An Evaluation of Cache Invalidation Strategies in Wireless Environments

Kian-Lee Tan, *Member*, *IEEE Computer Society*, Jun Cai, and
Beng Chin Ooi, *Member*, *IEEE Computer Society*

Abstract—Caching can reduce the bandwidth requirement in a wireless computing environment as well as minimize the energy consumption of wireless portable computers. To facilitate mobile clients in ascertaining the validity of their cache content, servers periodically broadcast cache invalidation reports that contain information of data that has been updated. However, as mobile clients may operate in a doze or even totally disconnected mode (to conserve energy), it is possible that some reports may be missed and the clients are forced to discard the entire cache content. In this paper, we reexamine the issue of designing cache invalidation strategies. We identify the basic issues in designing cache invalidation strategies. From the solutions to these issues, a large set of cache invalidation schemes can be constructed. We evaluate the performance of four representative algorithms—two of which are known algorithms (i.e., Dual-Report Cache Invalidation and Bit-Sequences) while the other two are their counterparts that exploit selective tuning (namely, Selective Dual-Report Cache Invalidation and Bit-Sequences with Bit Count). Our study shows that the two proposed schemes are not only effective in salvaging the cache content but consume significantly less energy than their counterparts. While the Selective Dual-Report Cache Invalidation scheme performs best in most cases, it is inferior to the Bit-Sequences with the Bit-Count scheme under high update rates.

Index Terms—Mobile computing, disconnection, doze mode, bit-sequences, cache invalidation, access time, energy consumption.

1 Introduction

In today's increasingly mobile world, mobile users with battery-operated palmtops can access data stored at information servers located at the static portion of the network without space and time restriction [7], [8], [11], [14]. However, there are two obstacles to the wide-spread adoption of this technology: the limited bandwidth of wireless communication channels and the short battery lifespan of portable computers. Caching of frequently accessed data at the mobile clients has been considered to be a very effective mechanism in reducing wireless bandwidth requirements as well as energy consumption (since no energy is expended to transmit and receive data) [1], [2]. For caching to be effective, the cache content must be consistent with those stored in the server. This is, unfortunately, difficult to enforce due to the frequent disconnection and mobility of clients.

In the literature, the basic approach adopted is for the server to periodically broadcast invalidation reports that contain information about objects that have been updated recently. Based on the report, clients can invalidate objects that have been updated and salvage their cache content that are still valid. Most of the existing algorithms address three issues. The first issue deals with the content of the invalidation reports. The second issue concerns how invalidation is performed. The last issue looks at the support the server provides.

• *The authors are with the Department of Computer Science, National University of Singapore, 3 Science Drive 2, Singapore 117543. E-mail: {tankl, caijun, ooibc}@comp.nus.edu.sg.*

Manuscript received 15 Aug. 1999; revised 1 Sept. 2000; accepted 8 Mar. 2001.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 110443.

In this paper, we revisit the problem of designing cache invalidation strategies in wireless environments. This work is different from previous work in two ways. First, based on our understanding of the problem and the schemes proposed in the literature, we develop a framework for designing cache invalidation schemes from which a large class of energy efficient cache invalidation schemes can be constructed. Second, as case studies, we design two cache invalidation schemes that organize the invalidation reports to facilitate selective tuning. The performance of the new schemes is studied and compared with that of two existing algorithms. Our results show that the proposed schemes are not only effective in salvaging as much of the valid cache content as possible, but are also more energy efficient than existing schemes. To our knowledge, this is the first reported work that evaluates comprehensively several cache invalidation schemes.

Several observations motivated this work. First, while existing cache invalidation schemes proposed in the literature are different, most of them can be seen as variants of one another. We would like to develop a framework where we may identify the similarities and differences. We believe that one way to do so is to identify the issues that are involved in the development of a cache invalidation scheme. Second, most of the work has largely been evaluated against the basic cache invalidation scheme proposed by Barbara and Imielinski [2]; comparative evaluation against other algorithms is lacking. We believe a comprehensive comparative study of these algorithms will provide insights into the strengths and weaknesses of these schemes. In particular, the schemes should be compared on their ability to salvage the cache content as well as their energy utilization. Third, most of the existing

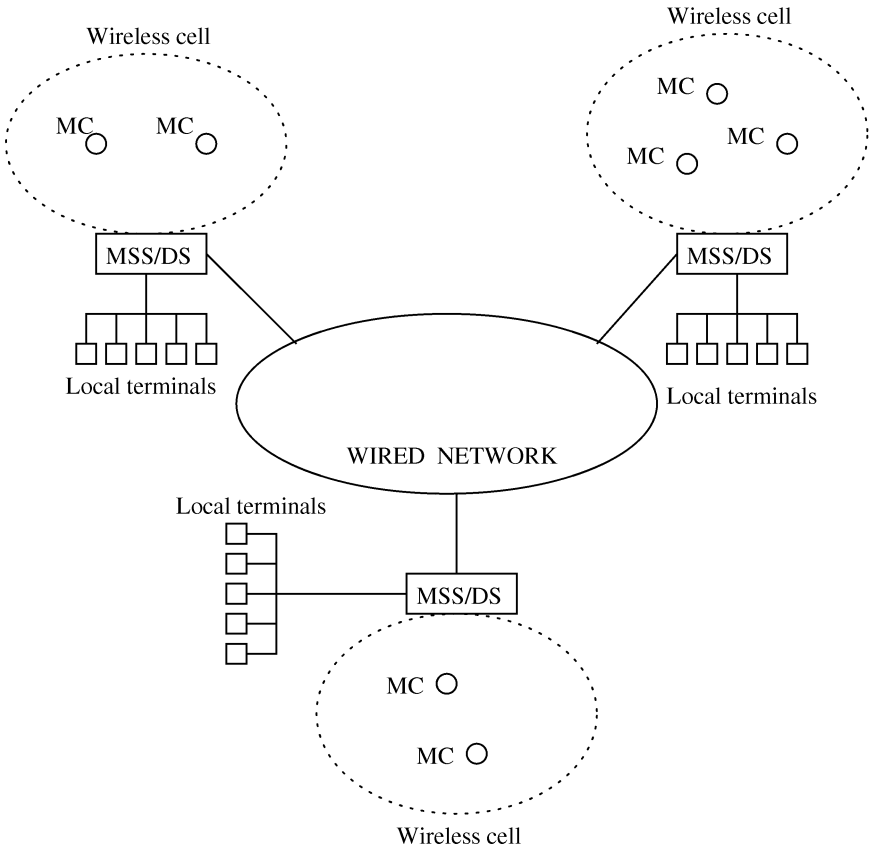

Fig. 1. The wireless computing environment.

work focuses on the effectiveness of the proposed schemes in salvaging the cache content; the issue of minimizing energy consumption is largely ignored. We would like to study how cache invalidation can be made to be more energy efficient.

The rest of this paper is organized as follows: In the next section, we provide some preliminaries. In Section 3, the issues and solutions related to cache invalidation schemes are discussed. As a result, a large set of cache invalidation schemes may be derived. Section 4 presents four representative algorithms obtained from the taxonomy in Section 3. In particular, two of the schemes are existing techniques while the other two are newly proposed. In Section 5, we study the relative performance of the four strategies in terms of their effectiveness in salvaging the cache content and their energy efficiency. Finally, we conclude and discuss some promising future work in Section 6.

2 The Context

The model for a mobile data access system adopted in this paper, as shown in Fig. 1, is similar to that specified in [2]. The mobile environment consists of two distinct sets of entities: a larger number of mobile clients (MC) and relatively fewer, but more powerful, fixed hosts (or database servers) (DS). The fixed hosts are connected through a wired network and may also be serving local terminals. Some of the fixed hosts, called mobile support stations (MSS), are equipped with wireless communication capability. An MC can connect to a server through a wireless communication channel. It can disconnect from the server by operating in a doze mode or a power-off mode. (To conserve energy, hardware vendors have come up with dual-mode processors. One example is the Hobbit Chip, from AT&T, that consumes 250 mW in the full operational *active* mode but only 50 μW in the doze mode [13].) Each MSS can communicate with MCs that are within its radio coverage area called a wireless cell. A wireless cell can either be a cellular connection or a wireless local area network. At any time, an MC can be associated with only one MSS and is considered to be local to that MSS. An MC can directly communicate with an MSS if the mobile client is physically located within the cell serviced by the MSS. An MC can move from one cell to another. The servers manage and service on-demand requests from mobile clients. Based on the requests, the objects are retrieved and sent via the wireless channel to the mobile clients. The wireless channel is logically separated into two subchannels: an uplink channel which is used by clients to submit queries to the server via MSS, and a downlink channel which is used by MSS to pass the answers from the server to the intended clients. We assume that updates only occur at the server and mobile clients only read the data.

To conserve energy and minimize channel contention, each MC caches its frequently accessed objects in its nonvolatile memory such as a hard disk. Thus, after a long disconnection, the content of the cache can still be retrieved. To ensure cache coherency, each server periodically broadcasts invalidation reports. All active mobile clients listen to

the reports and invalidate their cache content accordingly. As in [10], [18], we assume that all queries are batched in a query list and are not processed until the MC has invalidated its cache with the most recent invalidation report. We assume that each server stores a copy of the database and broadcasts the same invalidation reports. In this way, clients moving from one cell to another will not be affected. Thus, it suffices for us to restrict our discussion to just one server and one cell.

There are two metrics that are used to characterize information retrieval in wireless computing environment. The first is the *access time* which is the time elapsed from the moment the client submits a request to the point when all the resultant objects are downloaded by the client. The second deals with the *energy efficiency* of the retrieval mechanisms. There are two measures for this: 1) The *tuning time* is the amount of time the client spent on listening to the channel and 2) the *number of uplink bits transmitted* reflects the amount of energy consumed in transmitting data. Traditionally, once a client submits a query, it listens to the channel until all the resultant objects are received. This leads to the tuning time being equal to the access time. Since listening/transmitting operations requires the CPU to be in full operation (*active mode*), they should be minimized to cut down on power consumption.

3 A Taxonomy of Cache Invalidation Schemes

Two basic categories of cache invalidation strategies have been proposed in the literature [2]. In the first category, the *stateful-based* approach, the server knows the objects that are cached by the mobile clients. As such, whenever there is any update to the database, the server will send invalidation messages to the affected clients. The other category, the *stateless-based* approach, does not require the server to be aware of the state of the clients' cache. Instead, the server broadcasts information on objects that are most recently updated and the clients will listen for and use the reports to invalidate their caches. Because the stateful-based approach is apparently more complex (for example, the server must locate the clients and the clients need to inform the server when they relocate), all reported work in the literature fall into the stateless-based category [2], [4], [3], [10], [15], [18].

For the stateless-based approaches, we can further categorize invalidation methods into either *asynchronous* or *synchronous* methods [2]. Under the asynchronous method, once a record is updated, the server broadcasts updated value immediately. The asynchronous method is effective for connected clients, and allows them to be notified immediately of updates. However, for a client who reconnects after a period of disconnection, the client has no idea of what has been updated and so the entirety of its cache content has to be invalidated. To salvage the cache content, Barbara and Imielinski [2] have proposed that an invalidation report can be piggybacked with each invalidation notice. In this case, upon reconnection, clients will have to wait for the first asynchronous invalidation report. However, since the report is sent asynchronously, there is no guarantee on how long the client must wait.

