Wireless Networks 5 (1999) 489–502

Energy-efficient selective cache invalidation

Jun Cai and Kian-Lee Tan

Department of Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore 119260

In a mobile environment, users of portable computers can query databases over the wireless communication channels. To reduce contention on the limited bandwidth of the wireless channels, frequently accessed data are cached at the mobile clients. However, maintenance of the cache consistency is complicated by the mobile clients' frequent disconnection to conserve energy. One promising approach in the literature is for the server to periodically broadcast invalidation reports from which clients can salvage their cache content that are still valid. This mechanism is, however, not energy efficient as clients are expected to examine the entire invalidation report. In this paper, we reexamine the cache coherency problem and propose three novel cache coherency schemes. While these schemes are based on periodic broadcast of invalidation reports, they allow clients to selectively tune to the portions of the invalidation report that are of interest to them. This allows the clients to minimize the power consumption when invalidating their cache content. We conducted extensive studies based on a simulation model. Our study shows that the proposed schemes are not only effective in salvaging the cache content that are still valid (and hence result in lower access time), but are also efficient in energy utilization. While none of the proposed algorithms is superior in both the access time and energy efficiency, one of the schemes, Selective Cache Invalidation, provides the best overall performance.

1. Introduction

Recent advances in wireless networks and computer down-sizing technologies have led to the development of the concept of mobile computing. In the near future, millions of mobile users will be equipped with small, yet powerful battery-operated palmtops. Through the wireless links, these portable equipments will become an integrated part of existing distributed computing environments, and mobile users can access data stored at information servers located at the static portion of the network even while they are on the move. In our increasingly mobile world, the ability to access information on demand in any location and at any time can satisfy people's needs as well as confer them with a competitive advantage. As such, the potential market for mobile computing applications has been estimated to be billions of dollars annually [8]. For example, passengers will access airline schedules, investors will access stock activities and travelers will access weather and/or traffic conditions.

For mobile computing to be widely accepted, there are two obstacles to be overcome. First, for palmtops that operate on AA batteries, power conservation is a key issue. According to [1,8], for an "average user", the power source is expected to last 2–3 hours before replacing or recharging becomes necessary. What makes it worse is the prediction by battery experts of the modest improvement in battery capacity of only 20–30% over the next 5–10 years [6,14]. Therefore, mobile computers would frequently be disconnected from the network or be kept in weak connection status to save energy. Second, the bandwidth of the wireless channel is also very limited, varying from 1.2 Kbps for slow paging channels, through 19.2 Kbps (e.g., CDPD) to about 2 Mbps for the wireless LAN. This calls for mechanisms (such as minimizing transmissions) to avoid the contention of the limited bandwidth. These two issues pose a great challenge to researchers in the community.

Caching of frequently accessed data at the mobile clients has been considered to be a very useful and effective mechanism in conserving bandwidth and energy. When the data items are available in the cache, mobile clients can avoid uplink requests (provided these objects are valid). As argued in [2,3], reduced transmission implies better utilization of the limited bandwidth. Moreover, energy does not need to be spent in transmitting data. Note that transmitting data consumes more energy than receiving data, and energy consumed is proportional to the fourth power of the distance between the clients and the server and can be affected by factors such as terrain, season and rain. In addition, minimizing transmissions can also result in cost benefits, especially, if one is billed on a pay-per-packet basis.

However, the frequent disconnection and mobility of clients complicate the issue of keeping the cache consistent with that of the server. One promising approach in the literature is for the server to periodically broadcast object invalidation reports. An object invalidation report contains information about objects that have been updated recently. Based on the report, clients only need to invalidate objects that are found in the report and can salvage their cache content that are still valid. This mechanism is, however, not energy efficient as clients are expected to examine the entire invalidation report.

In this paper, we reexamine the cache coherency problem and propose three novel cache coherency schemes. While these schemes are based on periodic broadcast of invalidation reports, they are unique in two ways. First, all the proposed schemes adopt a group-based approach, i.e., broadcast a group invalidation report. The Group Cache Invalidation scheme broadcasts a group invalidation report while the Hybrid Cache Invalidation scheme and the Selec-

tive Cache Invalidation Scheme broadcast a pair of invalidation reports (an object invalidation report and a group invalidation report). Second, all the schemes allow clients to selectively tune to the portion of the invalidation report(s) that are of interest to them. This allows the clients to minimize the power consumption when invalidating their cache content. We conducted extensive studies based on a simulation model. Our study shows that the proposed schemes are not only effective in salvaging the cache content that are still valid, but are also efficient in energy utilization. While none of the proposed algorithms is superior in both the access time and energy efficiency, the Selective Cache Invalidation scheme provides the best overall performance.

The remainder of this paper is organized as follows. In the next section, we present the system model and review related work. A detail discussion of the basic cache coherency scheme that has been proposed in the literature is also given. Section 3 presents our proposed cache coherency schemes. The simulation model and the simulation results are presented in section 4. In section 5, we conclude with discussions on some future work.

2. Preliminaries

2.1. The context

The model for a mobile data access system adopted in this paper, as shown in figure 1, is similar to that specified in [3]. 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

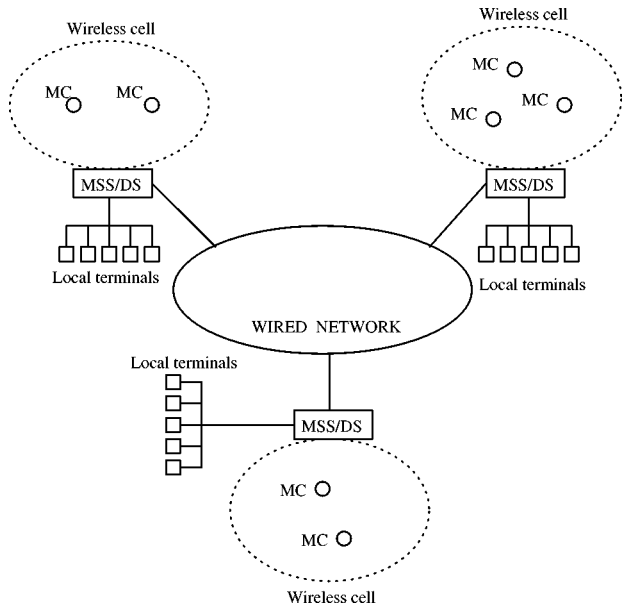


Figure 1. The wireless computing environment.

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 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 non-volatile 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 [16,18], 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 can be used to characterize information retrieval in wireless environment. The first is the access time which is the time from the initiation of the query to the time when all requested objects are downloaded. The second is the energy consumption during the retrieval process. It is highly desirable to design mechanisms that minimize both the access time and energy consumption.

2.2. Related work

In this section, we review the cache coherency schemes proposed in the literature. We also briefly review selective tuning mechanisms which have largely been studied in the context of “indexing on air”.

2.2.1. Cache coherency schemes

The use of caching strategy requires a cache coherency scheme to ensure that the cached objects are consistent with those stored in the server. However, as mobile clients can disconnect and be powered off, it becomes difficult for the mobile clients to know whether their cached data are still valid or not, especially after long disconnection time. Two basic categories of cache coherency strategies have been discussed in the literature [3]. Under 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 sends 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 keeps track of the update history (of a reasonable length) and provides this information to clients either by periodic broadcasting or as and when requested by the clients. Because the stateful approach is more complex (e.g., servers may have to track clients who have moved out of the current cell or disconnected, etc.), most of the existing work are based on the stateless approach. This paper also focuses on stateless schemes.

