International Journal of Research & Development Organisation

ISSN: 4567-7860

JOURNAL OF ELECTRICAL AND ELECTRONINS

# Wireless Mobile Computing using Energy-E Client Caching

Mamta Gothwal

Sukhwinder Kaur

## Abstract

Caching can reduce the bandwidth requirement in a mo-bile computing environment. However, due to battery power limitations, a wireless mobile computer may of-ten be forced to operate in a doze or even totally dis-connected mode. As a result, the mobile computer may miss some cache invalidation reports broadcasted by a server, forcing it to discard the entire cache contents after waking up. In this paper, we present an energy-e cient cache invalidation method, called GCORE, that allows a mobile computer to operate in a discon-nected mode to save battery while still retaining most of the caching bene ts after a reconnection. We present an e cient implementation of GCORE and conduct simulations to evaluate its caching e ectiveness. The results show that GCORE can substantially improve mobile caching by reducing the communication band-width (or energy consumption) for query processing.

## 1 Introduction

Mobile computing enables people with unrestricted mo-bility. It can satisfy people's information needs at any time and in any place. In mobile computing, battery-powered, portable machines can be used by users to query the information/database servers through the wireless communication channels [1, 2, 3, 4]. However, due to limitations on battery technologies, these mobile computers may be frequently disconnected (i.e., pow-ered o ) in order to conserve battery energy.

In general, the bandwidth of the wireless channels is rather limited. Thus, caching of frequently used data in a mobile computer can be an e ective approach to reducing the wireless bandwidth requirement [4]. Once caching is used, a cache invalidation strategy is needed to ensure the data cached in the mobile computers are consistent with those stored in the server. However, if mobile computers must be powered o for energy con-servation, cache consistency may be di cult to enforce.

Depending on whether or not the server maintains the state of the mobile clients' cache, there are two categories of invalidation strategies [4]. In the rst category, the server knows which data are cached by which mobile computers and it is called a stateful server. Once a data item is changed, the server sends inval-idation messages to the clients that are caching that particular data. The server has to locate the clients. But, disconnected mobile clients cannot be contacted by the server. Thus, a disconnection by a mobile com-puter automatically means its cache is no longer valid. Moreover, if a mobile computer wants to relocate, it may have to notify the servers. This implies some re-strictions on the freedom of the mobile computer.

In the second category, the server is not aware of the state of its clients' cache and it is called a stateless server. The server does not even know which mobile computers are currently active. To ensure cache con-sistency, the server simply periodically broadcasts an invalidation report containing the data items that have been updated recently. The mobile clients listen to the broadcast and invalidate their caches accordingly.

In [4], three cache invalidation schemes using di er-ent invalidation reports were proposed for the case of a stateless server. In these three schemes, no attempt was made to check with the server whether or not some of the cached objects are still valid after a reconnection. As a result, when a mobile computer wakes up, it may have to discard the entire cache contents if the discon-nection has been too long. This is because the mobile computer does not know whether or not some of its cached objects have been updated since it became dis-connected. Discarding the entire cache because of a disconnection can be costly as most of the bene ts of caching are lost, especially if most of the cached objects are still valid.

In this paper, we propose an energy-e cient cache in-validation scheme that salvages as many cached objects as possible after a reconnection. Unlike the schemes proposed in [4], which do not check cache validity after a reconnection, our schemes check the cache validity with the server, if necessary, and retain as many valid objects as possible. Since checking cache validity not only requires uplink bandwidth but also consumes bat-tery energy, it must be done e ciently. One simple checking approach is to send all the cached object ids to the server. This is costly because the number of object ids can be large. One possible approach to re-ducing the overhead of validity checking is to do it at a group level, instead of object level. With such a simple grouping scheme, however, the entire group must be invalidated if any of the objects in the group has been updated. In this paper, we propose a new scheme called

Grouping with COld update-set REtention (GCORE). In GCORE, instead of invalidating the entire group, we retain the cold update set of the objects in the group if all the updated objects (most likely belong to the hot update set) have been included in the most recently broadcasted invalidation report.

Here, the hot update set represents the set of ob-jects that are frequently updated by transactions in the server, while the cold update set is the set of objects that are less frequently updated. Obviously, some of the objects referenced by queries and cached in a mobile computer may belong to the hot update set and thus may often be invalidated. Nevertheless, objects that are frequently updated are highly likely to be included in the latest broadcast invalidation report. Therefore, the objects in a group that belong to the cold update set are likely to be retained. GCORE tries to e ciently retain, if possible, the cold update set of a group on the mobile computer.