On the contrary, the synchronous method is based on the periodic broadcasting of invalidation reports. The server keeps track of the records that are recently updated and broadcasts this information to clients periodically. Based on the report, a client determines whether its cache is valid for the query; if it is, it can be used to answer the query, otherwise, the query may have to be submitted to the server. Because of its periodic broadcast nature, synchronous methods provide a bound on the waiting time of the next report.

In this paper, we focus on *synchronous stateless-based* approaches. From existing work, we have identified some common issues that have been addressed in designing cache invalidation schemes. These are the content of the report, the invalidation process, and the information (log) that the server must maintain. Fig. 2 summarizes these issues and the possible solutions.

3.1 On the Content of the Invalidation Report

Ideally, the server should keep track of all updates and broadcast them to the mobile clients. But, this is costly and impractical in view of the limited bandwidth and short battery life of mobile clients. Instead, the server maintains a short (of reasonable length) history of updates and broadcasts an update report (UR) that reflects the most recent changes. Several issues need to be addressed with regards to the content of the report.

3.1.1 Granularity

The *granularity* of the report refers to the level of details of information each record of the report captures. A record in the report can be an (id, TS) pair, where id is the identifier of the object that is updated and TS is the timestamp at which this object is updated. Alternatively, the report can reflect the full detail of the object that is updated at time TS, i.e., the record is the pair (object, TS). The former is commonly known as *update invalidation* as clients can only invalidate their cache content. The latter, on the other hand, allows clients to immediately update their invalid copy with the object that is broadcast. It is thus referred to as an *update propagation* mechanism. Clearly, there is a tradeoff between the two mechanisms. Under update propagation, when the disconnection time is short, clients can update its cache immediately. Under update invalidation, clients must still submit requests to retrieve the updated records even if the disconnection time is short. However, under update propagation, since the entire record is broadcast, the report is much larger and can take up a significant portion of the downlink channel capacity, which is a scarce resource in wireless environment. Moreover, given the same report size, update invalidation can afford to reflect a longer history of updates. Thus, existing works are largely based on update invalidation [2], [4], [3], [15], [18]. Works on update propagation can be found in [4].

Instead of associating each object (or its id) with a timestamp in the report, an alternative is to associate a list of ids/objects with a timestamp. The report would then comprise multiple (list of ids, TS) pairs (or (list of objects, TS) pairs). Each (list of ids/objects, TS) pair means that the list of ids/objects have been updated since TS. For example, if objects O_1, O_2, and O_3 are updated at TS_1, TS_2, and TS_3, respectively, where ($TS_1 < TS_2 < TS_3$), then we may have the record ($O_1, \tilde{}O_2, \tilde{}O_3$, TS_1) in the report. This approach can reduce the number of timestamps needed. However, it

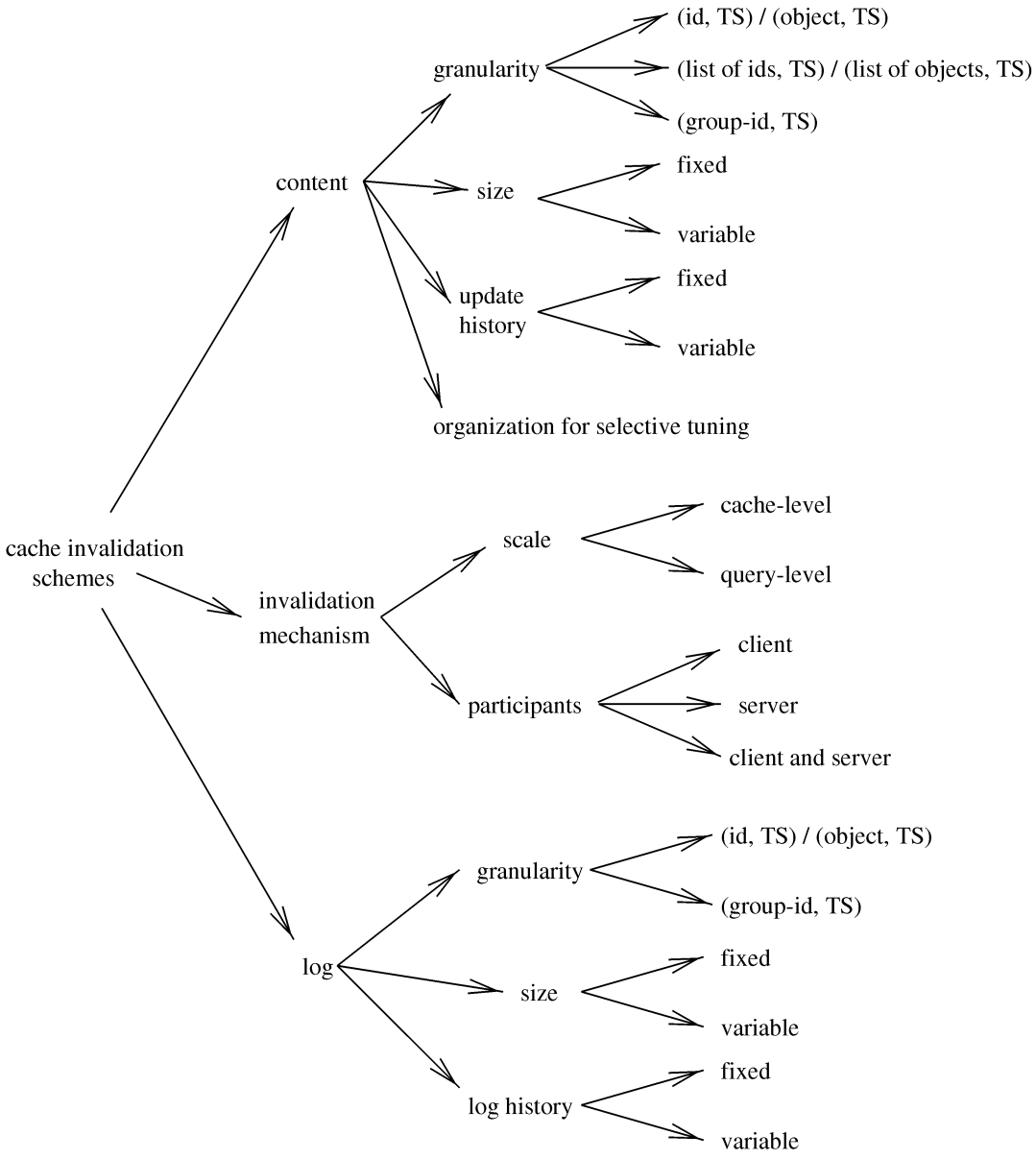

Fig. 2. Issues and solutions in designing cache invalidation schemes.

may lead to false invalidation. For example, if a client accesses the three objects at time $\mathrm{TS_d}$, $\mathrm{TS_2} < \mathrm{TS_d} < \mathrm{TS_3}$, and reconnects at time TS_r, $TS_r > TS_3$, then even though $\mathrm{O_1}$ and $\mathrm{O_2}$ are still valid, they would be invalidated. This approach has been studied in [10].

Another alternative that can minimize the usage of bandwidth for the invalidation report is to disseminate only group-based information. In other words, the database objects are organized into groups. The update report contains (group-id, TS) pairs to reflect the most recent timestamp TS at which an object in the group (identified by group-id) has been updated. Using such a report, the client will invalidate all objects in the group if the group has recently been updated. The main problem with the group-based approach is that of false invalidation, i.e., objects belonging to a group that are valid will still have to be invalidated. Group-based schemes have been studied in [3], [15], [18].

3.1.2 Size and Update History

The size of the invalidation report can be fixed or varied. The update history refers to the history of the updates that are reflected in the report and can be fixed or varied too. These two factors are interrelated in the sense that one typically affects the other. For example, the report size can be fixed by the amount of bandwidth allocated for the report. It can also be fixed by the number of objects/groups to be included in the report. Obviously, under these cases, the update history cannot be predetermined (i.e., it has to be variable) since the number of updates varies over a fixed period of time. On the other hand, the report size can vary from broadcast to broadcast by fixing the update history being reflected. As an example, an update report broadcast, at time T, may reflect all updates during the interval $[\mathrm{T}\text{-}w\mathrm{L}, \mathrm{T}]$ for some $w > 0$ and L; w is referred to as the broadcast window and L is the fixed interval at which the report is periodically broadcast. Works that employ fixed

report size with variable update history have been reported in [3], [10] while works on variable report sizes that fix the update histories have appeared in [2], [3], [15], [18].

3.1.3 Organization for Selective Tuning

To conserve energy, it may be necessary to organize the invalidation report to facilitate selective tuning. This can be done by interleaving the content of the report with "indexes" that can provide "direct" access to the targeted portion of the report. Thus, only the desired portion of the report needs to be examined. Organization of the report to facilitate tuning has been explored in [3]. While there is some work done on indexing broadcast data [12], [9], [16], these schemes cannot be readily applied to our context as invalidation information cannot be predetermined and it varies from report to report. Moreover, these schemes are more complex (e.g., B^{+}-tree).

3.2 On Invalidation Mechanism

There are two issues to address here. The first concerns the scale of the invalidation, whether it is cache-level or query-level. The second concerns the participants that are involved, whether the invalidation is performed by the client only, by the server only, or by a collaboration between the two.

3.2.1 Scale

When a client receives an invalidation report, it can invalidate its cache content in two ways. First, it can perform *cache-level* invalidation, i.e., cache validity is performed for all objects cached. This requires scanning a large portion of the invalidation report, if not the entirety of the report. As a result, it is not particularly suited for selective tuning (unless the number of cached objects is very small). Under this approach, the cache content is associated with one timestamp only—the timestamp of the most recent invalidation report read.

On the other hand, the client can perform *query-level* invalidation, where validation is performed only on the objects queried. This reduces the number of objects to be invalidated and, hence, the report can be organized for selective tuning. However, each cached object has to be associated with a timestamp (compared to a single timestamp for all cached object in cache-level invalidation). The timestamp of an object represents the timestamp at which the object is last known to be valid. This is usually the timestamp of the invalidation report that was last used to validate the object. Thus, different cached objects will have different timestamps. The consequence of this is that each queried object may use a different list of objects for invalidation (recall that different timestamps are associated with a different list of objects). When a query is issued, the query objects' timestamps are checked against that of the invalidation report received. For each object queried, the appropriate list of objects is used to (in)validate it.

Examples of query-level invalidation schemes are SCI and OCI [3], while GCORE [18], Broadcasting Timestamps [2], Bit-sequences [10], and BGI [15] are cache-level oriented.

3.2.2 Participants

Invalidating the cache content can be performed by the client alone. This requires that the client based its invalidation purely on the invalidation reports. Thus, the effectiveness of such approaches are dependent on the content of the report. On the other extreme, we can allow the server to perform the invalidation alone. This, however, will require the client to inform the server about its cache content which can be costly since transmitting this information consumes energy and bandwidth. Finally, the client and server can collaborate to identify the cache content that should be invalidated: The client uses the invalidation report to invalidate its cache content; for those that remain uncertain, the client submits their information to the server for invalidation. Broadcasting Timestamps [2], BGI [15], and SCI [3] are examples of schemes that employ client-only invalidation while GCORE [18] adopts a client-server collaboration approach. To our knowledge, there is no server-only approach.