Barbara and Imielinski are among the first to address cache invalidation issues in wireless environment [3]. In [3], they proposed three cache invalidation schemes where the server periodically broadcasts object invalidation reports to mobile clients. Mobile clients use the invalidation reports to invalidate their caches accordingly. The three schemes differ in the contents of the invalidation reports. Under the schemes, if the disconnection time exceeds a certain threshold, clients may be forced to discard the entire cache contents upon reconnection. This is because clients are unable to determine the validity of their cache contents since their disconnection. Discarding the entire cache after a long disconnection time essentially takes away the benefits of caching, especially if the update rate is low and most of the cached objects are still valid. The effectiveness of these schemes, thus, depend on the length of the disconnection, the update rate as well as the objects updated.

In [11], two promising cache invalidation techniques that broadcast invalidation reports were proposed. The invalidation report for the first technique, called Bit-Sequences (BS), contains a set of binary bit sequences with an associated set of timestamps. The bit sequences are organized as a hierarchical structure with the highest level having as many bits as the number of objects in the database, and the lowest level has only 1 bit. Clients will use the bit sequence at level k with the most recent timestamp that is equal to or predates their disconnection times to invalidate their caches, and items in the sequences marked "1" will be invalidated. The nice property of the scheme is that clients are guaranteed that, among the data items that are marked in the bit sequence, at least half of them have actually been updated in the database. While the scheme was shown to be effective in reducing the number of cached objects discarded, the size of the bit sequence broadcast is large and can lead to poor bandwidth utilization. To cut down on the size of the bit sequence, a scaleable version, called Multi-Level BS (ML-BS), was proposed. Unlike BS, each bit in ML-BS corresponds to a group. The simulation study also showed the effectiveness of ML-BS especially for "information feed" application domain.

In [18], several group cache invalidation strategies were proposed. Unlike the schemes in [3,11], the basic group cache invalidation strategy checks the cache validity with the server after a reconnection. As such, the schemes can retain as many valid objects as possible. To avoid excessive usage of the uplink bandwidth, the database is partitioned into a number of groups. The server maintains for each group an object update history of the past WL seconds where L is the time interval for broadcasting an invalidation report, and W is an integer greater than zero. The server can then check the validity of a group by examining whether the objects in the group have been updated since the mobile clients become disconnected. If any of the objects in a group has been updated, the entire group has to be invalidated. However, under the basic strategy, some invalidation may be unnecessary. This is because the objects in a group that have been updated may also appear in the most recent invalidation report received by the clients, while the other objects of the group have not been updated. Under such a scenario, the group is still valid. To reduce such unnecessary invalidation, the GCORE scheme was proposed. GCORE distinguishes between hot and cold update objects in each group. Hot objects are those that are included in the most recent invalidation report that is broadcast. Instead of examining the entire group, GCORE excludes hot objects from the group before checking for their validity. In this way, valid groups can be salvaged, and future downlink cost can be significantly reduced. The reported study also demonstrated the superiority of GCORE in terms of bandwidth requirement. Unfortunately, the scheme still requires a substantial amount of uplink requests: for each query, uplink requests are necessary to invalidate the cache and uplink requests are also needed to request for the desired objects (that are not in the cache or have been invalidated).

In [15], a Broadcast-base Group Invalidation (BGI) was proposed. The BGI scheme can be considered a broadcast-based version of GCORE [18]. However, it is distinguished from GCORE in several ways. First, it minimizes uplink requests (and hence energy). Second, in GCORE, the server is responsible for validating the cache at group levels, but under BGI, the clients are the ones performing the task. Finally, BGI exploits grouping of hot and cold update objects, while GCORE did not address grouping issues.

More recently, several incremental cache invalidation schemes have also been proposed for relational operators (such as selection, projection and join) [4].

2.2.2. Selective tuning schemes

The idea of selective tuning is first introduced by Imielinski, Viswanathan and Badrinath [9]. The basic idea is to organize broadcast data such that the CPU can oper-

ate in the less power consuming doze mode most of the time and "wake up" to listen to the channel only when the data of interest are broadcast. This is achieved by building "indexes on air", i.e., transmitting an index along with the data. Two schemes were studied in this paper. The $(1,m)$ indexing scheme broadcasts an index m times during one full broadcast. While this method reduces tuning time, it leads to long access time because of the additional m copies of indexes. The second scheme, the distributed indexing scheme, exploits only partial replication at index level so that several distinct indexes are used to index different portion of the data broadcast. It was shown that distributed indexing results in the best access time and tuning time. The same authors also proposed two hash-based indexing schemes and a flexible indexing scheme [10]. Signature-based techniques have also been studied in [9,12]. Indexing over non-uniform broadcast environment (where popular data are broadcast more frequently than less popular ones) can be found in [5,16].

2.3. The basic scheme: Object-based Cache Invalidation (OCI)

Suppose the server contains a database *DB* of objects. The server also maintains an update history of objects during the last wL time units, where w is referred to as the update broadcast window. The client maintains a cache of frequently accessed objects. Each object is also associated with a timestamp *TS* that reflects the last known update time of the object. The client also keeps a variable T^c that indicates the last time it receives an update report.

The Object-based Cache Invalidation (OCI) algorithm is similar to the one proposed in [3]. Under this algorithm, the server broadcasts the object invalidation reports (OIR) periodically at times $T_i = iL$ where $i > 0$ and L is the broadcast interval of update report. The content of OIR at time T_i is defined as follows:

$OIR_i = \{[id, TS_{id}] \mid id$ is the object identifier of an object $O \in DB$ and TS_{id} is the timestamp of the last update of O such that $T_i - wL \leqslant TS_{id} \leqslant T_i\}$.

Suppose the next invalidation report will be broadcast at time T_i. At the client, the client maintains a list of query objects, Q_i, in the interval $[T_{i-1}, T_i]$. At time T_i, the client listens to the invalidation report, and validates its cache content as follows. If the difference between the current report timestamp (i.e., T_i) and T^c is bigger than wL, the entire cache is dropped. This is because there is no mechanism for the client to know whether the cached content is still valid. Otherwise, the timestamps of the cached objects are compared against the timestamps of the objects in the report. If the object has been updated recently, then it is invalidated; else its timestamp is updated to reflect the latest timestamp at which it is known to be valid. Figure 2 shows an algorithmic description of the scheme.

```
if (T_i − T^c > wL) {drop the entire cache}
else {
    for (every object O with identifier id
        in the client cache) {
        if (there is a pair [id, TS_id] in OIR_i) {
            if (TS^c_id < TS_id) {
                invalidate O
                remove O from cache
            }
            else {TS^c_id ← TS_id}
        }
    }
}
for (every object O ∈ Q_i) {
    if (O is in the cache) {
        use the cache to answer the query
    }
    else {submit uplink request for object}
}
T^c ← T_i
```

Figure 2. The object-based cache invalidation scheme.