To evaluate and compare the performance of GCORE with these schemes, we developed an event-driven simulator. The e ectiveness of caching is com-puted as the bandwidth requirement for query process-ing in a mobile computer. Lower bandwidth require-ment means more energy-e cient caching in a mobile computer because it consumes less energy to receive and send messages. The simulation results show that compared with no checking (such as the ones presented in [4]) and simple checking schemes, both GCORE and the simple grouping schemes signi cantly improve the caching e ectiveness by reducing the bandwidth re-quirement for query processing. Moreover, GCORE is more energy e cient than th e simple grouping scheme.

The recent popularity of portable personal comput-ers has attracted a lot of interests in mobile computing. There have been many papers discussing other aspects of supporting mobile computing, including location management, data replication, communication and other system design issues, such as [5, 6, 7, 8, 9, 10, 11]. These citations are by no means exhaustive as many research and development projects are currently being actively conducted to build the national information infrastructure. Our attentions in this paper speci -cally focus on the issues of supporting energy-e cient caching in a wireless mobile computing environment, closely related to [4, 1]. E ective caching and other is-sues for supporting mobile computing are very impor-tant in the future for providing information services to users at any time and in any place.

The rest of the paper is organized as follows. Sec-tion 2 describes the cache invalidation schemes for

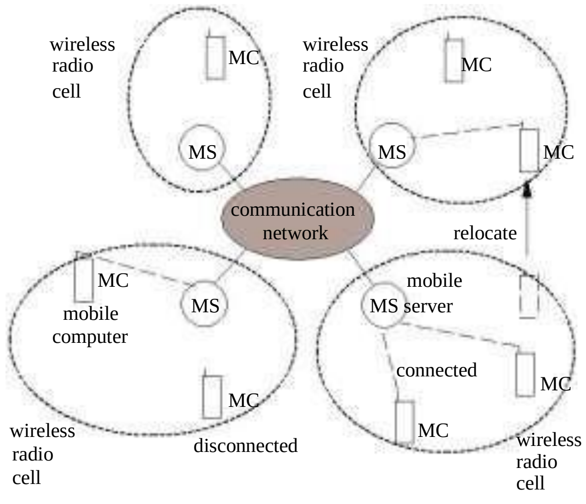


Figure 1: A wireless mobile computing environment.

a stateless server. Section 3 presents the simulation model. Section 4 discusses the simulation results.

## 2 Cache invalidation schemes

Fig. 1 shows a generic wireless mobile computing environment, similar to the one described in [4, 2]. There are multiple wireless radio cells. Each cell has a mobile server that is equipped with wireless communication capability. It stores a complete copy of the database. The mobile servers are connected through a communi-cation network (typically wired). A mobile computer can connect to a server (uplink) through a wireless com-munication channel. It can disconnect from the server by operating in a dose mode (consumes signi cantly less energy) or a power-o mode. It can move from one cell to another cell. The server can communicate with a particular mobile computer through a wireless channel, if the mobile computer is not powered o . We assumed that data are only updated in the servers. Mo-bile computers only read the data and do not update them. To ensure cache consistency, the server periodi-cally broadcasts invalidation reports and all the mobile computers, if active, listen to the reports and invalidate their caches accordingly. Database is assumed to be completely replicated in the mobile servers, so that when a mobile computer moves to another cell, it re-ceives similar invalidation reports.

Frequently referenced objects by queries are cached in a mobile computer. We assumed that the cache at the mobile computer is a nonvolatile memory such as a hard disk. After a disconnection, the content of the cache can still be retrieved. The server keeps track of the object ids that are recently updated and broadcasts an invalidation report every L seconds. The most re-cent invalidation report broadcasted is denoted as IR in this paper. IR consists of the current timestamp T

No-Checking:

```
if ($T_{lb}$ < (T , w L)) invalidate
```

```
            the entire
cache;  else          f
        for 8oi 2 IR f
 if (oi is in the cache) and (ti > tci )
                invalidate oi;
            g
            g
   8q 2 QL
            f
        if all the objects referenced by q are in the cache
process q;          else
 send the missed object ids to the server;       g
   Tlb = T ;
```

Figure 2: Algorithm for query processing using the nochecking scheme.

and a list of $(o_i; t_i)$ such that $t_i > (T ,w L)$, where $o_i$ is an object id and $t_i$ is its corresponding most recent update timestamp, and w is the invalidation broadcast window. Namely, IR contains the update history of the past w broadcast intervals.