3.3 On Update Log Structure

Another important issue in the design of a cache invalidation scheme concerns the information (update logs) maintained at the server to reflect the updates on the database.

The update logs may contain update information of each individual object or its identifier. For the former, the log record is of the form (object, TS) to reflect that object has been updated at timestamp TS. For the latter, the server only needs to maintain (id, TS) pairs, each of which indicates that the object with identifier id is updated at TS.

Alternatively, each log record may reflect updates on a collection of objects. In this *group*-based approach, objects are organized into groups and the log record reflects the latest update to the group, i.e., each log record is of the form (group-id, TS) where TS is the most recent timestamp that an object in group identified by group-id has been updated.

The second issue concerns the size and log history (duration that the update logs should be maintained), which, like the content, are interrelated. The size can be fixed by restricting updates to be maintained for a fixed number of objects. In this case, the log history changes depending on the updates. This approach has been adopted in Bit-Sequences [10] where updates to half the database size are maintained.

On the other hand, variable sized logs can be maintained by fixing the log history to a fixed interval, say $[\mathrm{T}\text{-}W\mathrm{L}, \mathrm{T}]$ for some $W > 0$; W is the update log window, L is the fixed interval of broadcasting the invalidation report, and T is the timestamp at which the report is broadcast. In other words, only updates in the interval $[\mathrm{T}\text{-}W\mathrm{L}, \mathrm{T}]$ are maintained. Clearly, keeping all updates is ideal but can be costly in terms of storage, whereas keeping only very recent updates may not be effective as it may lead to false invalidation. The approach of maintaining update history for the last WL time units has been adopted in GCORE [18] Broadcasting Timestamps [2], BGI [15], and SCI [3].

TABLE 1
Summary of Some of the Existing Cache Coherency Schemes

Algorithm	Content	Invalidation Mechanism	Log Structure
Broadcasting Timestamps [?]	(id, TS), size depends on number of updates in fixed update history [T-wL, T], no selective tuning	cache-level, client oriented	(id, TS), log history ([T-wL, T])
Simple Group [?]	same as Broadcasting Timestamps	cache-level, client + server	(id, TS)+(group, TS), log history ([T-wL, T])
GCORE [?]	same as Broadcasting Timestamps	cache-level, client + server	(id, TS)+(group, TS), log history ([T-WL, T])
BGI [?]	(id, TS)+(group, TS), size depends on number of updates in fixed update history [T-wL,T] and fixed group history [T-WL,T] ($W > w$), no selective tuning	cache-level, client	(id, TS)+(group, TS), log history ([T-WL, T])
OCI [?]	(group, TS), fixed size depends on number of groups, update group history based on recent update to group selective tuning on groups	query-level, client	(group, TS), log history (depends on updates on groups)
SCI [?]	(id, TS)+(group, TS) size depends on number of updates in fixed update history [T-wL,T] and fixed group size for updates in [T-WL,T] ($W > w$), selective tuning	query-level, client	(id, TS)+(group, TS), log history ([T-WL, T])
Bit-Sequences [?]	(list of ids, TS), fixed size (database size), variable update history no selective tuning	cache-level, client oriented	(id, TS), log history (variable (intervals, depends on database size)
UP-C [?]	(object, TS), variable size for updates in [T-wL,T], no selective tuning	cache-level, client oriented	(object, TS), log history ([T-WL,T], $W > w$)
UI-CS [?]	(id, TS), variable size for updates in [T-wL,T], no selective tuning	cache-level, client + server	(id, TS), log history ([T-WL,T], $W > w$)

3.4 Cache Invalidation Schemes

Based on the above discussion, we can derive a large number of cache invalidation schemes from the product of the various alternatives in addressing each issue. It should be pointed out that some of the options may not be practical/feasible. For example, it is not possible to have a fixed size report that reflects all updates in a fixed update history [T-wL, T]. On the other hand, we can design techniques that combine multiple solutions. For example, there exists techniques (see Table 1) that reflect both recent updates for individual items as well as group updates. Table 1 summarizes some of the existing work based on the dimensions discussed.

As can be seen from the table, most of the existing techniques are based on update invalidation rather than update propagation. This is to conserve the limited wireless bandwidth. Furthermore, except for the work in [3] (i.e., OCI and SCI), most of the work did not address the issue of selective tuning. In addition, OCI and SCI are the only query-level schemes. We also note that very little work (e.g., GCORE and UI-CS) adopts a client-server collaborative effort for invalidation. Such techniques incur the additional uplink overhead, which can be costly in terms of energy consumption (since transmitting data consumes more

ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
TS	24	16	10	6	22	18	26	32	2	20	14	30	8	4	12	28

Fig. 3. A running example; Database objects and update timestamps.

energy than receiving data). Most of the recent techniques also maintain a longer log history than the update history (i.e., $W > w$) in order to salvage more of the cache content. Finally, we would like to point out that the work in [4] deals with relational operations while the rest focused on object retrievals.

4 Selected Cache Invalidation Schemes

In this section, we describe four cache invalidation schemes designed under the framework proposed in the last section. For all the schemes studied, the client is the one to invalidate the cache, i.e., they all adopt the client-only invalidation mechanism. As such, all the schemes broadcast invalidation reports and clients make use of the reports to invalidate the cache content. Among the four schemes, two of them—Dual-Report Cache Invalidation and Bit-Sequences—are (variations of) existing schemes and do not support selective tuning. The other two schemes—Selective Dual-Report Cache Invalidation and Bit-Sequences with Bit Count—are new algorithms designed to support selective tuning to minimize energy consumption. Throughout this section, we will use the following running example in illustrating the various schemes:

Running Example. We shall use a database of 16 objects as our running example. Fig. 3 shows the database objects, and the timestamp at which each object has been updated. For example, object 5 was updated at time 22 and object 12 was updated at time 30. This example will be used to illustrate how the various cache invalidation schemes studied in this paper (in)validate the content of the client's cache.

4.1 The Dual-Report Cache Invalidation (DRCI)

The Dual-Report Cache Invalidation Scheme (DRCI) is a variation of the BGI [15] scheme. We note that BGI is a broadcast-based version of GCORE [18] so that the server does not need to participate in the invalidation process. BGI has been shown to be as effective as GCORE in salvaging the cache content at a lower energy requirement. DRCI is different from BGI in the way the data is organized into categories. In BGI, there are only two categories while DRCI has four categories.

DRCI is based on the following:

- **Content.** The report consists of a list of (id,TS) pairs and a list of (group-id,TS) pairs. The size and the update history varies. There is no selective tuning.
- **Invalidation Mechanism.** The invalidation is at the cache-level and is performed by the client only.
- **Log.** The server maintains logs for objects. The size of the log varies though the log history is fixed at [T-WL, T] for objects, where T is the current time, w and W are update log windows, and L is the fixed interval at which the reports are broadcast.

Under DRCI, the server broadcasts every L time units a pair of invalidation reports, an *object invalidation report* (OIR), and a *group invalidation report* (GIR). Let the current timestamp be T. To generate the reports, the server keeps track of all the objects updated between interval $T - wL$ and T and between $T - WL$ and T, where w and W are the update log windows and $W > w > 0$. Note that the distinction on the information maintained between the two time intervals is only a logical one. Clearly, objects appearing in the interval $[T - wL, T]$ will also appear in the interval $[T - WL, T]$ and in an actual implementation, such redundancies can be avoided altogether. The most recent OIR broadcast contains the update history of the past w broadcast intervals and is obtained from the information maintained in the interval $[T - wL, T]$. The contents of OIR are the current timestamp T and a list of (o_{id}, t_{id}) pairs, where o_{id} is an object identifier of an object updated during the interval $[T - wL, T]$, t_{id} is the corresponding most recent update timestamp, and $t_{id} > (T - wL)$. The GIR, on the other hand, is a fixed-size report that contains the update history of the past W broadcast intervals and is obtained from the information maintained in the interval $[T - WL, T]$. GIR contains for each group a timestamp that reflects the most recent timestamp in which the group is valid. In other words, the contents of the GIR is a list of (G_{id}, T_{id}) pairs, where G_{id} is a group identifier and T_{id} is the most recent timestamp in which the group G_{id} is valid.

The timestamps of the groups in GIR are determined only at the time when the invalidation reports are to be broadcast and are assigned as follows:

- **Step 1.** Based on the observation that objects that are updated during $[T - wL, T]$ would be reflected in the OIR, we ignore these objects when determining the timestamp of a group.
- **Step 2.** For each group, perform the following: Among the group's remaining objects in the update history, find the one with the largest timestamp that is less than $T - wL$. Let this timestamp be t. If no such objects exists (i.e., no updates during $[T - WL, T]$ or the updates are already included in OIR), then $t = 0$. The timestamp for the group is $\max(T - WL, t)$.

The DRCI scheme facilitates invalidation in two ways. First, for those clients with a small disconnection time, a direct cache checking is performed using the OIR. Second, for those disconnected before the time $T - wL$, OIR and GIR can work together so that the clients need not discard the entirety of its cache—the OIR is used to invalidate individual objects in the cache first, the GIR is then used to invalidate the remaining objects whose groups have been updated. Since the GIR exclude objects in OIR, the OIR

```
//  T – timestamp of current report
//  T^c – timestamp of last valid report received by mobile client
for every pair [o_id, t_id] ∈ OIR {
    if (object o_id is in the client cache) {
        if (T^c < t_id )
            remove object from the cache
    }
}
if (T − T^c > wL) {
    for every group G_id in the client cache {
        tune to pair [G_id, T_id] in GIR
        if (T^c < T_id)
            remove all objects in group from the cache
    }
}
for every object O ∈ Q_i {
    if (O is in the cache)
        use the cache's content to answer the query
    else
        submit request for O
}
T^c ← T
```

Fig. 4. The Dual-Report Cache Invalidation scheme.

effectively helps to minimize false invalidation. The GIR can further minimize invalidation of the entire cache for long disconnection time. The algorithmic description of the protocol is shown in Fig. 4. Note that the entire cache content will be discarded if the client disconnection time is longer than $T - WL$.

Although the basic DRCI scheme is independent of the way the database objects are grouped, its performance can be influenced by the grouping scheme. We observe that very often only a small set of data objects are hot in demand and, likewise, only a small set of data objects are hot in terms of frequency of update. As such, we can divide the whole database into four data sets: hot update-hot demand (HH), hot update-cold demand (HC), cold update-hot demand (CH), and cold update-cold demand (CC). Clearly, it makes no sense to group objects in the HH category (hot update-hot demand) with objects in the CC category (cold update-cold demand). Therefore, groups are formed from objects in the same category.