OCI has the following noticeable characteristics:

1. The scheme is not energy efficient. To check the cache's validity, the whole OIR has to be downloaded. The energy consumed solely on invalidation can be significant and wasteful because only a very small portion of information in the OIR may actually be useful to a specific mobile client. For example, suppose a query only involves 30 data objects that are cached locally. If the average disconnection time of a client is around 200 seconds. Then intuitively we will set w to no less than 200 seconds. Suppose the update rate is about 10 obj/sec, then an OIR will contain roughly 2000 records of updated information (if the database is very large and update is uniformly distributed on the database). That is to say, to check the validity of 30 data objects, an OIR of 2000 records will be downloaded. Although the size of each OIR record could be very small, the entirety is just too large to be ignored. One can easily calculate that with some certain values the energy consumption for this algorithm will be even greater than a strategy without any caching technique.
2. The update window w plays a vital role in this algorithm since the entirety of the cache will be discarded when $T_i - T^c > wL$. The purpose of the update report is to tell the clients at what time what data have been updated. As disconnection of a mobile client is quite a common phenomenon in mobile environment, it becomes important that the value of w be carefully chosen so that the update report contains enough information to meet the requirements of most of the clients. Suppose the average disconnection time is fixed. Then, it can be observed that the size of the update report depends on the update rate. When the update rate is high,

the size of the OIR can also be very large and takes up a significant portion of the downlink bandwidth. To keep the OIR small, one can use a small value for w which would mean low cache salvage rate, i.e., fewer objects can be salvaged by the clients. This is equally bad since more data have to be transmitted over the downlink channel to answer the queries. Finding an optimal value is a non-trivial task.

3. The above drawbacks of this algorithm result in the fact that invalidation is done at the cache level, that is to say, each time the client processes a query, the whole cache content should be checked for its validity in order to restore the cache to the current status. Without doing this, the timestamp of the cache content cannot be advanced to the timestamp of the current update report, and eventually, $T_i - T^c$ will be larger than w even if the mobile client actually disconnects for a short while each time.

```
for (every pair [gid, TS_gid] ∈ GIR_i) {
    if (client caches objects in group gid) {
        if (TS^c_gid < TS_gid) {
            remove all objects in corresponding
                group from the cache
        }
        else {TS^c_gid ← TS_gid}
    }
}
for (every object O ∈ Q_i) {
    if (O is in the cache) {
        use the cache to answer the query
    }
    else {submit uplink request for O}
}
```

Figure 3. The group-based cache invalidation scheme.

3. Energy-efficient cache coherency schemes

In this section, we present the three proposed cache coherency schemes to address the limitations of the OCI scheme. In other words, our algorithms attempt to minimize energy consumption, salvage as many cache objects as possible, and perform invalidation at query level.

3.1. Group-based Cache Invalidation (GCI)

We shall begin by looking at the basic group-based scheme, Group-based Cache Invalidation (GCI). GCI operates as follows. At the server, the whole database is partitioned into a set of disjoint groups, each containing the same number of objects. Note that each data object has an additional attribute (i.e., group id) associated with it. The server also keeps information on the most recent updates to the groups. In other words, each group has associated with it a timestamp which corresponds to the most recent update on an object in the group. We shall refer to this fixed size information at time T_i as the GIR_i:

$$GIR_i = \{[gid, TS_{gid}] \mid gid \text{ is the group id of a group of objects and } TS_{gid} \text{ is the timestamp of the most recent update of the group}\}.$$

The server periodically broadcasts GIR. At the client, the client organizes the cache content based on the group ids, i.e., all objects with the same group id are placed together, and the entire group has associated with it a single timestamp TS^c_{gid} which represents the last known update time of the group. As in OCI, the client also maintains a list of query objects, Q_i, during the interval $[T_{i-1}, T_i]$. At time T_i, the client listens to the GIR and uses it to invalidate the cached objects based on the group timestamps. The resulting algorithm is as shown in figure 3.

The GCI scheme has several advantages over OCI. First, the size of GIR is fixed and is independent of the duration of the client's disconnection and update rate. This means a fixed bandwidth utilization for broadcasting GIR. Second, discarding of cache content is based purely on invalidation (though it may be false invalidation) and has nothing to do with the client's disconnection time. Third, the concept of selective tuning can be exploited to reduce energy consumption. As group identifiers are in fact virtual ids, we can assign them in consecutive values. The GIR can then be broadcast in a fixed order (such as an increasing order of *gid*). Since the clients are aware of the groups that they are interested in, they only need to be in active mode to examine the timestamps of the relevant groups, while most of the other time can be spent in doze mode. This can be done since the size of a (gid, TS_{gid}) pair is fixed. Furthermore, the client need not download the entire GIR; instead, only the relevant (gid, TS_{gid}) pairs from GIR need to be examined. In other words, cache invalidation can be done at query level and not cache level. Therefore energy consumed by invalidating cache can be extremely small.

However, GCI also has several disadvantages. The most distinguished shortcoming of this scheme is false invalidation. False invalidation is defined as the phenomenon that a cached object is wrongly invalidated even though it is still valid (e.g., the cached object is valid but other objects in the same group have been updated resulting in the group being invalidated). As invalidation is done at group level, we can conceive that, under high update rate, the cache salvage rate for this scheme can be significantly smaller than the basic algorithm. While bandwidth can be saved by fixing the GIR size and energy saved by the group-based invalidation, the penalty brought by lower cache salvage rate is reflected in more unnecessary downlink transmission for false invalidation. As downloading information also consumes energy, thus the penalty for this scheme is expected to far outweigh the gain for high update rates.

3.2. The Hybrid Cache Invalidation (HCI)

A very natural way to handle the limitations of algorithms OCI and GCI is to integrate them into a Hybrid Cache Invalidation (HCI) scheme. In this case, the server broadcasts every L time units a pair of invalidation reports, an object invalidation report (OIR) and a group invalidation report (GIR). To realize this, the server keeps two update lists (corresponding to the GIR and OIR respectively):

$OU_i = \{[id,\ TS_{id}] \mid id$ is the identifier of object $O \in DB$ and TS_{id} is the timestamp of the last update of O such that $T_i - wL \leqslant TS_{id} \leqslant T_i\}$,

$GU_i = \{[gid,\ TS_{gid}] \mid gid$ is a group identifier and TS_{gid} is the timestamp of the latest update among all objects in the group such that $TS_{gid} < T_i - wL\}$.

Essentially, OU_i (and OIR) reflects the updates on individual objects during the last w broadcast intervals of the invalidation report, while GU_i (and GIR) monitors the group update behaviors. Unlike scheme GCI, the determination of TS_{gid} is optimized to salvage as much cache content as possible. This is based on the observation that OIR would have contained the information of some of the recently updated objects, and hence the valid timestamp for each group can be pushed further back than the most recent update time of this group. Thus, the valid timestamp for each group is given by the timestamp of the object in the group that is updated at time closest to $T_i - wL$.

The HCI 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 $T_i - wL$, OIR and GIR can work together so that the clients need not discard the entirety of its cache. The OIR minimizes false invalidation and the GIR avoids invalidation of the entire cache for long disconnection time. The algorithmic description of the protocol is shown in figure 4.

3.3. Selective Cache Invalidation (SCI)

HCI, unfortunately, is limited by the energy consuming operation of downloading the OIR. To minimize the energy consumption, we propose that we should focus on objects related to the query only. In this way, we can facilitate selective tuning of the reports. We call the new scheme Selective Cache Invalidation (SCI). The main difference between SCI and HCI lies in the organization of the pair of invalidation reports. First, in SCI, GIR is broadcast before OIR. Second, the entries in OIR are ordered and broadcast based on the groups. In other words, updates in the same group will appear together. A partition symbol will separate continuous groups. Third, an additional pointer should be added to each element of GIR, and this pointer points to the starting position of the objects within this group in the OIR. Figure 5 shows the organization of the invalidation reports.