## 2.1 No-checking caching scheme

Due to the constraint of limited energy, a mobile com-puter is usually required to operate in a doze mode (not active) or even to be completely disconnected for a prolonged period of time. As a result, a mobile client may miss certain invalidation messages broadcasted by the server. Upon missing an invalidation report, a mo-bile computer may have to discard the entire cache, since it does not know which parts of the cache is valid. This simple scheme is called the no-checking scheme in this paper and is similar to the broadcasting timestamp scheme proposed in [4]. Fig. 2 shows the query processing algorithm for a mobile computer after re-ceiving an invalidation report. Throughout this paper, we assumed that all the queries are batched in a query list, QL, and are not processed until a mobile computer invalidates its cache after receiving an invalidation re-port. Also, the timestamp of the latest invalidation report received, denoted by $T_{lb}$ , is also reliably main-tained so that after a mobile computer wakes up from a disconnection, it knows the timestamp of the latest report that it received.

In Fig. 2, $t^c_i$ is the timetamp of the cached copy of ob-ject $o_i$ . For those objects that are missed in the cache, the object ids are sent to the server and the server then sends back the data and the associated update time-stamps to the mobile computer. They will be again cached in the mobile computer. For the no-checking scheme shown in Fig. 2, if a mobile computer has been disconnected for more than w broadcast intervals, then it must discard the entire cache contents once it re-connects. This can signi cantly increase the wireless bandwidth requirement between the mobile computer and the server as most of the objects subsequently ref-erenced by queries result in cache misses.

## 2.2 Simple-checking caching scheme

Note that even after a long period of disconnection, we may still retain many objects in the cache. This can signi cantly reduce the bandwidth requirement be-cause the mobile computer can use its cached data. In order to retain cached data, we need to identify which cached data are still valid.

There are several approaches to identifying valid cache entries after a disconnection. They involve dif-ferent trade-o s. To accurately identify the valid cache entries, a mobile computer can send all the cached ob-ject ids and their corresponding timestamps. However, this requires a lot of uplink bandwidth as well as bat-tery energy. On the other hand, the mobile computer can send group ids and group timestamps; the validity can be checked at the group level. This reduces the uplink bandwidth requirement. But, a single object updated essentially invalidates the entire group. As a result, the amount of cached objects salvaged after a reconnection may be quite small. GCORE combines the advantages of both schemes by salvaging as many cached objects as possible and consumes as little uplink cost as possbile.

Note that it is su cient to just send $T_{lb}$ with object ids or group ids to the server. The object timestamps or group timestamps need not be sent. This is because once a mobile computer processes a new IR, all the valid cache entries can be viewed as being timestamped at that moment. The server can check the validity of an object or a group of objects based on $T_{lb}$ .

For the simple checking scheme, only $T_{lb}$ and all the object ids that are still not yet invalidated by a mobile computer are sent to the server. The server then com-pares $T_{lb}$ with the object update timestamp stored in the server and sends a validity report back to the mo-bile computer. This validity report can simply be a bit vector, with each bit indicating yes or no for the cor-responding object. To process queries, a mobile com-puter rst invalidates its cache according to IR. Then, if the mobile computer just wakes up from a disconnection and $T_{lb} < (T , w L)$, it sends the cached objects that are not yet invalidated to the server for validity checking. After receiving the validity report from the server, the mobile computer then processes the queries.

## 2.3 Simple-grouping caching scheme

In order to reduce the uplink communication costs for validity checking, the database can be partitioned into a number of groups and a mobile computer checks its cache validity at the group level. The grouping function can be simply a modulo function. Or it can be di er-ent for di erent types of objects. Whatever grouping function is chosen, it must be agreed upon between the server and the mobile computer. Data objects belong-ing to a group may or may not be in a mobile cache. But, the group validity checking is not a ected by the grouping function or by the fact that some of the group objects are not in a mobile cache. The grouping of ob-jects is used only for cache validity checking after a mo-bile computer reconnects, it is not used by the server for broadcasting invalidation reports.

The query processing algorithm for simple grouping is similar to the one for simple checking, except that it sends much less information to the server and as a re-sult requires much less uplink bandwidth. It is the same as GCORE, to be described next, in sending the group ids and $T_{lb}$ to the server for group validity checking. However, it di ers from GCORE in the way the server determines whether or not a group is valid. In the sim-ple grouping scheme, if any object within a group is updated after $T_{lb}$, then the entire group is considered to be invalid. Thus, the amount of cached objects that can be retained may be small after a reconnection, re-sulting in a large amount of uplink and downlink costs due to cache misses from future query processing.