Example 1. We shall illustrate the DRCI scheme using our running example. Let $T = 34$ be the timestamp where the most recent pair of invalidation reports are broadcast, $L = 4$, $w = 2$, and $W = 6$. Suppose the objects are initially split into four groups:

$$G1 = \{o1, o2, o3, o4\},$$
$$G2 = \{o5, o6, o7, o8\},$$
$$G3 = \{o9, o10, o11, o12\},$$
$$G4 = \{o13, o14, o15, o16\}.$$

At time T, the update reports are as follows:

- The contents of OIR are as follows: $\{34, (o7, 26), (o8, 32), (o12, 30), (o16, 28)\}$
- The contents of GIR for groups G1, G2, G3 and G4 at time 34 are $\{(G1, 24), (G2, 22), (G3, 20), (G4, 12)\}$

Suppose a mobile client, MC, disconnects at time 23 (last set of reports received at time 22) and reconnects at time 34. Suppose MC generates a query that requests objects $o1, o2, o6, o7, o9, o12, o14$ that are in its cache. From the OIR, MC invalidates $o7$ and $o12$. Thus, these objects can be "removed" from the cache and the remaining objects, whose validity are unknown, are given by the resultant groups:

$$G1 = \{o1, o2\},$$
$$G2 = \{o6\},$$
$$G3 = \{o9\},$$
$$G4 = \{o14\}.$$

Since groups G2, G3, and G4 were updated before time 23, therefore, the remaining objects in these groups are still valid and, hence, will not be invalidated. However, all the objects in G1 are considered invalid as there exists an update after time 23 in the group.

4.2 The Bit-Sequences Scheme (BS)

The Bit-Sequences algorithm [10] adopts the following framework:

- **Content.** The report consists of a list of (list of ids, TS) pairs in a compact form. This allows the report size to be fixed, though the update history varies. There is no organization to support selective tuning.
- **Invalidation Mechanism.** The invalidation is performed by the client at cache-level.
- **Log.** The logs maintained at the server keeps track of individual object update information (using (id,TS) pairs) for up to half the database size, i.e., the size of the log is fixed but the update history is variable.

Let the number of database objects be $N = 2^n$. In the BS algorithm, the invalidation report reflects updates for n different times—$T_n, T_{n-1}, \ldots, T_1$, where $T_i < T_{i-1}$ for $1 < i \leq n$. The report comprises n binary bit-sequences, each of which is associated with a timestamp. Each bit represents a data object in the database. A "1" bit in a sequence means that the data object represented by the bit has been updated since the time specified by the timestamp of the sequence. A "0" bit means that the object has not been updated since that time. The n bit-sequences are organized as a hierarchical structure with the highest level (i.e., bit-sequence B_n) having as many bits as the number of objects in the database and the lowest level (i.e., bit-sequence B_1) having only two bits. For the sequence B_n, as many as half of the N bits (i.e., $N/2$) can be set to "1" to indicate the $N/2$ objects that have been updated (initially, the number of "1" bits may be less than $N/2$). The timestamp of the sequence B_n is T_n. The next sequence in the structure, B_{n-1}, has $N/2$ bits. The kth bit in B_{n-1} corresponds to the kth "1" bit in B_n, and $N/2^2$ bits can be set to "1" to indicate that $N/2^2$ objects have been updated since T_{n-1}. In general, for sequence B_{n-i}, $0 \leq i \leq n-1$, there are $N/2^i$ bits and the sequence will reflect that $N/2^{i+1}$ objects have been updated after the timestamp T_{n-i}. The kth bit in sequence B_{n-i} corresponds to the kth "1" bit in the preceding sequence (i.e., B_{n-i+1}). An

```
//  T – timestamp of current report
//  T^c – timestamp of last valid report received by mobile client
if T_0 ≤ T^c
    all cached objects are valid
else {
    if T^c < T_n
        remove the entire cache content
    else {
        determine the bit sequence B_i such that T_i ≤ T^c < T_{i-1}, 1 ≤ i ≤ n
        invalidate all the objects marked "1" in B_i
    }
}
for every object O ∈ Q_i {
    if (O is in the cache)
        use the cache's content to answer the query
    else
        submit request for O
}
T^c ← T
```

Fig. 5. The Bit-Sequences scheme.

additional dummy sequence B_0, with timestamp T_0, is used to indicate that no object has been updated after T_0.

In general, N does not need to be a power of two and the number of lists can also be any value other than n. Furthermore, the list associated with timestamp T_i does not need to reflect the updates for half the number of objects in the list associated with T_{i+1}, $1 \leq i \leq n-1$. However, for simplicity, we have chosen to follow the approach proposed in [10].

The Bit-Sequences structure is broadcast to clients periodically. The protocol for invalidating the cache is shown in Fig. 5.

Example 2. Fig. 6 shows the invalidation report represented by the Bit-Sequences structure for our running example. As shown, the first level (B_4) has 16 bits, eight of which have been set to "1". These eight objects are the most recently updated objects. The timestamp for B_4 is 18. Similarly, bit sequence B_1 has two bits, reflects the most recently updated object 8, and has a timestamp of 32.

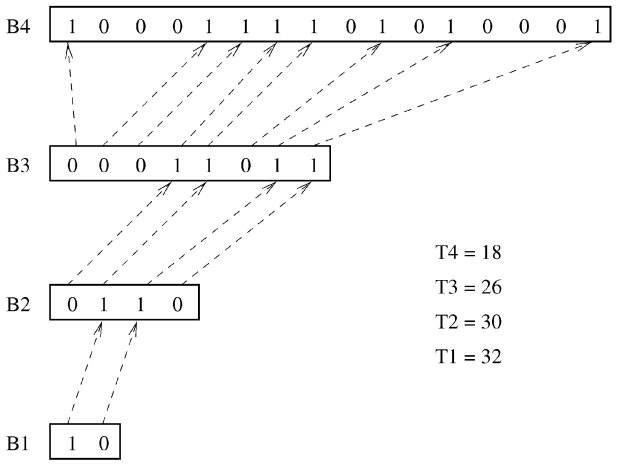

Fig. 6. The Bit-Sequences structure.

Assume that the client receives this invalidation report when it submits its query for objects 5 and 8. Suppose the last invalidation report received by the client before it disconnects is at time 31. Since the client's cache content is last valid at time 31, it should use the sequence B_2 to invalidate its cache. To locate those objects denoted by the two "1" bits, the client will check the sequences B_2—B_4. This is accomplished as follows: To locate the object corresponding to the second bit that is set to "1" in B_2, the client has to check the second "1" bit in B_3. Since the second "1" bit in B_3 is in the fifth position, the client will have to examine the 5th "1" bit in B_4. Because B_4 is the highest bit-sequence and the 5th "1" bit is in the eighth position, the client can conclude that object 8 has been updated since time 31. Similarly, the client can determine that the 12th object has also been updated since that time. Therefore, both objects will be invalidated by the client. Since the client requests for objects 5 and 8, object 5 remains valid and can be used to answer the query while the request for the invalid object 8 has to be submitted to the server.

4.3 Selective Dual-Report Cache Invalidation (SDCI)

Schemes DRCI and BS have been separately shown to be effective in salvaging the cache content (though a comparative study has not been performed). However, they are both not energy efficient as there is no mechanism to support selective tuning of the invalidation report and the invalidation is cache-based. In other words, the entirety of the report has to be downloaded by the clients and every object in the report examined. The next two algorithms that we shall see will address this shortcoming.

The first scheme, the Selective Dual-Report Cache Invalidation scheme (SDCI), is essentially the same as DRSI except that the report is organized to facilitate selective tuning and the invalidation is query-level-based. Unlike DRCI, SDCI organizes the pair of invalidation reports (the

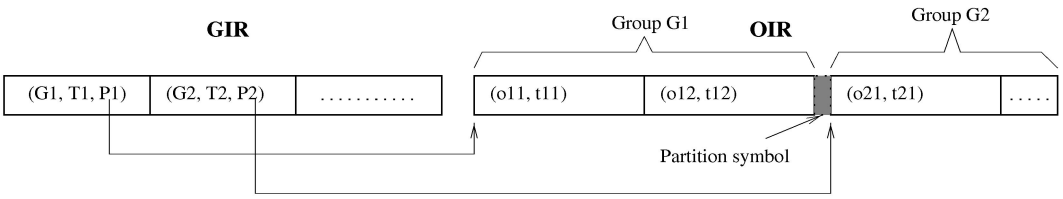

Fig. 7. Organization of the invalidation reports in SDCI.

OIR and GIR) as follows: First, the GIR is broadcast before the OIR. Second, the entries in the OIR are ordered and broadcast based on the groups, i.e., updates in the same group will appear together. A partition symbol will separate continuous groups. Third, an additional "pointer" is added to each element of the GIR and this pointer reflects the starting position of the objects within this group in the OIR. Thus, the GIR consists of triplets of the form (group-id, TS, ptr) where group-id represents the group identifier, TS is the timestamp of the most recent update (excluding those in the OIR) of the group, and ptr is an offset to the starting position of the objects in OIR corresponding to the group identified by group-id. Fig. 7 shows the organization of the invalidation reports.

As noted, the invalidation process is query-level-based. Thus, at the client, it operates as follows: For each group queried, it first selectively tunes to the GIR and keeps the pointers of interested groups (i.e., those in query) in memory. Once all the desired groups are determined, it selectively tunes to the respective position in the OIR using the pointers. Thus, only the desired groups and objects are examined. The functionality of the improved algorithm SDCI is as shown in Fig. 8.

4.4 Bit-Sequences with Bit Count

The second energy efficient scheme that we will look at is the Bit-Sequences with Bit Count (BB) method. Like the Bit-Sequences approach, it comprises a set of bit sequences organized in a hierarchical manner. However, only the relevant bits need to be examined. This is achieved by associating each bit sequence with a bit count array.

Let N be the number of objects in the database. Furthermore, let b_t denote the size of a timestamp. We also assume that a query Q returns the set of objects $\{O_1, O_2, \ldots, O_q\}$ as answers. Furthermore, we assume that the objects are already ordered in the same manner as the information reflected in the invalidation report, i.e., information for O_1 will be received before information for O_2 and so on. If the objects are not ordered accordingly, then they can be sorted. For simplicity, we ignore this sorting phase in our discussion. We also denote the corresponding timestamps when these objects are last valid (in the client cache) as $t_1, t_2, \ldots, t_q$, respectively.