At the client, it operates as follows. 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 are examined. The functionality of the improved algorithm SCI is as shown in figure 6.

```
for (every pair [id, TS_id] ∈ OIR_i) {
    if (object id is in the client cache){
        if (TS^c_id < TS_id) {
            remove object from the cache
        }
        else {TS^c_id ← TS_id}
    }
}
if (T_i − T^c > wL) {
    for (every group gid in the client cache) {
        selectively tune to the pair
            [gid, TS_gid] in GIR_i
        if (TS^c_gid < TS_gid) {
            remove all objects in group
                from the cache
        }
        else {TS^c_gid ← TS_gid}
    }
}
for (every object O ∈ Q_i) {
    if (O is in the cache) {
        use the cache to answer the query
    }
    else {submit request for O}
}
T^c ← T_i
```

Figure 4. The hybrid cache invalidation scheme.

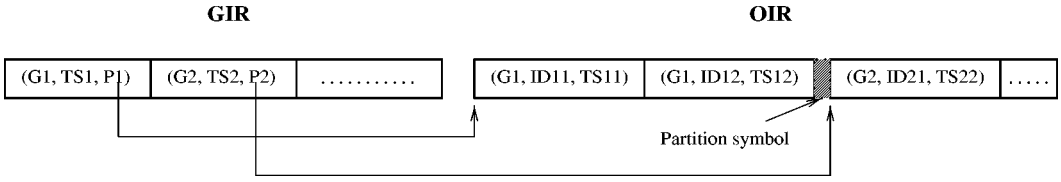


Figure 5. Organization of the invalidation reports.

```
P ← φ
for (every group in the client cache) {
    selectively tune to the
        triplet [gid, TS_gid, P_gid] in GIR_i
    if TS^c_gid < TS_gid {
        invalidate all objects of group
    }
    else {
        TS^c_gid ← TS_gid
        P ← P ∪ {P_gid}
    }
}
while (P ≠ ∅) {
    p ← first(P)
    P ← P − {p}
    switch to doze mode until the position p
        is coming
    while (not encounter partition symbol) {
        download pair [id, TS_id]
        if (id is in client cache) {
            if TS^c_id < TS_id {
                invalidate object
            }
            else {
                TS^c_id ← TS_id
            }
        }
    }
}
for (every object O ∈ Q_i) {
    if (O is in the cache) {
        use the cache to answer the query
    }
    else {submit uplink request for O}
}
```

Figure 6. The selective cache invalidation scheme.

3.4. Discussion on grouping

The performance of the three proposed group-based schemes can be influenced by the grouping scheme. There are two issues that concern grouping, namely how the objects should be grouped and the size of each group (or number of groups). For the first issue, we based our approach on the observation 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 as shown in figure 7.

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 can be formed from objects in the same category. In our study, we divided the whole database into the four different categories, and groups are formed within each category.

	Hot demand	Cold demand
Hot update	HH	HC
Cold update	CH	CC

Figure 7. Grouping of objects.

There is also a tradeoff in the number of groups. If the number of groups is too large, the size of the GIR will also be large, and this will utilize a significant amount of the channel bandwidth which may lead to poor channel utilization. On the other hand, a small number of groups may also be bad. This is because when the number of groups is small, the size of the groups will be large. As a result, it becomes more likely for groups to be invalidated. In our performance study, we shall study the effect of the number of groups.

4. A performance study

We conducted extensive studies based on a simulation model. Four schemes are studied in this simulation, i.e., OCI, GCI, HCI and SCI. 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 request and the energy consumed to download the desired data. 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 1 K bits of information, and based on the findings in [7], transmitting data is assumed to be 10 times more power consuming than receiving data.

4.1. The simulation model

Figure 8 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 receives 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 1 shows the notations and default parameter settings used in the simulation. The database has a total of D objects that are organized into N groups, each with G

objects. The size of each object (excluding the object id and group id) is O bits. The size of the object id and group id are O_{id} bits and G_{id} bits, respectively. Each timestamp is T_{id} bit. β_1% of the data objects are in the hot update set, while $(100 - \beta_1)$% of them belong to the cold update set. β_2% of the data objects are in the hot demand set, while $(100 - \beta_2)$% 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% of the updates are focused on the hot update set while the remaining $(100 - \alpha_1)$% 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% 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 with a mean of ν. The update reports are broadcast every L seconds, and the update broadcast window is given by w. Finally, the bandwidth for the downlink and uplink channels are given by C_{down} and C_{up}, respectively. For the selective cache invalidation scheme, the separators between groups are each p bits. Most of the settings used are similar to those used in previous work [15,18].

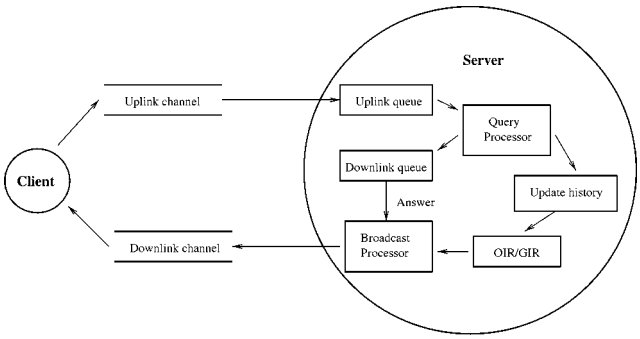


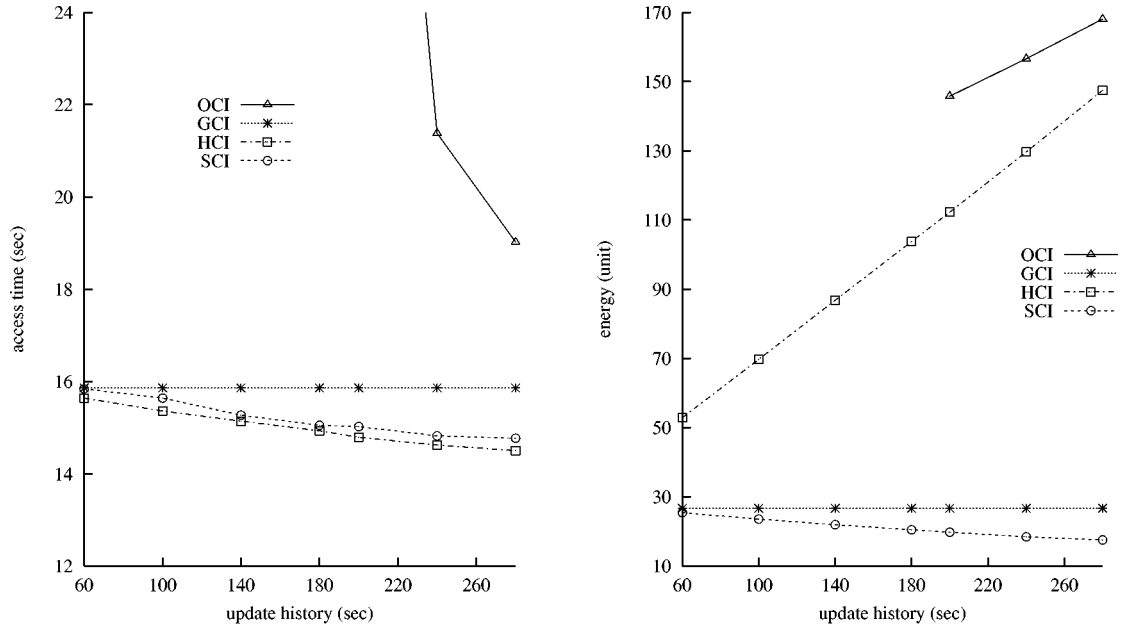
Figure 8. Mobile communication model.

