 3. That is, $N = 1,000$, $\delta = 10^{-7}$, 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 1,000) 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. $\square$