As in the BS scheme, the BB structure comprises a set of n bit sequences: Sequence B_n has a timestamp T_n that indicates that updates after T_n are reflected and comprises N bits, half of which are set to "1"; sequence B_{n-1} has timestamp T_{n-1} and $N/2$ bits, of which $N/2^2$ bits are set to "1" and so on. In fact, the content of the bit sequences are exactly the same as those of the BS scheme. Like the BS scheme, if the bit sequence B_{n-i} is to be used to invalidate the cache, then the sequences $B_{n-i}, B_{n-i+1}, \ldots, B_n$ *may* have to be examined. However, the proposed BB strategy adopts a *top-down* examination of the sequences, i.e., from B_n to B_{n-i}, rather than the *bottom-up* approach (i.e., B_{n-i} to B_n) of BS scheme. Moreover, for some valid objects, it may not be necessary to examine all the sequences from B_n to B_{n-i} as it may be possible to determine their validity and terminate the search before sequence B_{n-i}. Furthermore, the proposed scheme only examines the relevant bits in each sequence. As the kth "1" bit in B_{n-i} corresponds to the kth bit in B_{n-i-1}, we need a mechanism that can count the number of "1" bits in a sequence, say B_{n-i}, without examining the entire sequence. Before we discuss this mechanism, let us illustrate how selective tuning can be facilitated with such a mechanism. To validate an object O, the client first identifies the bit

```
// T_id^c – last known timestamp of group with id G_id
// t_id^c – last known valid time of object with id o_id
P ← φ
for (every group queried) {
        selectively tune to the triplet [G_id, T_id, P_id] in GIR_i
        if T_id^c < T_id
            invalidate all objects of group
        else {
            T_id^c ← T_id
            P ← P ∪ {P_id}
        }
}
while (P ≠ φ) {
        p ← first(P)
        P ← P − {p}
        switch to doze mode until the position p is coming
        while (not encounter partition symbol) {
            download pair [o_id, t_id]
            if (o_id is in client cache) {
                if t_id^c < t_id
                    invalidate object
                else
                    t_id^c ← t_id
            }
        }
}
for (every object O ∈ Q_i ) {
        if (O is in the cache)
            use the cache's content to answer the query
        else
            submit uplink request for O
}
```

Fig. 8. The Selective Dual-Report Cache Invalidation scheme.

```
// AnswerSet = {o_1, o_2, ..., o_q} — objects for query Q
// t_id – last known valid time of object with id o_id
download the counter, the timestamps and the bit count arrays
for each object o_i ∈ AnswerSet
        if T_O ≤ t_i // object o_i is valid
            AnswerSet = AnswerSet - {o_i}
        else if t_i < T_n // object o_i is invalid
            AnswerSet = AnswerSet - {o_i}
// AnswerSet contains the remaining objects whose validity is still uncertain
for each object o_i ∈ AnswerSet
        determine the bit sequence to be used to validate o_i
k = n
repeat {
        for each object o_i ∈ AnswerSet, examine bit sequence B_k {
            tune to packet p containing information on o_i
            examine the bits in packet p until position of o_i
            let the number of "1" bits set (in p inclusive of o_i) be b
            if B_k is the bit sequence to be used to validate o_i {
                if the bit corresponding to o_i is set to "1"
                    o_i is invalid
                else
                    o_i is valid
                AnswerSet = AnswerSet - {o_i}
            } else {
                if the bit corresponding to o_i is set to "0" {
                    o_i is valid
                    AnswerSet = AnswerSet - {o_i}
                } else {
                    // we need to examine the next sequence B_{k-1}
                    from the bit count array of B_k
                        determine the number of "1" bit from packet 1 to p − 1 of B_k
                    let this value be c
                    the position of o_i in bit sequence B_{k-1} is (c + b)
                }
            }
        }
        k = k − 1
} until AnswerSet = ∅
```

Fig. 9. The Bit-Sequences with Bit Count scheme.

sequence that should be used. This is accomplished by examining the set of timestamps. Suppose this is sequence B_{n-i}. This means that we need to examine sequence B_n, followed by B_{n-1}, and so on until B_{n-i}. From object O, the client can selectively tune to the corresponding bit in B_n (without scanning the entire B_n). If the bit is set to "0", then the object is valid (since object O will not be found in any subsequent sequences B_{n-1}, B_{n-2}, ...); otherwise, the client determines (we shall discuss shortly how this can be done) the number of "1" bits from the beginning of B_n to the bit corresponding to O. From this number, it can again selectively tune to B_{n-1} and examine the corresponding bit of O in B_{n-1}. Again, if the bit is "0", then the object O is valid and the search terminates; otherwise, its position in the B_{n-2} is determined and this process is repeated until sequence B_{n-i}. We can terminate when we encounter "0" bit at any of the sequences from B_n to B_{n-i}. If the relevant bit at B_{n-i} is "1", then the object is invalid; otherwise, it is valid.

Now, the mechanism to facilitate selective tuning is simple. We associate with each bit sequence a bit count array, all of which have entries that are j bits. For bit sequence B_{n-i}, $0 \leq i \leq n-1$, the sequence is partitioned into packets of 2^j bits. In other words, there are $\lceil (N/2^i)/2^j \rceil$ packets. In general, for sequence B_{n-i}, the number of array entries is $\lceil (N/2^i)/2^j \rceil$. (We note that sequences with fewer than j bits do not need to be associated with a bit count array as all bits have to be examined anyway.) Essentially, the kth entry in the bit count array of sequence B_{n-i} represents the number of "1" bits that have been set for the kth packet in the sequence. Selective tuning is achieved as follows: We know for which bit/position in a sequence we should be looking. Let packet i contain the bit in which we are interested. From the bit array count, we can determine the number of "1" bits that have been set for packets 1 to $i-1$. The client can then tune into the ith packet and scan the ith packet until the relevant bit. In this way, we will be able to compute the number of "1" bits!

To check the validity of the answer objects to query Q, the client employs the protocol shown in Fig. 9.

The invalidation report is organized as follows: The counter is broadcast first, the timestamps are broadcast

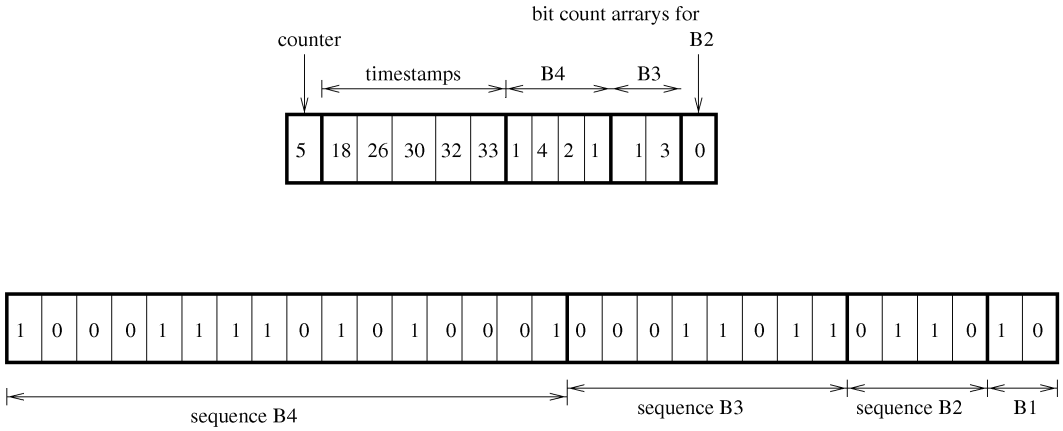

Fig. 10. The Bit-Sequences with Bit Count method.

next, followed by the bit count arrays for sequences $B_n, B_{n-1}, \ldots$, and, finally, the bit sequences $B_n, B_{n-1}, \ldots, B_1$.

Example 3. We illustrate scheme BB with the same running example in Fig. 3. The organization of the invalidation report for BB is shown in Fig. 10. In this example, each bit count array entry keeps track of the number of "1" bits set for four objects. Since there are 16 objects in the database, there are four entries in the bit count array corresponding to B_4, two entries in B_3's bit count array, and one entry in B_2's bit count array. We note that B_1 is not associated with a bit count array. As before, assume that a query requests for objects 5 and 8 whose cached timestamps are, respectively, 31 and 27. From the timestamps in the invalidation report, the client knows that it needs to check B_2 for the validity of object 5, and B_3 for the validity of object 8. The client first determines which two bits in B_3 correspond to the two queried objects. This is done as follows: As both objects 5 and 8 are in the same packet, from the first bit count array entry (of B_4), the client knows that there is only one "1" bit among the first four objects in the bit-sequence B_4. Thus, it will tune to the beginning of the second packet of B_4 and examine the first bit (in the second packet) till the fourth bit. Since the first bit corresponds to object 5 and it is set to "1", the client knows that object 5 is the second bit in B_3. Similarly, the client can determine that object 8 is the fifth bit in B_3. For object 5, the client examines the corresponding bit in B_3 which has been set to "0" indicating that the object is valid. For object 8, the client first examines the bit count array for B_3 and knows that the first entry contains a value of 1. By examining the first bit of the second packet of B_3, it determines that the bit corresponding to object 8 is set to "1". This also means that object 8 can be found in the second bit of B_2. It then examines the second bit of B_2 and notes that object 8 is invalid.

5 A Performance Study

We conducted extensive studies based on a simulation model. Four schemes are studied in this simulation: Dual-Report Cache Invalidation (DRCI) and Bit-Sequences (BS) and their energy-efficient counterparts Selective Dual-Report Cache Invalidation (SDCI) and Bit-Sequences with Bit-Count (BB). Earlier work has shown that BGI, a variation of DRCI, is more energy efficient than GCORE without sacrificing the access time performance, so we did not evaluate the performance of GCORE here [15]. The schemes are evaluated based on two metrics, the access time and energy consumption. The access time also serves to indicate the utilization of wireless bandwidth. The energy consumption contains three components: The energy consumed on cache invalidation, the energy consumed on uplink requests and the energy consumed to download the desired data. In our study, these three components are measured by the energy expended to examine the invalidation report, the energy consumed to transmit queries on objects that are not valid or in the cache, and the energy to receive the objects. Furthermore, the energy expended on these components is expected to be proportional to the amount of data received/transmitted. Thus, for simplicity, we use the amount of data received/transmitted as an indication of the energy consumption. The access time is calculated in seconds while energy is measured in units of data received/transmitted. A unit is defined as the energy consumed on receiving 1K-bits of information and, based on the findings in [5], transmitting data is assumed to be 10 times more power consuming than receiving data.

5.1 The Simulation Model

Fig. 11 shows our simulation model. The model comprises several modules: *clients*, *uplink channel*, *downlink channel*, and *server*. Clients submit requests to the server via the uplink channel and receive results from the server via the downlink channel. When a client reconnects, it employs an invalidation scheme to invalidate its cache content and submits query objects accordingly. In order to focus on the cache invalidation effect, we assume that a cache miss is only resulted from invalidation. In other words, we assume that all the queries in a mobile computer reference a fixed subset of objects that are initially cached in its local storage. The model has been implemented using a C-based simulation package [17].

Table 2 shows the notations and default parameter settings used in the simulation. The database has a total of

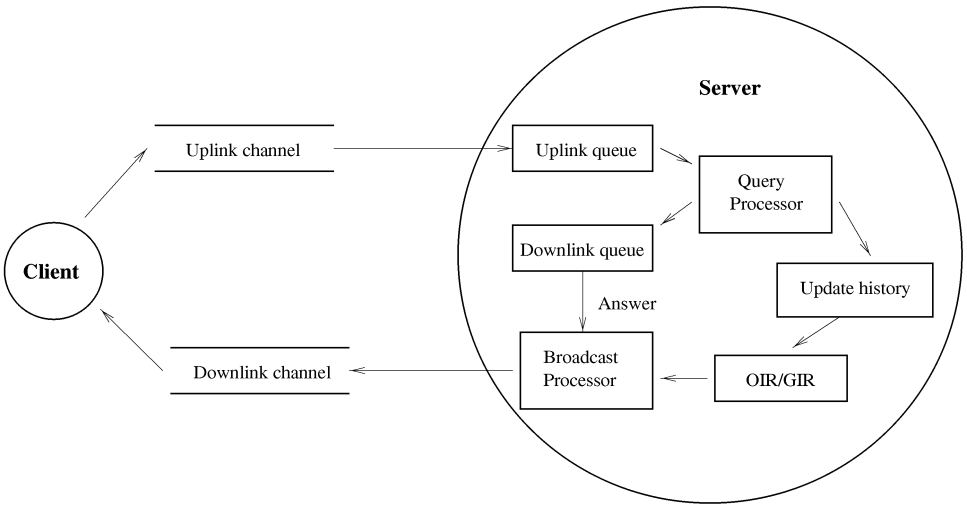

Fig. 11. Mobile communication model.