4.2. Effect of update history, wL

In this experiment, we study the effect of w on the various algorithms. Intuitively, for OCI, since only OIR is employed, w should be selected such that wL is no less than ν. However, for the other schemes, the inclusion of GIR provided some information of the updates before time $T - wL$. Thus, a smaller w can achieve reasonable performance. Our simulation results confirm our intuition. In figure 9(a), we see that for small update window w, the access time for scheme OCI is very large. This is so because a small w value increases the probability of $\nu \geqslant wL$ (i.e., client's disconnection time is relatively longer than wL). As a result, more cached objects are being invalidated. This in turn results in an increasing demand on the downlink channel as more queries have to be processed from scratch (no cache content can be reused). The other three schemes avoid this problem by including a GIR in their update report. With GIR, clients disconnect before time $T - wL$ can always use it to salvage its cache content. Thus no query should be reprocessed unless all the related cache contents are invalidated. As no OIR is broadcast in the GCI scheme, the access time of this scheme is not affected by w. However, since GCI bases its validation scheme solely on GIR, false invalidation is high, resulting

Table 1
System and workload parameters.

Notation	Definition	Default values
D	Server database size	100,000 obj
λ_{q}	Query arrival rate, obeys Poisson distribution	2 query/sec
λ_{u}	Update arrival rate, obeys Poisson distribution	5 obj/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
G	Group size	100 obj
N	Total number of groups	$\approx D/G$
ν	Disconnection time obeys negative exponential distribution with mean ν	200 sec
C_{up}	Bandwidth of uplink channel	19.2 kbps
C_{down}	Bandwidth of downlink channel	100 kbps
L	Periodic broadcast interval	20 sec
w	Broadcast window	10
O	Object size	4096 bits
O_{id}	Object id size	32 bits
G_{id}	Group id size	16 bits
L_{gid}	Link size	16 bits
T_{id}	Timestamp size	64 bits
p	Partition symbol (for SCI only)	8 bits



(a) Access time (b) Energy consumption

Figure 9. Effect of update history (wL).

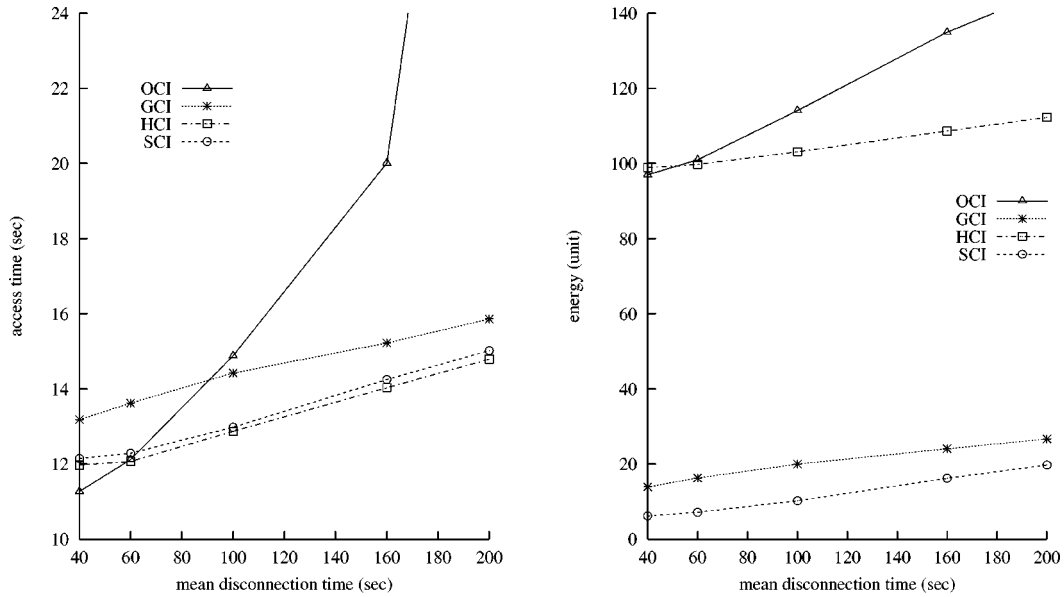
in a longer access time than schemes HCI and SCI. In fact, when the update rate is high, the performance of GCI will be severely affected (see in figure 11(a)). Both HCI and SCI are equally effective in salvaging the cache content. Because SCI has an additional link in each element of GIR for the purpose of selective tuning on OIR, thus it's access time is slightly longer than HCI.

The results on power consumption is shown in figure 9(b). As expected, OCI is the least energy efficient since it can salvage the least number of objects and requires downloading the entirety of the OIR. HCI also shows a high energy consumption because of its need to download the whole OIR when performing cache invalidation, but since it has a higher cache salvage rate than OCI, the energy consumed on receiving invalidated data objects is smaller than OCI. Thus the total energy consumed is also smaller than OCI. As for GCI and SCI, as selective tuning technique is applied in both schemes, energy consumption for these two schemes are quite small. Besides, as SCI has a higher cache salvage rate (SCI uses OIR to reduce the phenomenon of false invalidation which can be significant in GCI), it consumes the least energy.

4.3. Effect of disconnection time, ν

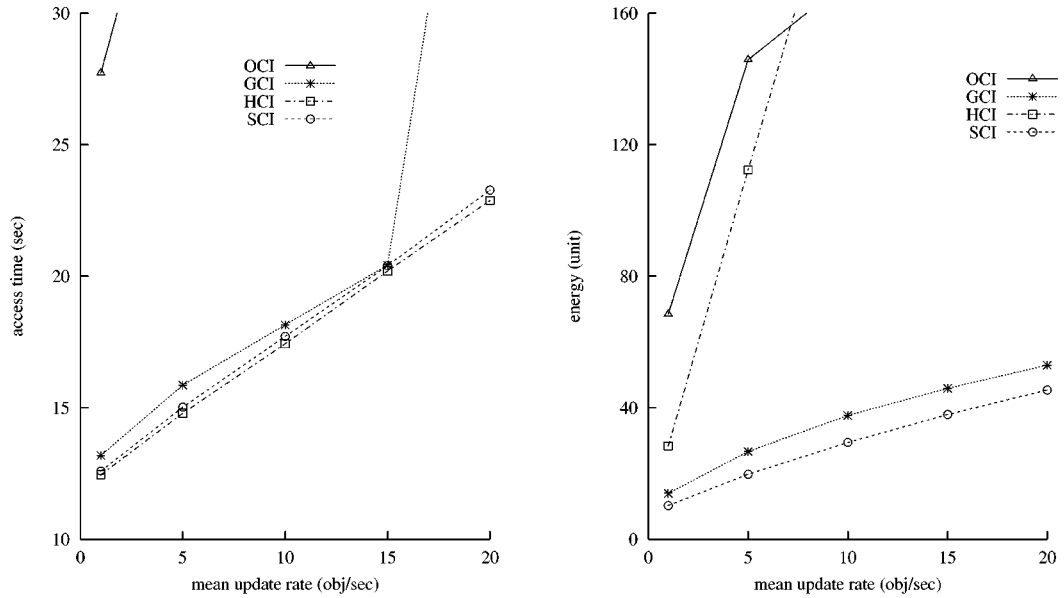
A user's behavior is difficult to predict especially in a mobile environment. Mobile client may or may not disconnect from the wireless network, and if it should, it may disconnect with different duration each time. As we mentioned before, disconnection is one of the most important factors that complicates 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. Figure 10 shows the results.