## 2.4 Grouping with cold update-set re-tention (GCORE)

To avoid discarding of a group, we present a group-ing with cold update-set retention scheme to improve the caching e ectiveness. Similar to the simple group-ing scheme, the server partitions the database into a number of groups. So, it incurs relatively small uplink costs for validity checking. However, unlike the simple grouping scheme, GCORE tries to salvage a group so that the future downlink costs due to cache misses can be signi cantly reduced.

In addition to grouping, the server also dynamically identi es hot update set that has been updated in a group and excludes it from the group when checking the group's validity. If all the updated object ids in a group have already been included in IR, these objects should be invalidated by the mobile computer when it receives IR. With the hot update set excluded from a group, the server can conclude that the objects that are not updated in the group can be retained in the cache and validate the rest of the group. This scheme is therefore referred to as grouping with cold update-set retention in this paper. GCORE is energy e cient because it incurs both low uplink costs for validity checking and low downlink costs as it retains more cached objects.

```
struct group_table_entry
{
  double time; int
 total_wW;
  struct pair *uplist; }
group_table[];
```

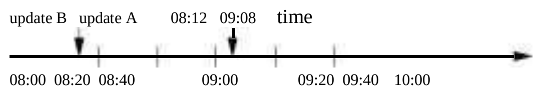
Figure 3: Data structure for group update history.

To facilitate a mobile computer to salvage many of its local cache contents without incurring high uplink costs, the server needs to maintain a more sophisti-cated data structure for the group update history. It maintains for each group the object update history of the past W broadcast intervals (W w), consisting of a list of object ids and their most recent update timestamps, and the most recent update time of the group. This group update history is maintained in group table[] (see Fig. 3 for its de nition). In ad-dition, it also maintains the number of distinct objects that were most recently updated between (T , W L) and (T ,w L) to speed up the group validity checking.

As an example, Fig. 4 shows a snapshot of a group table entry, group table[1]. In Fig. 4, group 1 con-tains objects A; B; C; D; E; and F. A broadcast in-terval is 20 minutes, w = 3 and W = 6. Object A was updated at 09:08 and object B was updated at 08:12. The server keeps track of the update history of the past 6 broadcast intervals. Thus, both objects A and B and their update timestamps are maintained in group table[1].uplist. It also shows that the most recent update to this group was at time 09:08. Since IR contains the update history of the past 3 broadcast intervals, the number of distinct objects that were most recently updated between (T ,W L) and (T ,w L) is 1. Namely, group table[1].tot wW at the moment is 1 (the number of distinct objects most recently up-dated between 08:00 and 09:00 is
1).

For every update to objects, the server updates group table[]. As a continuing example, Fig. 5 shows the changes to group table[1] at time 10:19 after ob-ject A and E were updated at 10:02 and 10:16, respec-tively. The update timestamp of object A is changed to the most recent update time of 10:02 and a new pair (E, 10:16) is inserted into the update_list pointed to by group table[1].uplist. Of course, the group update time is also changed to 10:16 to re ect the most recent update to the group.

Each time the server broadcasts an invalidation re-port, it also updates group table[]. For every group, the server removes the pairs in group table[].uplist that have update times less than T , W L. This would eliminate any objects that were updated before T , W L, and limit the amount of history that the server must maintain under GCORE. In addition, the



group_table[1]:

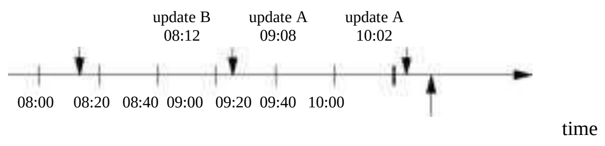
time = 09:08; total_wW = 1; uplist-->(A, 09:08)-->(B, 08:12)-->NULL;

Group[1] = {A, B, C, D, E, F}; w = 3; W = 6

Current time = 10:00

T = 10:00

Figure 4: Example of a group table[].