D objects. The size of each object is O bits, with O_{id} bits being used for the object id. Each timestamp is T_{id} bits. β_1 percent of the data objects are in the hot update set while $(100 - \beta_1)$ percent of them belong to the cold update set. β_2 percent of the data objects are in the hot demand set while $(100 - \beta_2)$ percent of them belong to the cold demand set. Data in the hot update and hot demand sets are randomly chosen from the D objects. α_1 percent of the updates are focused on the hot update set while the remaining $(100 - \alpha_1)$ percent are on the cold update set. The number of objects requested per query is uniformly distributed in the interval $[Q/2, 3Q/2]$. The objects are picked such that α_2 percent of them are from the hot demand set and the rest are from the cold demand set. The query arrival rate follows a Poisson distribution with a mean of λ_q while the update arrival rate follows a Poisson distribution with a mean of λ_u. Clients disconnection time is assumed to follow a negative exponential distribution

TABLE 2
System and Workload Parameters

Notation	Definition	Default Values
General Parameters		
D	Server database size	100,000 obj
λ_q	mean arrival time	0.5 sec
λ_u	mean update time	0.5 sec
Q	Mean objects referenced by a query, which has a uniform distribution with low Q/2 and high 3Q/2	30 obj
β_1	% of data objects in the hot update set	10
β_2	% of data objects in the hot demand set	10
α_1	% of updates on hot update set	90
α_2	% of demands on hot demand set	90
ν	mean disconnection time	1000 sec
C_{up}	Bandwidth of uplink channel	19.2 kbps
C_{down}	Bandwidth of downlink channel	100 kbps
L	Periodic broadcast interval	20 sec
O	Object size	4096 bits
O_{id}	Object id size	32 bits
T_{id}	Timestamp size	64 bits
P	Link size	16 bits
Dual-Report Approaches		
G	Group size	100 obj
N	Total number of groups	$\approx D/G$
w	Broadcast window	10
G_{id}	Group id size	16 bits
s	Partition symbol (for SDCI only)	8 bits
Bit-Sequences Approaches		
U_{id}	Unique number size	32 bits

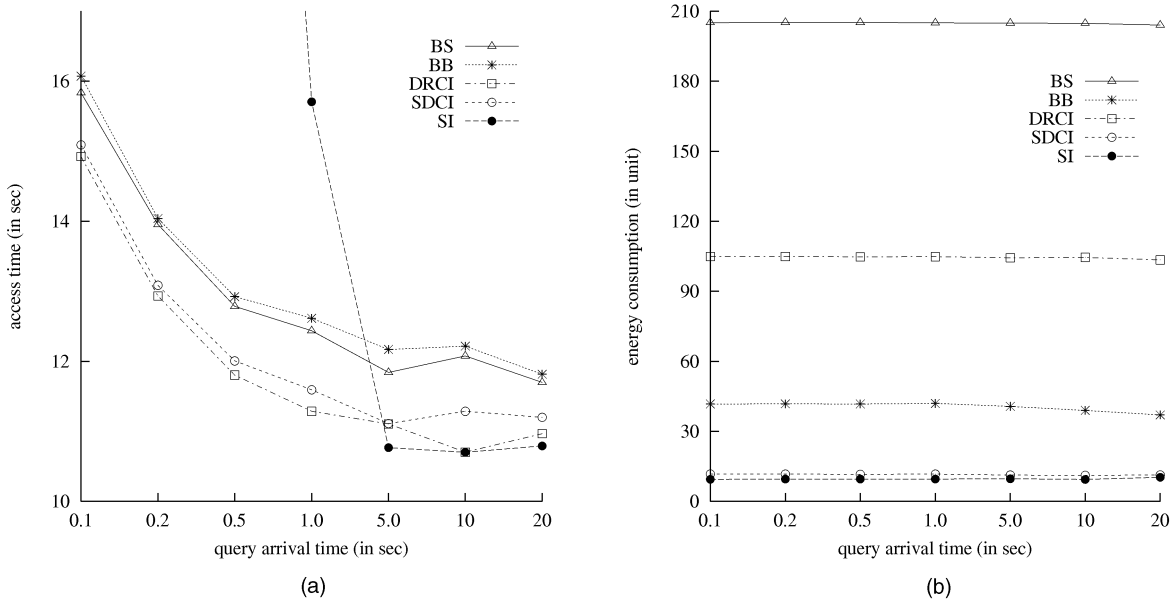

Fig. 12. Effect of query arrival time. (a) Access time. (b) Energy consumption.

with a mean of ν. The update reports are broadcast every L seconds. Finally, the bandwidth for the downlink and uplink channels are given by C_{down} and C_{up}, respectively. For schemes that support selective tuning, there is an additional link information of P bits. Most of the settings used are similar to those used in previous work [3], [15], [18].

For the dual-report-based approaches, the database is further organized into N groups, each with G objects. Each group has an additional group id information that takes G_{id} bits. The update broadcast window is given by w. For scheme SDCI, the separators between groups are each s bits.

For the bit-sequences approaches, each object is also assigned a unique number (while the unique value can be used as the object id, we opt to treat them as two separate attributes) that corresponds to its position in the bit sequence. The unique number is U_{id} bits.

5.2 Effect of Query Interarrival Time

In this experiment, we study the number of concurrent users that can be supported (i.e., the scalability). This is important as the larger the number of queries the environment can support, the better the scheme is. We vary the query arrival time from 0.1 to 20 seconds. As reference, we also modeled a simple straightforward *server-only* invalidation strategy where the clients query the server for the validity of all objects touched by the query. We shall refer to this scheme as the *server invalidation* scheme (denoted SI). The results of this study is shown in Fig. 12.

We shall first look at the results of the four schemes: BB, BS, DRCI, and SDCI. For the four schemes, the access time decreases with increasing query arrival time. This follows from the fact that with larger arrival time, the number of concurrent queries in the system is smaller and the bandwidth is sufficient to handle these queries. However, when the query arrival time is small, the downlink channel becomes congested as more queries are issued. The contention for the limited bandwidth leads to the higher access time. We also note that the energy-efficient schemes are slightly worse than their counterparts that do not facilitate selective tuning, i.e., BB is worse than BS and SDCI is worse than DRCI. As BB and BS reflects the same update information in the invalidation reports, this result is expected by virtue of the fact that the report size of the BB scheme is larger. The same reason applies to schemes SDCI and DRCI. We also observe that the Bit-Sequences methods are worse than the Dual-Report approaches. This is because the report size of the Bit-Sequences methods is larger than that of the Dual-Report approaches, thus consuming more bandwidth (for this experiment, the report size of the Bit-Sequences methods is more than twice as large).

In terms of energy consumption, we note that all the schemes are largely unaffected by query arrival time. This is so since the energy consumption comes from examining the invalidation report, submitting the queries and receiving the results of invalidated objects, and all these components are independent of the query arrival time. As expected, the energy efficient schemes outperform their counterparts by a wide margin (more than 10 times more energy efficient). Since the BB scheme has to examine the bit count arrays, it consumes more energy than the SDCI scheme.

From the access time result, we note that scheme SI performs best when the interarrival time is large. This is because the system load is light. More importantly, all the other four approaches have a smaller effective downlink bandwidth (part of the bandwidth is used to broadcast the invalidation reports) compared to scheme SI. However, when the interarrival time becomes small, scheme SI's need to submit the requested objects' information results in the server and the uplink channel becoming a bottleneck very quickly. The other approaches, on the other hand, only submit invalid objects' information. In terms of energy consumption, we note that scheme SI's energy consumption is fairly good. In fact, it is the best among all the schemes. While it incurs more energy in transmitting objects'

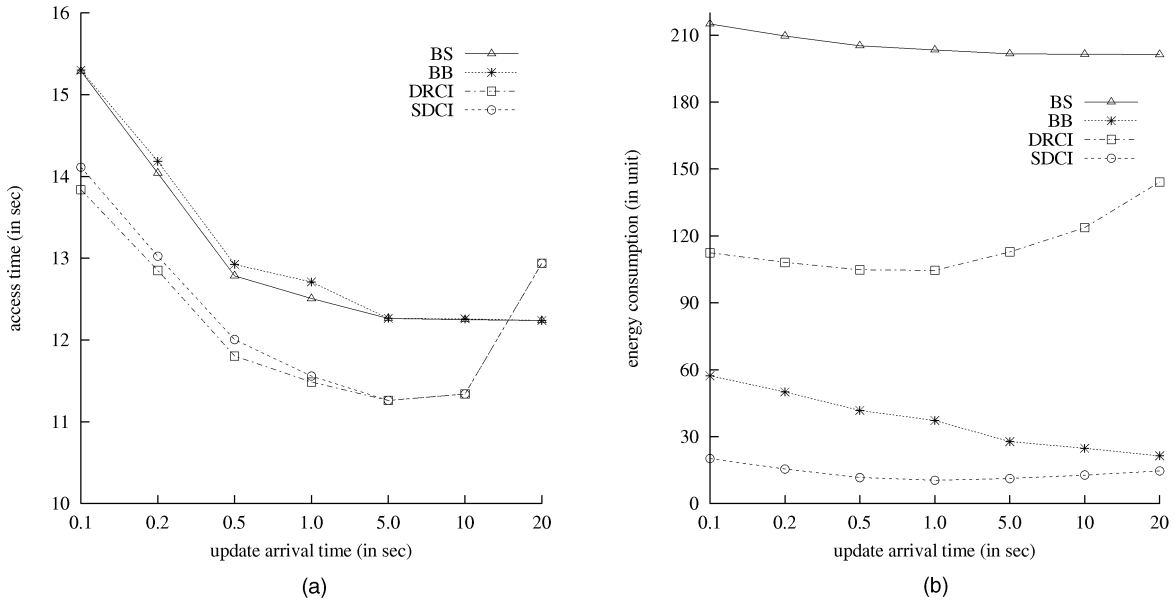

Fig. 13. Effect of update arrival time. (a) Access time. (b) Energy consumption.

information, it does not need to examine invalidation reports. The net effect in this experiment is an overall better performance.

This paper focuses on the broadcast-based schemes and, since scheme SI is not expected to scale well, we shall not study it further here. However, we would add that both methods may be deployed in practice—broadcast-based approaches to be used for less frequently updated objects while the method SI is used for the remaining objects.

5.3 Effect of Update Arrival Time

In this experiment, we study the effect of update arrival time by varying the update arrival time from 0.1 to 20 seconds. Fig. 13 shows the effect of update time on access time and energy consumption. First, we observe that the relative performance between BS and BB and DRCI and SDCI, is similar to those results presented earlier, i.e., BS and DRCI are respectively slightly better than BB and SDCI in terms of access time, but are respectively less energy efficient than their counterparts in terms of energy consumption.