As shown in figure 10(a), the OCI scheme is the least robust while the proposed schemes are almost unaffected by the disconnection time. This is expected since we have seen in the previous simulation how w is related to ν. When ν is smaller than wL, all the schemes can effectively salvage many valid objects in the cache. In particular, we observe that for short disconnection time (< 60 sec in this experiment), GCI performs the worst because of its problem of false invalidation. We also observe that OCI outperforms HCI and SCI for two reasons. First, the short mean disconnection time minimizes the phenomenon of whole cache invalidation in OCI. Second, in HCI and SCI, a GIR is broadcast together with OIR which consumes additional bandwidth. Since OIR can cover enough update history because of the relatively small disconnection time, the usefulness of the GIR is very limited. However, as ν increases, the performance of OCI degenerates because most of the clients need to reprocess from scratch as a result of discarding their entire cache content. As for HCI and SCI, they reap the benefits of the GIR that lead to a high cache salvage rate. We note that GCI performs (slightly) worse than HCI and SCI (because of false invalidation). At a first glance, it appears that the performance of GCI is not that bad since it employs only GIR when broadcasting invalidation messages. However, in this experiment, the update rate is low (5 obj/sec), and the impact of false invalidation (which is the intrinsic weakness of group-based schemes) is not significant in the GCI schemes. In the next experiment, we shall study this effect on the GCI scheme.



(a) Access time (b) Energy consumption

Figure 10. Effect of disconnection time.



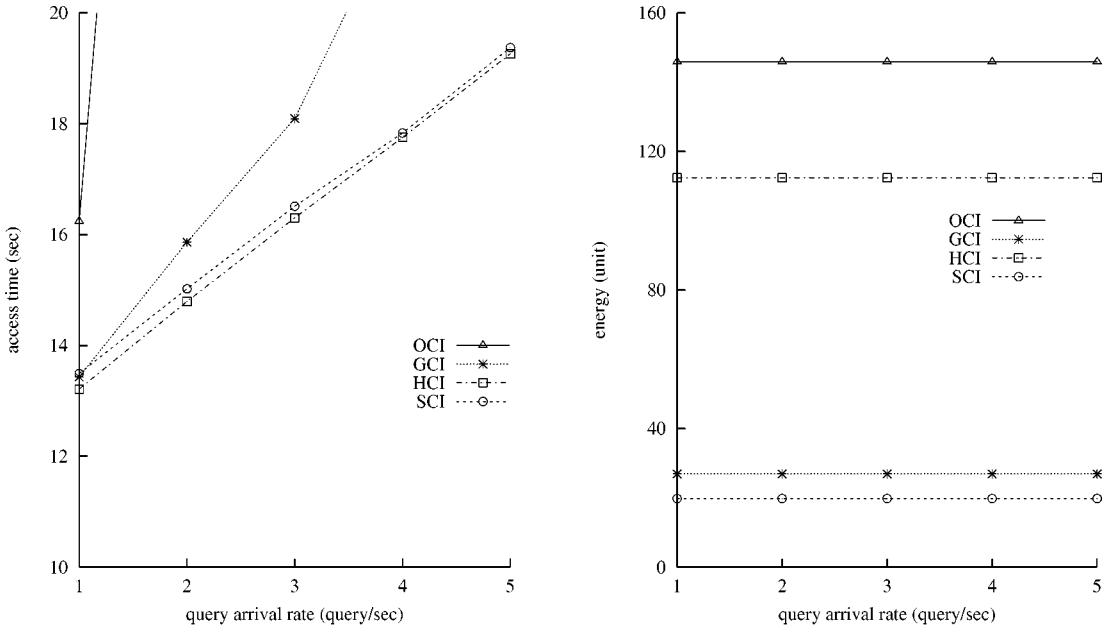
(a) Access time (b) Energy consumption

Figure 11. Effect of update rate.

Figure 10(b) shows the energy consumption of the various schemes. As in the previous experiment, SCI and GCI are highly energy efficient (with SCI slightly better than GCI). Schemes OCI and HCI consume more energy due to the downloading of the entirety of the OIR.

4.4. Experiment on update rate, λ_u

Figure 11 shows the effect of update rate on access time and energy consumption. A higher update rate means a higher probability of cache invalidation, and also a larger update history for a fixed w. This explains why the performance of OCI deteriorates so rapidly in both figure 11(a) and figure 11(b). GCI also degenerates rapidly as the update rate increases. When update rate is high (greater than 15 obj/sec), the average access time increases very fast. Even though GCI has a fixed update report size (thus more downlink bandwidth spared for invalidated objects), the penalty brought about by false invalidation is significant to result in unacceptable access time performance. Both SCI and HCI present a good performance in terms of access time demonstrating the effectiveness of exploit-



(a) Access time (b) Energy consumption

Figure 12. Effect of query arrival rate.

ing the pair of (OIR, GIR) reports in salvaging cache content.

In terms of energy consumption, HCI also shows a trend of high energy consumption because of the huge OIR size when update rate is high. As selective tuning is adopted in SCI, the energy consumption for this scheme is the least. GCI also performs selective tuning on GIR, however, since it has a smaller cache salvage ratio, the energy consumed on downloading invalidated objects is larger than SCI.

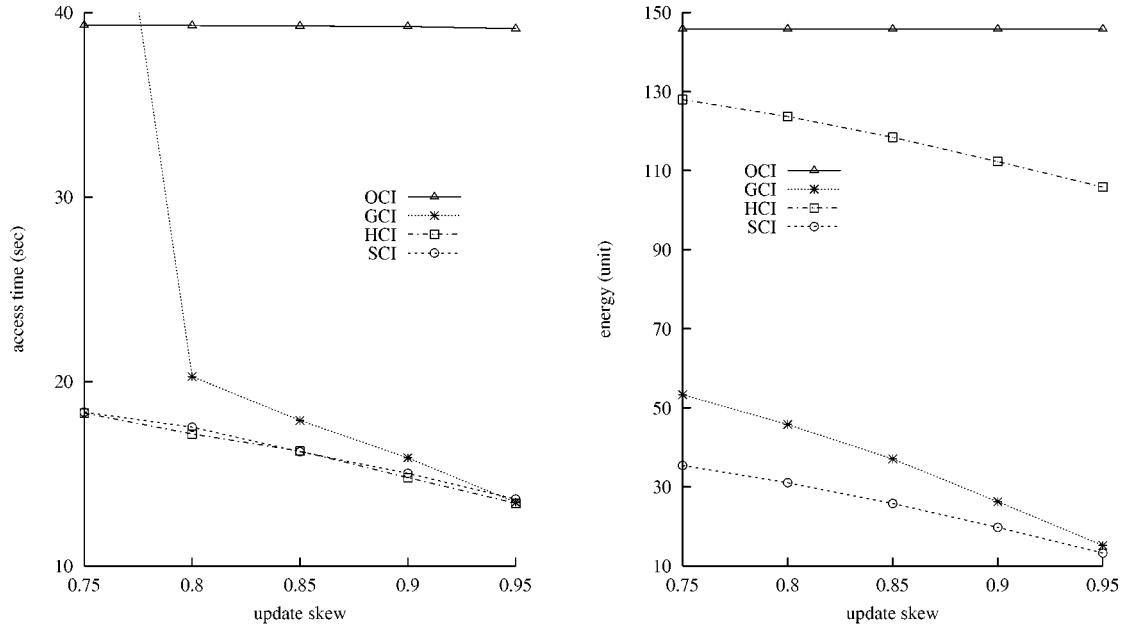
4.5. Effect of query arrival rate, λ_q (scaleability)

Scale (which represents the number of concurrent users that can be supported) is another important issue. The larger the number of queries the environment can support, the better the scheme is. Keeping other parameters fixed, this problem is mainly determined by two factors: the amount of downlink bandwidth that is available for data transmission (other than the invalidation reports) and the cache salvage rate. The results of this study is shown in figure 12.