10:16
update E

group_table[1]:

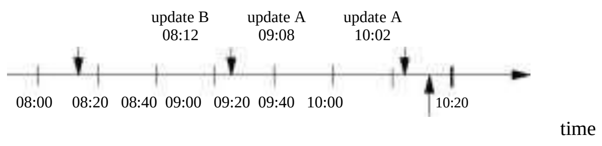
time = 10:16;
total_wW = 1;
uplist-->(E, 10:16)-->(A, 10:02)-->(B, 08:12)-->NULL;

Group[1] = {A, B, C, D, E, F}; w = 3; W = 6

Current time = 10:19

T = 10:00

Figure 5: Changes to group table[] due to an update.



10:16
update E

group_table[1]:

time = 10:16;
total_wW = 0;
uplist-->(E, 10:16)-->(A, 10:02)-->NULL;

Group[1] = {A, B, C, D, E, F}; w = 3; W = 6

Current time = 10:20 T
= 10:20

Figure 6: Changes to group table[] due to a new broad-cast.

server also updates group table[].total wW, which keeps the number of distinct objects that were most recently updated between T , W L and T , w L. The maintenance of group table[].total wW is for a fast group validity checking. If no object was most recently updated during T , W L and T , w L and $T_{lb}$ > (T , W L), it means if there are any ob-jects in the group updated since $T_{lb}$ , their ids have been included in IR. Notice that in GCORE the cache va-lidity checking is done after the mobile computer rst invalidates its cache based on IR, the most recent in-validation report. As a result, when a just-woke-up mobile computer sends a group validity checking to the server, it can ensure that those recently updated hot data have already been invalidated in the mobile computer's cache.

Continuing the example of Fig. 4 and 5, we show the changes to group table[1] at time 10:20 when a new IR is broadcasted in Fig. 6. Object B is now discarded from the update history since group table[1] only keeps track of the update history between 08:20 and 10:20 now. Also, group table[1].tot wW becomes 0 now since object A has been again updated at 10:02. Thus, the number of distinct objects most recently up-dated between 08:20 and 09:20 is zero.

The server checks the validity of a group by exam-ining whether or not all the objects updated since the mobile computer becomes disconnected have been in-cluded in the latest invalidation report IR. If yes, then the group can be retained by the mobile com-puter. Otherwise, the entire group is invalid. This can be achieved by rst checking if group table[].time < $T_{lb}$. If yes, the group is valid since the group is last updated before the mobile computer becomes disconnected. If not, the server further checks if group table[].total wW equals to 0 and $T_{lb}$ > (T ,
W L). If yes, this group is also valid since the up-dated object ids are all included in IR. Otherwise (either group table[].tot wW is greater than zero or $T_{lb}$ < (T , W L)), this group is invalid.

## 3 Simulation model

In order to evaluate the performance of GCORE, an eventdriven simulator was developed to model a server and a mobile computer. In the simulation model, we assumed that database objects are only updated in the server by transactions and queries are read-only and are processed in the mobile computer. If the referenced data objects are not cached in the mobile computer, it sends the requested object ids to the server and the server sends back the object data. The e ectiveness of caching is measured by the communication bandwidth requirement for query processing. This communication requirement includes the receiving of the broadcast in-validation reports, the uplink communication for va-lidity checking and asking for missed objects, and the

| Notation | De nition (Default values) |
|---|---|
| D | server database size (100,000 objects) |
| B | mobile cache size (5000 objects) |
| u | server transaction arrival rate, Poisson in terarrival time (0.01 jobs/sec) |
| q | mobile query arrival rate, Poisson interar rival time (0.1 jobs/sec) |
| U | mean objects updated by a transaction (5) |
| Q | mean objects referenced by a query (20) |
|  | reference skew by transactions (90%-10%) |
| G | group size (100 objects) |
| w | window for broadcast invalidation (10 intervals) |
| W | update history maintained (60 intervals) |
| L | length of a broadcast interval (20 seconds) |
| O | object size (256 bytes) |
| $O_{id}$ | object id size (64 bits) |
| $G_{id}$ | group id size (64 bits) |

| | |
|---|---|
| T | timestamp of the current broadcast invali dation report |
| $T_{lb}$ | the timestamp of the latest broadcast in validation report received by a mobile com-puter before it went to sleep |
| $P_{disc}$ | conditional probability of a disconnection in the next broadcast interval given that a mobile computer is active now (0.2) |
| $L_{disc}$ | mean disconnection length (200 seconds) |

Table 1: System and workload parameters.

downlink communication for sending the data to the mobile computer. High bandwidth requirement means less e ective caching and more energy consumption.