Second, for the Bit-Sequences-based approaches, we note that as the update time increases, the access time and energy consumption also reduce. A shorter update time means more objects being updated in a shorter timespan. As a result, the difference between timestamps in the invalidation report is small. This implies that updated objects associated with a smaller timestamp value (those further away from the current time) are the ones that the client is likely to be looking out for. Recall that in the invalidation report, a smaller timestamp would be associated with a larger number of updated objects. Thus, more objects are being invalidated and the client needs to refresh more objects. This explains why both the access time and energy consumption are higher. On the other hand, a longer update time means that the difference between timestamps in the report is large and, so, fewer objects are likely to be invalidated. Thus, the access time and energy consumption are smaller compared to that of shorter update time.

Third, for the Dual-Report-based approaches, we see a slightly different picture. There exists a certain update rate at which the system performs best, i.e., too low or too high an update rate can degenerate the system performance. A high update rate implies a high probability of cache invalidation and, also, a larger update history for a fixed w. This results in a large OIR that consumes the downlink bandwidth and, hence, the poor performance. On the other hand, when the update rate is low, the probability of false invalidation increases as the updated object may no longer appear in the OIR but reflected in the GIR.

Finally, in terms of energy consumption, we note that the relative performance between the four schemes remains largely the same. For the Dual-Report schemes, we note that there is also an "optimal" point. This follows from the same reasons as the case of the access time performance.

5.4 Effect of Disconnection Time

A user's behavior is difficult to predict especially in a mobile environment. A mobile client may or may not disconnect from the wireless network and, if it should, it may disconnect with a different duration each time. As we mentioned before, disconnection is one of the most important factors that complicates the problem of cache invalidation. Thus, for a scheme to be practical and useful, it should be robust to the clients' disconnection behaviors. In this experiment, we study how disconnection time will affect the performance of the four schemes. Fig. 14 shows the results.

As shown in Fig. 14a, all the schemes are largely unaffected by disconnection time of less than 1,000 seconds. Even for disconnection time of 10,000 seconds, the schemes degrade no more than 30 percent. The degradation is expected since a longer disconnection time implies more objects being invalidated. As in earlier experiments, BB is inferior to BS and SDCI is worse than DRCI. Besides the larger report size of the Bit-Sequences methods compared to the Dual-Report approaches, there is another reason: When the disconnection time is small, the OIR of the Dual-Report

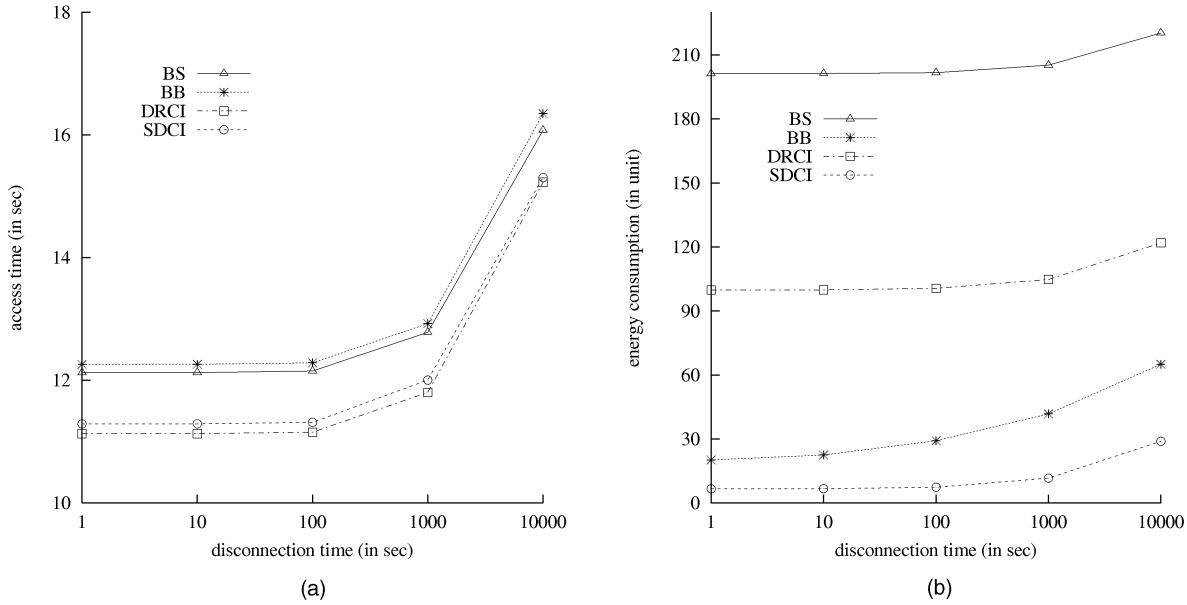

Fig. 14. Effect of disconnection time. (a) Access time. (b) Energy consumption.

methods can be used to invalidate stale records and false invalidations are minimal; when the disconnection time is high, all methods have about the same amount of invalidations.

Fig. 14b shows the energy consumption of the various schemes. As shown, all the schemes' energy consumption increases with disconnection time. This is because as the disconnection time lengthens, the number of invalidated objects increases, which means that more energy has to be expended to refresh these objects. As expected, the energy efficient schemes outperform their counterparts by a wide margin and the BB scheme is less energy efficient than the SDCI scheme.

5.5 Effect of Update Skew

Another factor that can affect the effectiveness of the caching schemes is the update skew, i.e., how the updates are directed at the database objects. In the previous experiments, we have used the default of 90 percent of the updates directed at 10 percent of the database objects. In this experiment, we shall see the effect of update skew on our schemes. For simplicity, we shall restrict our study to update skew of the form $\alpha - \beta$, where α percent of the updates are directed at β percent of the database and $\alpha + \beta = 100\%$. We experimented with α values from 50 percent to 95 percent. Intuitively, as α increases, a cache invalidation scheme should improve in performance since fewer data are being invalidated. This is confirmed in

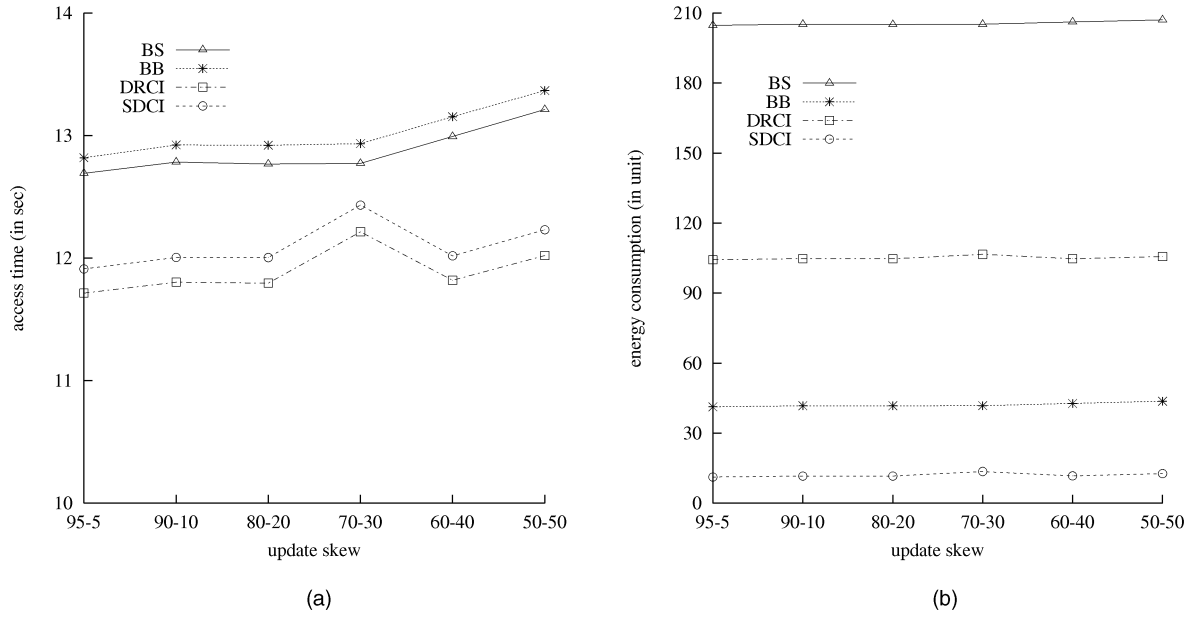

Fig. 15. Effect of update skew. (a) Access time. (b) Energy consumption.

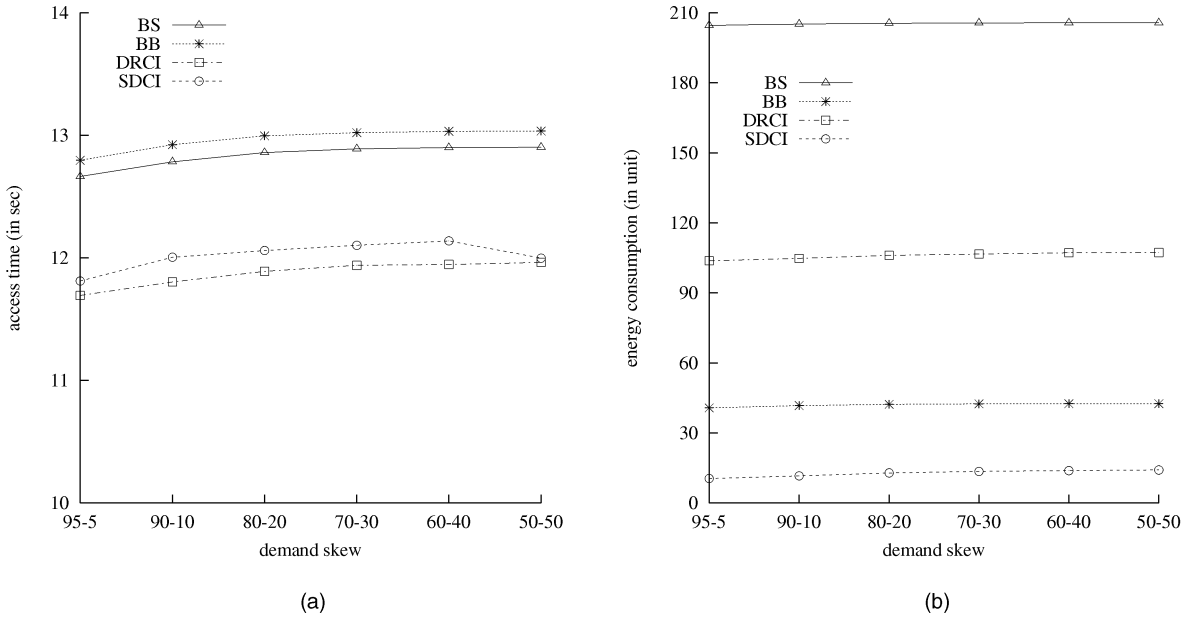

Fig. 16. Effect of demand skew. (a) Access time. (b) Energy consumption.

our results shown in Fig. 15. We note that for all the schemes, the access time and energy consumption increase as α decreases. The relative performance remains largely the same.