Again OCI leads to the worst access time. This is because of its huge OIR size and low cache salvage rate. The large amount of data to be transmitted (as query rate increases) due to invalidation causes channel congestion quickly. For the same reason of contention on downlink channel due to high false invalidation, GCI is also inferior. For HCI and SCI, although their update reports are larger compared with OCI and GCI, the benefits brought by higher salvage rate of both OIR and GIR reports far outweigh their cons, leading to a much shorter access time (compared with OCI and GCI). Since energy consumption is determined by cache salvage rate and the invalidation scheme, the query arrival rate has no impact on it. As shown, the relative performance of the scheme remains largely the same: SCI is the best, followed by GCI, followed by HCI and OCI is the worst.

4.6. 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% of the updates directed at 10% 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% of the updates are directed at β% of the database, and $\alpha + \beta = 100$%. Intuitively, as α increases, a cache invalidation scheme should improve in performance, since fewer data are being invalidated. Figure 13 shows the results, where the x-axis denote the α values. From the result, we observe two phenomenon. First, the OCI scheme does not benefit from higher α values compared to the other three schemes. This is resulted from an intrinsic deficiency of the OCI scheme. With this scheme, when the disconnection time is greater than wL, the whole cache will be invalidated (i.e., 100% invalidation). Thus the small benefit brought about by higher α values (reflected in the other 3 schemes) is largely masked by the huge loss on whole cache invalidation. Second, the GCI scheme performs even worse than OCI for small α values. In our experiment, this happens when $\alpha < 0.8$. This is because more groups are being updated, leading to large number of false invalidation.



(a) Access time (b) Energy consumption

Figure 13. Effect of update skew (α_1).

The result on energy consumption is largely similar to previous experiments with SCI being the most energy efficient and OCI consumes the most energy.

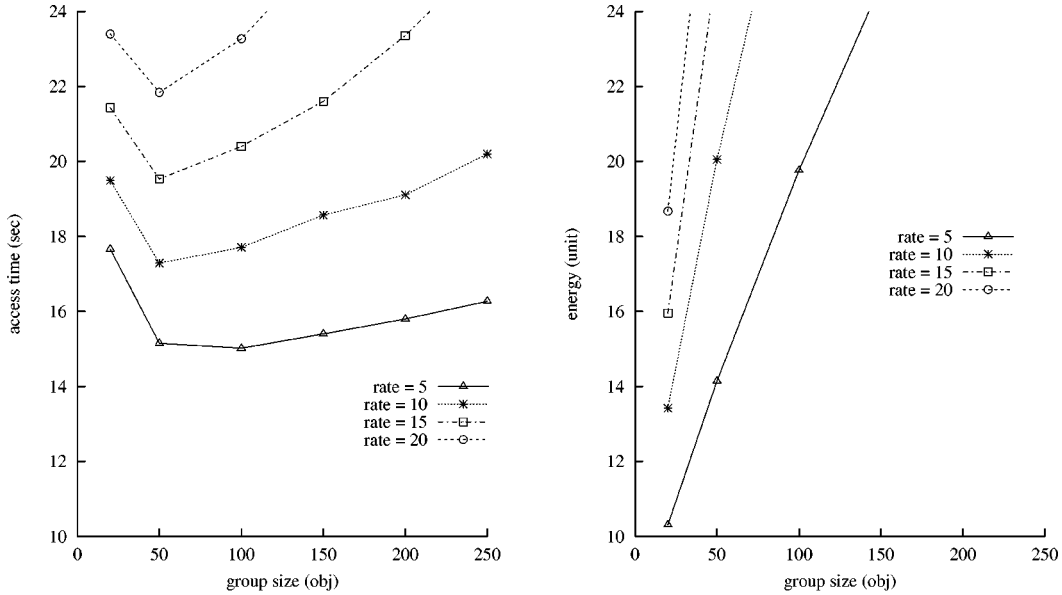
4.7. Effect of group size on SCI

In the above five experiments, we have demonstrated the superiority of the SCI scheme both in terms of energy consumption and access time. As grouping is an essential component used in group-based schemes, in this experiment, we shall study its effect on SCI. Specifically, we will explore the impact of group size on SCI. Recall that the main functionality of GIR is to provide clients with update information before time $T - wL$. Since a large group size implies high false invalidation, intuitively we prefer to adopt a small group size. However, a small group size results in a larger number of groups and thus a larger GIR size. The resultant effect may be poor performance since a significant portion of the downlink bandwidth may have to be utilized. Eventually the gain on reducing false invalidation will be overtaken by the loss due to severe contention on the downlink bandwidth. Since update rate is a critical source of cache invalidation in SCI, we examine the impact of grouping with four different update rates. The results of the experiment are shown in figure 14.

Figure 14(a) shows the tradeoffs that we have discussed in section 3.4 – neither too small or too large a group size is effective. Furthermore, we note that when the update rate is low, a large group size can lead to lower access time since the number of groups will be small and invalidation is not frequent. However, as the update rate increases, a large group size is not desirable because of the large number of false invalidation. In terms of energy consumption (figure 14(b)), the scheme favors small group size since decreasing group size results in fewer false invalidation – despite a larger GIR, the net effect is a gain in energy consumption.

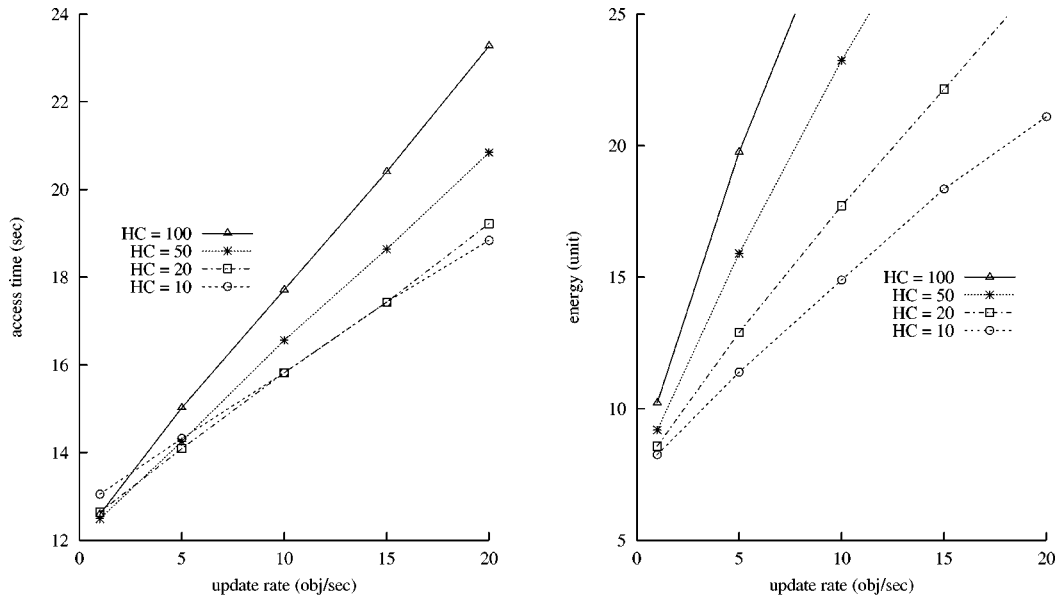
However, by further looking at the grouping scheme, we found that, since most user requests are focused on hot demand data, the benefit of the caching scheme can be improved if efforts are focused on salvaging those data. From the result of figure 14, we note that smaller group size is better when update rate is high (see the curve when update rate $= 15$). Therefore, in order to salvage more hot demand data, we need to use a smaller size for this data category. Besides, as hot update data are difficult to salvage, it makes sense to salvage hot demand, cold update data (i.e., the HC category). Figure 15 justifies our intuition.