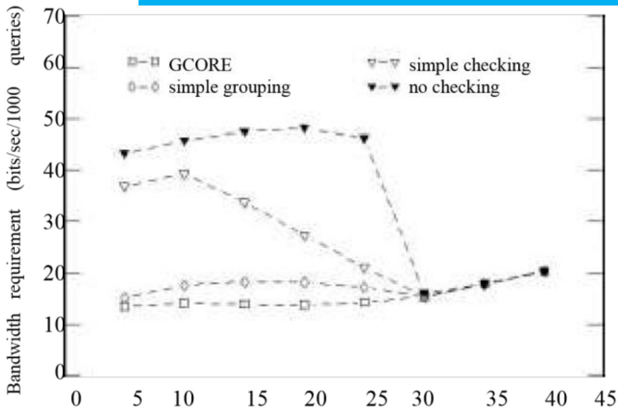
A total of D total objects are in the database. Of the D objects, portion of them are hot update set, while (1, ) portion of them are cold update set. Data in the hot update set are randomly chosen from the D objects. The number of objects updated by a trans-action is uniformly distributed between U=2 and 3U=2 objects, where U is the mean. Of the data objects up-dated by a transaction, fraction of them are from the hot update set, and the rest from the cold update set. Update transaction arrival is a Poisson process with rate $_u$ .

In order to focus on the cache invalidation e ect, we assumed that a cache miss is resulted only from inval-idation. It does not result from a query referencing an object that is replaced by another object. In other words, we assumed that all the queries in a mobile com-puter reference a xed subset of objects that are ini-tially cached. The cache size is B objects. These B objects are randomly chosen from the D objects in the database. Some of the cached objects may be invali-dated because they have been updated by transactions in the server.
The objects referenced by a query are randomly chosen from these B objects and the number of objects referenced by a query is uniformly distrib-uted between Q=2 and 3Q=2, where Q is the mean. The probability of a mobile computer becoming dis-connected in the next broadcast interval given that it is active now is denoted as $P_{disc}$. The length of discon-nection is uniformly distributed between $L_{disc}$=2 and 3$L_{disc}$=2, where $L_{disc}$ is the mean. When a mobile computer is active, query interarrival times are expo-nentially distributed with mean 1= $_q$ seconds. Queries are batched in QL and are not processed until a mo-bile computer receives a broadcast invalidation report. We accumulated the communication costs for query processing in a mobile computer for a period of time (50,000 broadcast intervals), and then compute its av-erage bandwidth requirement. For the computation of communication costs, the size of an object id is $O_{id}$ bits and the size of each object is O bytes. The size of a group id is $G_{id}$ . In the simulations, we assumed that O = 256 bytes, $O_{id}$ = 64 bits and $G_{id}$ = 64 bits. The size of a timestamp is 256 bits. Notation and its de nition for all the simulation parameters are summarized in Table 1. The default values used in the simulations, if not otherwise speci ed, are included in the parenthe -ses.

Note that broadcast invalidation requires communication bandwidth every L seconds for all schemes even when a mobile computer is disconnected. However, it consumes energy for a mobile computer only when it is active. For the no-checking scheme, the communi-cation costs for query processing can be rather high both for uplink and downlink costs resulted from cache misses, especially if the disconnection length is longer than the broadcast invalidation window. For all the caching schemes, there are uplink costs due to a mo-bile computer requesting missed objects in its cache. For the other checking schemes, the uplink costs for cache validity checking are added to the total band-width costs.

## 4 Simulation results

We compute the communication costs of a mobile com-puter, including the costs of receiving the broadcast invalidation reports, the uplink costs of checking the cache validity and requesting missed objects, and the downlink costs of the validity report and the data ob-jects due to cache misses. The bandwidth requirement is the average of the communication costs over 50,000 broadcast intervals. Since the bandwidth requirement would be smaller if the mobile computer is often dis-connected, we plotted the bandwidth requirement for 1000 queries. By doing this, the true e ectiveness of caching is manifested in bandwidth requirement. Un-less otherwise speci ed, the default values for most of the simulations are provided in Table 1.