5.6 Effect of Demand Skew

Another factor that can affect the effectiveness of the caching schemes is the demand skew, i.e., the set of objects that are being queried. In the previous experiments, we have used the default of 90 percent of the query objects are directed at 10 percent of the database objects. In this experiment, we shall see the effect of demand skew on our schemes. As in the experiments on update skew, we experimented with demand skew of the form $\alpha - \beta$, where α percent of the queries are directed at β percent of the database and $\alpha + \beta = 100\%$. We experimented with the demand skew with α values from 50 percent to 95 percent. Fig. 16 shows the results. Like the update skew scenario, as α increases, a cache invalidation scheme should improve in performance. This is because the "working set" (objects queried) is small and, therefore, the updates reflected in the invalidation report are largely from objects not in the working set. In other words, the invalidation rate is expected to be low. Thus, both the access time and energy consumption for all schemes increase (slightly) as the demand skew decreases. The relative performance between

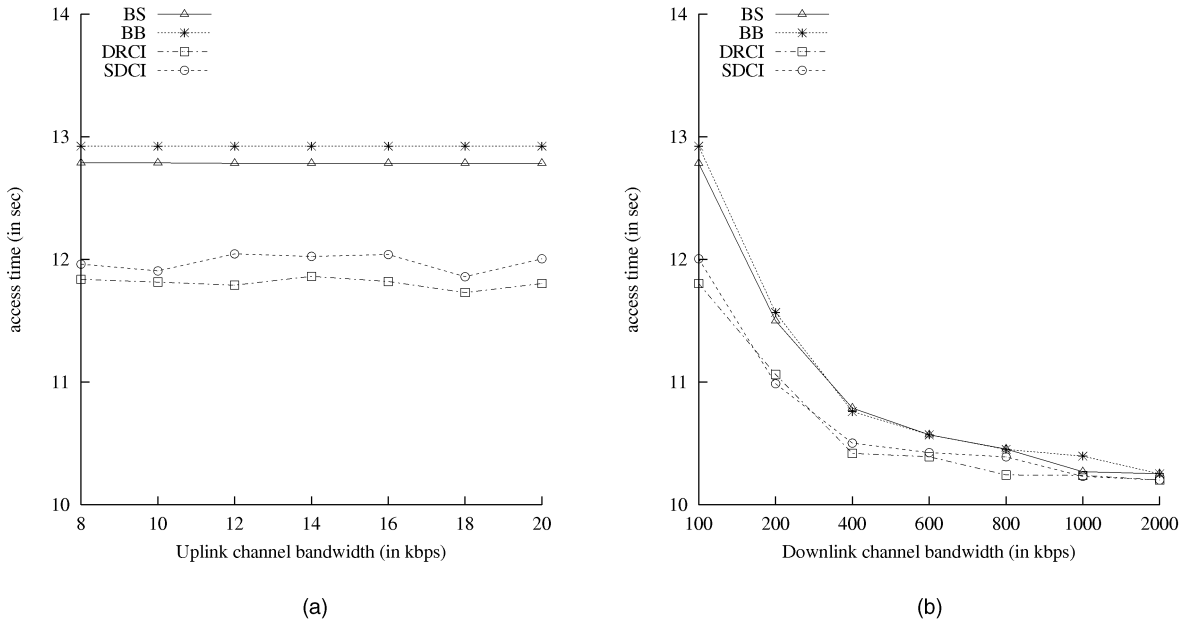

Fig. 17. Effect of channel bandwidth. (a) Uplink channel. (b) Downlink channel.

the algorithms are largely the same as that in previous experiments.

5.7 Effect of Channel Bandwidth

In this set of experiments, we study the effect of the channel bandwidth on the caching schemes. Since the energy consumption is not affected by the channel bandwidth, we shall only present the result for the access time. We first study how the uplink channel bandwidth may affect the performance of the caching schemes. We vary the uplink channel bandwidth from 8 kbps to 20 kbps. The access time result, as shown in Fig. 17a, indicates that the various schemes are largely unaffected by the uplink channel capability (at least not in the range we used in the study). Upon investigation, we note that this is because in our experiments, the uplink channel bandwidth is not a bottleneck. Moreover, since the schemes studied are largely broadcast-based schemes, the uplink cost to submit requests are very small. As such, increasing the uplink channel bandwidth did not reap much gain.

Next, we vary the downlink channel bandwidth from 100 kbps to 2 Mbps to study its effect on the performance of the various schemes. The result of this study is shown in Fig. 17b. As expected, increasing the downlink channel bandwidth can reduce the access cost significantly. It is also clear that the Bit-Sequences-based strategies benefit more from a larger channel downlink bandwidth.

The relative performance of the algorithms remains largely the same as in earlier experiments: BS is slightly better than BB, DRCI slightly outperforms SDCI, and the Dual-Report-based schemes are superior to the Bit-Sequences schemes.

6 Conclusion

In this paper, we have looked at the design of energy efficient cache invalidation schemes in a wireless computing environment. We have identified the issues that have to be considered in designing cache invalidation strategies. From the existing solutions to these issues, a large set of cache invalidation schemes can be constructed. We evaluated the performance of four representative algorithms. Two of the schemes—the Bit-Sequences (BS) scheme and the Dual-Report Cache Invalidation (DRCI) scheme—are variations of existing schemes. The two proposed schemes—the Bit-Sequences with Bit Count (BB) scheme and the Selective Dual-Report Cache Invalidation (SDCI) scheme—are respectively extensions of BS and DRCI that employ selective tuning to conserve energy. Our extensive performance study showed that the two proposed schemes are not only effective in salvaging the cache content but consume significantly less energy than their counterparts. While the Selective Dual-Report Cache Invalidation scheme performs best in most cases, it is inferior to the Bit-Sequences with Bit-Count scheme under high update rates.

We plan to extend this work in several directions. First, all the schemes studied require the clients to wait for the invalidation reports before their cache content can be validated. We plan to explore schemes that minimize this waiting time. Second, wireless channels are error prone. As such, the reports received may be corrupted. We are currently looking at how to handle errors in such an environment. Third, we plan to investigate how existing indexing schemes on broadcast data can be adapted for indexing invalidation report. Finally, some recent work has examined integrated schemes that combine multiple existing strategies [6] and adaptively broadcast different invalidation reports according to the runtime status. We would like to see how our schemes can be integrated as well.

References

[1] B.R. Badrinath and P. Sudame, "To Send or Not to Send: Implementing Deferred Transmissions in a Mobile Host," *Proc. 16th Int'l Conf. Distributed Computing Systems,* pp. 327-333, May 1996.

[2] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching in Mobile Distributed Environments," *Proc. 1994 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 1-12, June 1994.

[3] J. Cai and K.L. Tan, "Energy Efficient Selective Cache Invalidation," *Wireless Networks,* vol. 5, no. 6, pp. 489-502, 1999.

[4] J. Cai, K.L. Tan, and B.C Ooi, "On Incremental Cache Coherency Schemes in Mobile Computing Environment," *Proc. 13th Int'l Conf. Data Eng.,* pp. 114-123, Apr. 1997.

[5] G.H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *Computer,* vol. 27, no. 6, Apr. 1994.

[6] Q. Hu and D. Lee, "Adaptive Cache Invalidation Methods in Mobile Environments," *Proc. Sixth Int'l Symp. High Performance Distributed Computing,* 1997.

[7] T. Imielinski and B.R. Badrinath, "Data Management for Mobile Computing," *SIGMOD RECORD,* vol. 22, no. 1, pp. 34-39, Mar. 1993.

[8] T. Imielinski and B.R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," *Comm. ACM,* vol. 37, no. 10, Oct. 1994.

[9] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Energy Efficient Indexing on Air," *Proc. 1994 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 25-36, June 1994.

[10] J. Jing, A. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," *Mobile Networks and Applications,* vol. 2, no. 2, pp. 115-127, 1997.

[11] J. Jing, A. Helal, and A. Elmagarmid, "Client-Server Computing in Mobile Environments," *ACM Computing Surveys,* vol. 31, no. 2, pp. 117-157, June 1999.

[12] W.C. Lee and D. Lee, "Using Signature and Caching Techniques for Information Filtering in Wireless and Mobile Environments," *J. Distributed and Parallel Databases,* vol. 4, no. 3, pp. 205-227, 1996.

[13] P.V. Argade, S. Aymeloglu, A.D. Berenbaum, M.V. DePaolis Jr., R.T. Franzo, R.D. Freeman, D.A. Inglis, G. Komoriya, H. Lee, T.R. Little, G.A. MacDonald, H.R. Mclellan, E.C. Morgan, H.Q. Pham, G.D. Ronkin, R.J. Scavuzzo, and T.J. Woch, "Hobbit: A High-Performance, Low-Power Microprocessor," *Proc. COMPCON '93, Int'l Computer Conf.,* pp. 88-95, Feb. 1993.

[14] K.L. Tan and B.C. Ooi, *Data Dissemination in Wireless Computing Environments.* Kluwer Academic, 2000.

[15] K.L. Tan and J. Cai, "Broadcast-Based Group Invalidation: An Energy Efficient Cache Invalidation Scheme," *Information Sciences,* vol. 100, nos. 1-4, pp. 229-254, Aug. 1997.

[16] K.L. Tan and J.X. Yu, "Energy Efficient Filtering of Nonuniform Broadcast," *Proc. 16th IEEE Int'l Conf. Distributed Computing Systems,* pp. 520-527, May 1996.

[17] K. Watkins, *Discrete Event Simulation in C.* McGraw-Hill, 1993.

[18] K.L. Wu, P.S. Yu, and M.S. Chen, "Energy-Efficient Caching for Wireless Mobile Computing," *Proc. 12th Int'l Conf. Data Eng.,* pp. 336-343, Feb. 1996.

Kian-Lee Tan received the BSc (Hons) and PhD degrees in computer science from the National University of Singapore, in 1989 and 1994, respectively. He is currently an associate professor in the Department of Computer Science, National University of Singapore. His major research interests include multimedia information retrieval, wireless information retrieval, query processing, optimization in multiprocessor and distributed systems, and database performance. He has published more than 80 conference/journal papers in international conferences and journals. He has also coauthored three books. He is a member of the ACM and of the IEEE Computer Society.

Jun Cai received the BSc and MSc degrees in computer science from Fu Dan University, PRC, in 1989 and 1992, respectively, and the PhD degree in computer science from the National University of Singapore in 1998. He is currently a senior engineer with the Philips Electronics Singapore Pte Ltd.

Beng Chin Ooi received the BSc and PhD degrees in computer science from Monash University, Australia, in 1985 and 1989, respectively. He was with the Institute of Systems Science from 1989 to 1991 before joining the Department of Information Systems and Computer Science, National Unviersity of Singapore. His research interests include database performance issues, multimedia databases and applications, high-dimensional databases, and GIS. He is the author of a monograph entitled *Efficient Query Processing in Geographic Information Systems* (Springer-Verlag, LCNS #471, 1990) and a coauthor of three other books. He has published 70 conference/journal papers and served as a program committee member for a number of international conferences (including ACM SIGMOD, VLDB, EDBT, DASFAA) and serves as an editor for *Geoinformatics*, *International Journal of GIS*, and *Journal on Universal Computer Science*. He is a member of the ACM and of the IEEE Computer Society.