Figure 15 shows the results of an experiment that varies the HC group size, while keeping the group sizes for the other categories at the default setting. In figure 15, we see that, when update rate is low (update rate $= 1$), a smaller HC group size performs slightly worse than a larger group size for the same reason as that in the previous experiment. However, as the update rate increases, more groups will be invalidated. In this case, a smaller group size is preferred. However, by comparing figures 14 and 15, we can see that, reducing the group size of all four categories uniformly is not a good choice because the benefit of higher cache salvage rate for smaller group size is diminished by the cost on higher bandwidth usage of GIR. Instead, selectively reducing group size of particular categories (e.g., HC group) is a better choice. For example, in figure 14, we see that, when the update rate is 20, the best result for uniformly reducing the group size of all categories is achieved when the group size is around 50. Here, the access time is



(a) Access time (b) Energy consumption

Figure 14. Effect of group size on SCI.



(a) Access time (b) Energy consumption

Figure 15. Effect of HC group size on SCI.

around 22. In figure 15, a HC group size of 20 can bring down the access time to around 19. The reason is that, by only utilizing small group size for the HC category, we reap enough benefit to justify the additional usage of bandwidth.

5. Conclusion

In this paper, we have reexamined the cache coherency problem in mobile computing environment. Caching frequently accessed objects in mobile clients can reduce the contention of channel bandwidth, minimize energy consumption and cost. However, to determine whether the copy at the client's cache is consistent with that of the server can be an expensive process. We have proposed three novel energy-efficient cache coherency schemes that are based on periodic broadcast of invalidation reports. The schemes organize the database into groups and organize the invalidation reports to facilitate selective tuning. We conducted extensive studies based on a simulation model and evaluated the proposed schemes against the traditional object-based cache invalidation scheme. Our study showed

that the proposed schemes are not only effective in salvaging the cache content that are still valid, but are also efficient in energy utilization. While none of the proposed algorithms is superior in both the access time and energy efficiency, one of the schemes, Selective Cache Invalidation, provides the best overall performance.

We have already begun work on designing selective tuning mechanisms for demand-driven data. Traditional selective tuning methods proposed as "indexes on air" cannot be employed here because demand-driven data are not predictable. Second, we plan to look at the hybrid approach of disseminating data. In this case, frequently accessed data can be periodically broadcast, while less popular data are provided on demand. We will look at energy efficient solutions as well as effective caching schemes in this context. Finally, the implementation of some of the promising schemes to provide wireless access to courseware is also in our agenda.

Acknowledgements

This work is partially supported by the Research Grant RP960683 funded by the National University of Singapore. The anonymous referees provided very helpful comments that improve the technical quality and literary style of this paper.

References

[1] R. Alonso and S. Ganguly, Query optimization for energy efficiency in mobile environment, in: *Proceedings of the 1993 Workshop on Optimization in Databases*, Aigen, Austria (September 1993).

[2] B.R. Badrinath and P. Sudame, To send or not to send: Implementing deferred transmissions in a mobile host, in: *Proceedings of the 16th International Conference on Distributed Computing Systems* (May 1996) pp. 327–333.

[3] D. Barbara and T. Imielinski, Sleepers and workaholics: Caching in mobile distributed environments, in: *Proceedings of the 1994 ACM–SIGMOD International Conference on Management of Data* (June 1994) pp. 1–12.

[4] J. Cai, K.L. Tan and B.C Ooi, On incremental cache coherency schemes in mobile computing environment, in: *Proceedings of the 13th International Conference on Data Engineering* (April 1997) pp. 114–123.

[5] M.S. Chen, P.S. Yu and K.L. Wu, Indexed sequential data broadcasting in wireless mobile computing, in: *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems* (May 1997).

[6] R. Eager, Advances in rechargeable batteries pace portable computer growth, in: *Proceedings of the 1991 Silicon Valley Personal Computer Conference* (1991).

[7] G.H. Forman and J. Zahorjan, The challenges of mobile computing, IEEE Computer 27(6) (April 1994).

[8] T. Imielinski and B.R. Badrinath, Mobile wireless computing: Challenges in data management, Communications of the ACM 37(10) (October 1994).

[9] T. Imielinski, S. Viswanathan and B.R. Badrinath, Energy efficient indexing on air, in: *Proceedings of the 1994 ACM-SIGMOD International Conference on Management of Data* (June 1994) pp. 25–36.

[10] T. Imielinski, S. Viswanathan and B.R. Badrinath, Power efficient filtering of data on air, in: *Proceedings of the 4th International Conference on Extending Database Technology* (March 1994) pp. 245–258.

[11] 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 2(2) (1997) 115–127.

[12] W.C. Lee and D. Lee, Using signature and caching techniques for information filtering in wireless and mobile environments, Journal of Distributed and Parallel Databases 4(3) (1996) 205–227.

[13] P.V. Argade et al., Hobbit: A high-performance, low-power microprocessor, in: *Proceedings of COMPCON'93* (February 1993) pp. 88–95.

[14] S. Sheng, A. Chandrasekaran and R.E. Broderson, A portable multimedia terminal for personal communications, IEEE Communications Magazine (December 1992) pp. 64–75.

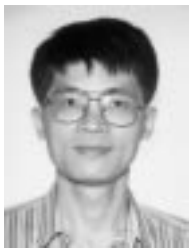
[15] K.L. Tan and J. Cai, Broadcast-based group invalidation: An energy efficient cache invalidation scheme, Information Sciences 100(1–4) (August 1997) 229–254.

[16] K.L. Tan and J.X. Yu, Energy efficient filtering of nonuniform broadcast, in: *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems* (May 1996) pp. 520–527.

[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, in: *Proceedings of the 12th International Conference on Data Engineering* (February 1996) pp. 336–343.

Jun Cai received the B.Sc. and M.Sc. degrees in computer science from FuDan University, PRC, in 1989 and 1992, respectively. He has recently completed his Ph.D. program from the Department of Computer Science, National University of Singapore. He is currently working as a System Specialist at Singapore Engineering Software (a member of Singapore Technologies Group).
E-mail: caijun@comp.nus.edu.sg



Kian-Lee Tan received the B.Sc. (Hons) and Ph.D. degrees in computer science from the National University of Singapore, in 1989 and 1994, respectively. He is currently an Assistant Professor in the Department of Computer Science, National University of Singapore. His major research interests include multimedia information retrieval, wireless information retrieval, query processing and optimization in multiprocessor and distributed systems, and database performance. He has published over 60 conference/journal papers in international conferences and journals. He has also co-authored a tutorial entitled "Query Processing in Parallel Relational Database Systems" (IEEE CS Press, 1994), and a monograph entitled "Indexing Techniques for Advanced Database Systems" (Kluwer Academic Publishers, 1997). Kian-Lee is a member of the ACM and IEEE Computer Society.
E-mail: tankl@comp.nus.edu.sg