($W$ = 60; $G$ = 100; $P_{disc}$ = 0.1; $L_{disc}$ = 400 seconds; □±□ = 90%±10%; u = 0.01 jobs/sec)

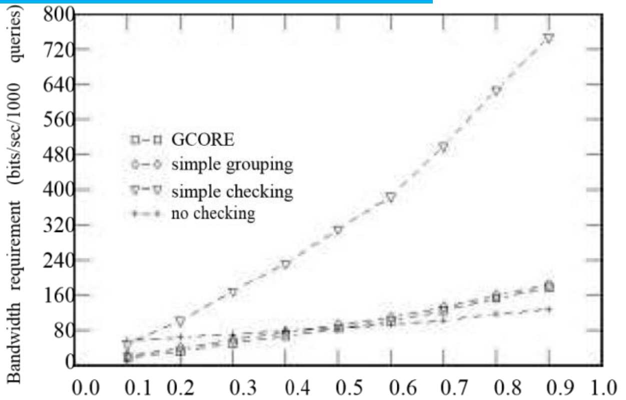
Figure 7: Impact of broadcast invalidation window size.

## 4.1 Four schemes

We examined the impact of w, relative to $L_{disc}$ , on the bandwidth requirement. Fig. 7 shows the total bandwidth requirement per 1000 queries for the four di erent schemes. In this experiment, the mobile com-puter was disconnected relatively infrequently, namely $P_{disc}$ = 0:1. (Sensitivity analysis on $P_{disc}$ will be pre-sented next.) The mean disconnection length $L_{disc}$ was 400 seconds, which is equivalent to 20 broadcast inter-vals. Since disconnection length was uniformly distrib-uted between $L_{disc}$=2 and 3$L_{disc}$=2, a mobile computer may power o for a period of between 10 and 30 broad- cast intervals. For GCORE and the simple grouping schemes, the group size was 100 objects.

Because the broadcast invalidation cost is propor-tional to the number of objects included in the inval-idation report, it increases as w increases for all four schemes. However, if w is greater than the disconnec-tion length, all cached objects can be validated with the invalidation report IR, and no checking is ever needed. Therefore, if w 30 broadcast intervals, all four schemes require exactly the same communication bandwidth. This is because the maximum disconnec-tion length is 30 broadcast intervals. For the cases where w < 30 broadcast intervals, both GCORE and simple grouping require signi cantly less bandwidth than simple checking and no checking schemes. Obvi-ously, validity checking helps retain some of the cached objects and thus substantially reduces bandwidth re-quirements. Even with a simple checking scheme where the uplink cost can be substantial, the overall cache ef fectiveness can still be improved.

If the bene ts of retaining cached objects are not re-alized by queries, e.g., a mobile computer may be dis-connected most of the time, it may not pay o to per-form validity checking, especially for the simple check-ing scheme. Here, we examine the impact of $P_{disc}$,
requirement for query processing.



($w$ = 10; $W$ = 60; $L_{disc}$ = 400 seconds; □±□ = 90%±10%; □u = 0.01 jobs/sec)

Figure 8: Impact of disconnection probability on the band-width the conditional probability of a mobile computer disconnected in the next broadcast interval given that it is active now. Fig. 8 shows the total bandwidth re-quirement for the four di erent caching schemes. For this experiment, w was 10 broadcast intervals (or 200 seconds) and $L_{disc}$ was 400 seconds. Namely, a dis-connection can last for between 200 seconds and 600 seconds. Thus, the rst invalidation report a mobile computer receives after a reconnection does not con-tain enough information for validating cached objects. It must either discard the entire cache, as in the case of no checking scheme, or check the
validity of cached objects with the server. As indicated from Fig. 8, as $P_{disc}$ increases the over-head of cache validity checking starts to outweigh the bene t of retaining cached objects. This is particularly true for the case of simple checking, as it needs high uplink bandwidth to send all the object ids. In this gure, simple checking is bene cial only for the case of very small $P_{disc}$. Fig. 8 clearly illustrates the impor-tance of reducing the uplink communication overhead for cache validity checking. Both GCORE and simple grouping have much smaller uplink costs, and as a re-sult can still be better than the no checking scheme for a larger $P_{disc}$ . But, as a mobile computer is discon-nected most of the time, it does not pay to check the validity of cached objects since they are not likely to be reused anyway. In such cases, it is better to just use the no checking scheme.

## 4.2 GCORE and simple grouping

For both simple grouping and GCORE, group size is an important design parameter. With a larger group size, less uplink bandwidth is needed. However, it be-comes more likely that the entire group or most of it may be invalidated since more objects in the group are

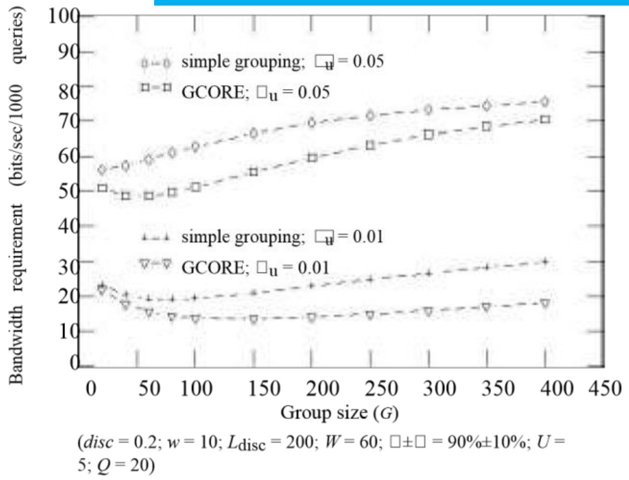


Figure 9: Impact of group size on bandwidth.

likely to be updated. Fig. 9 shows the bandwidth requirements of GCORE and simple grouping for vari-ous group sizes and update rates. In general, as group size increases, the uplink cost decreases but the down-link cost increases. Thus, the total bandwidth rst de-creases and then increases as group size increases. For the cases with low update rates (such as those with $_u$ = 0:01 in Fig. 9), the advantage of GCORE over simple grouping increases as group size increases. How-ever, it may not be true for the cases with high update rates. This is because GCORE can retain the cold up-date set of a group only if the updated objects of the group are all captured in the most recent invalidation report. If a group contains a large number of objects and the update rate is high, then it is less likely that all the updated objects will be captured in IR.

## 5 Summary

We have presented an energy-e cient caching scheme, called grouping with cold update-set retention GCORE, that allows a mobile computer to disconnect for saving energy, but still retains most of the caching bene ts. An e cient implementation of GCORE was presented in the paper. It uses a simple data structure to facilitate the dynamic exclusion of recently updated objects (likely to belong to the hot update set) from a group so that the rest of the group (likely to belong to the cold update set) can be retained in the cache. Upon waking up, a mobile computer checks its cache validity with the server at a group level to save uplink costs. The server determines that a group is valid if all the recently updated objects have already been included in the most recently broadcasted invalidation report.

Simulations were conducted to evaluate the perfor-mance of GCORE. We compared GCORE with a no checking scheme, a simple checking scheme and a sim-
ple grouping scheme. The results show that, compared with no checking and simple checking, both simple grouping and GCORE requires much less bandwidth for processing queries, particularly if a mobile computer is occasionally disconnected for a long period of time and most of the data objects are infrequently updated. Lower bandwidth requirement also consumes less en-ergy and thus more energy e cient.

## References

[1] T. Imielinski and B. R. Badrinath, \Querying in highly mobile distributed environments," in Proc. of Very Large Data Bases, pp. 41{52, 1992.

[2] T. Imielinski and B. R. Badrinath, \Mobile wire-less computing," Communications of the ACM, vol. 37, no. 10, pp. 18{28, Oct. 1994.

[3] T. Imielinski, S. Viswanathan, and B. R. Badri-nath, \Energy e cient indexing on air," in Proc. of ACM SIGMOD, pp. 25{36, 1994.

[4] D. Barbara and T. Imielinski, \Sleepers and workaholics: Caching in mobile distributed environments," in Proc. of ACM SIGMOD, pp. 1{12, 1994.

[5] R. H. Katz, \Adaptation and mobility in wireless information systems," IEEE Personal Communications, pp. 6{17, First Quarter 1994.

[6] A. Acharya and B. R. Badrinath, \Checkpointing distributed applications on mobile computers," in Proc. of Int. Conf. on Parallel and Distributed Information Systems, pp. 73{80, 1994.

[7] P. Krishna, N. H. Vaidya, and D. K. Pradhan, \Location management in distributed mobile en-vironments," in Proc. of Int. Conf. on Parallel and Distributed Information Systems, pp. 81{88, 1994.

[8] Y. Huang and O. Wolfson, \Object allocation in distributed databases and mobile computers," in Proc. of Int. Conf. on Data Engineering, pp. 20{ 29, 1994.

[9] Y. Huang, P. Sistla, and O. Wolfson, \Data replication for mobile computers," in Proc. of ACM SIGMOD, pp. 13{24, 1994.

[10] R. Alonso and H. Korth, \Database system issues in nomadic computing," in Proc. of ACM SIG-MOD, pp. 388{392, 1993.

[11] B. R. Badrinath and T. Imielinski, \Replication and mobility," in Proc. of the 2nd Workshop on the Management of Replicated Data, 1992.