RC 19877 (87945) 12/15/94
Computer Science

IBM Research Report

Energy-Efficient Caching for Bandwidth-Limited Wireless Mobile Computing

Kun-Lung Wu, Philip S. Yu and Ming-Syan Chen

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Energy-Efficient Caching for Bandwidth-Limited Wireless Mobile Computing

Kun-Lung Wu, Philip S. Yu and *Ming-Syan Chen*

IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Abstract

Caching of frequently accessed data in a wireless mobile computer can significantly reduce the bandwidth requirement in a mobile computing environment. However, due to battery power limitations, a wireless mobile computer may often be forced to operate in a doze or even totally disconnected mode. As a result, the mobile computer may miss some cache invalidation reports broadcasted by a server and it may have to discard the entire cache contents after waking up, giving up most of the caching benefits. In this paper, we present an energy-efficient cache invalidation method that allows a mobile computer to operate in a disconnected mode to save battery while still retaining most of the caching benefits after a reconnection. The new cache invalidation scheme is called grouping with cold update-set retention (GCORE). Upon waking up, a mobile computer checks its cache validity with the server. To reduce the bandwidth requirement for validity checking, data objects are partitioned into groups. However, instead of simply invalidating a group if any of the objects in the group has been updated, GCORE retains the cold update set of objects in a group if the hot update set has been included in the most recently broadcasted invalidation report. We present an efficient implementation of GCORE and conduct simulations to evaluate its caching effectiveness. The results show that GCORE is most energy efficient because it requires least communication bandwidth for query processing on a mobile computer under a wide range of operating conditions.

Index Term: mobile computing, energy-efficient caching, cache invalidation, wireless computing.

1 Introduction

In a mobile computing environment of the very near future, a large number of battery-powered, portable machines can be used by many users to query the information and/or database servers from different places through the wireless communication channels [7, 8, 9, 4]. These mobile computers may often be disconnected (i.e., power off) for perhaps prolonged periods of time in order to conserve battery energy. They may also frequently relocate from one cell to another and connect to different data servers at different times. This entire mobile computing information infrastructure would enable the users with unrestricted mobility, and satisfy their information needs at any time and in any place.

In general, the bandwidth of the wireless channels is very limited. Thus, caching of frequently used data in a mobile computer can be an effective technique to reduce the wireless bandwidth requirement [4]. Once caching is used, however, a cache invalidation strategy is needed to ensure the data cached in the mobile computers are consistent with those stored in the server.

Depending on whether or not the server maintains the state of the mobile clients' cache, Barbara and Imielinski classified cache invalidation strategies into two categories [4]. In the first category, the server knows which data are cached by which mobile computers and is called a *stateful* server. Once a data item is changed, the server sends invalidation 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 computer 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 restrictions on the freedom of the mobile computer.

In the second category, the server does not know about the state of its clients' cache and is called a *stateless* server. The server does not even know which mobile computers are currently active. To ensure cache consistency, 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 different invalidation reports were proposed for the case of a stateless server. However, in these three schemes if a mobile computer is disconnected for a long period of time for saving energy, it may miss some invalidation broadcasts from the server. As a result, when it wakes up, it may have to discard the entire cache contents. This is because the mobile computer does not know whether or not some of its cached objects have been updated since it became disconnected. Discarding the entire cache because of a disconnection is very costly as most of the benefits of caching to save bandwidth requirement are lost. It is also unnecessary because most of the cached objects in the mobile computer may be still valid and can be retained even after a disconnection.

In this paper, we propose an energy-efficient cache invalidation scheme that salvages most of the cached objects even after a prolonged disconnection. Instead of simply discarding the entire cache, the mobile computer can check its cache validity with the server. Since checking cache validity not only requires uplink bandwidth but also consumes battery energy, it is done in a group level, instead of object level, in order to reduce its overhead. However, instead of simply invalidating a group if any of the objects in a group has been updated, we retain the the cold update set of objects in a group if the hot update set has been included in the most recently broadcasted invalidation report. This new scheme is called Grouping with COld update-set REtention (GCORE). Note that the hot (cold) set of update transactions on a server may be different from the hot (cold) reference set of queries on the mobile computer. GCORE tries to retain the cold update set, which may be the hot reference set of queries on a mobile computer. Upon waking up, the mobile computer invalidates its cache based on the first broadcast invalidation report and then checks the validity of the rest of the cache in a group level. The server maintains a simple group update history that dynamically excludes the hot update set. As a result, the server can quickly determine whether or not the rest of a group of objects can be retained by any mobile computer without tracking the mobile computer's cache state. In other words, certain data objects belonging to a group may or may not be in a mobile computer's cache, yet the validity of the group can be efficiently determined by the server.

Detailed descriptions of GCORE along with a simple checking and a simple grouping schemes are presented in this paper. For the simple checking scheme, the mobile computer simply asks the server about its cache validity by sending all the cached object ids upon waking up. This scheme requires a substantial uplink bandwidth as the number of cached objects can be large. For the simple grouping scheme, the mobile computer asks the server about its cache validity by sending group ids. It reduces the uplink bandwidth requirement. However, once any of the objects in a group has been updated, the entire group is considered to be invalid. This can result in a high downlink bandwidth requirement for the server to send objects to the mobile computer due to subsequent cache misses. To evaluate and compare the performance of GCORE with these simple schemes, we developed an event-driven simulator. The effectiveness of caching is computed as the bandwidth requirement for query processing in a mobile computer, including the uplink and downlink communication costs. Lower bandwidth requirement means more energy-efficient caching in a mobile computer because it consumes less energy. 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 significantly improve the caching effectiveness by reducing the bandwidth requirement for query processing. Furthermore, GCORE requires substantially less energy consumption than the simple grouping scheme does, especially if the group size is large.

The recent popularity of portable personal computers and personal digital assistants 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 [10, 1, 12, 6, 5, 2, 3]. 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 specifically focus on the issues of supporting energy-efficient caching in a wireless mobile computing environment, closely related to [4, 7]. Effective caching and other issues for supporting mobile computing are very important in the future for providing information services to users at any time and in any place.

Mobile caching in some respect is similar to disconnected operations in the CODA file system [11]. However, they differ significantly in the communication and power constraints. In the CODA file system, there is less constraint in communication bandwidth and no concern for power consumption since once a disconnected workstation reconnects to the network it is back to the normal networked system. However, in wireless mobile computing, communication bandwidth and battery power are severe constraints based on today's technologies. Therefore, to ensure cache consistency in a wireless mobile computing environment, both power consumptions in mobile computers and wireless bandwidth requirements must be taken into consideration, as they are in this paper.

The paper is organized as follows. Section 2 describes the cache invalidation schemes for a stateless server. Section 3 presents the simulation model. Section 4 discusses the simulation results. Finally, Section 5 summarizes the paper.

2 Cache invalidation schemes

In this section, we describe the details of GCORE for energy-efficient mobile caching. For comparison purposes, we also present a no checking scheme, a simple checking scheme and a simple grouping scheme.

Fig. 1 shows a generic wireless mobile computing environment, similar to the one described in [4, 8]. There are multiple wireless radio cells. Each cell has a mobile server station that is equipped with

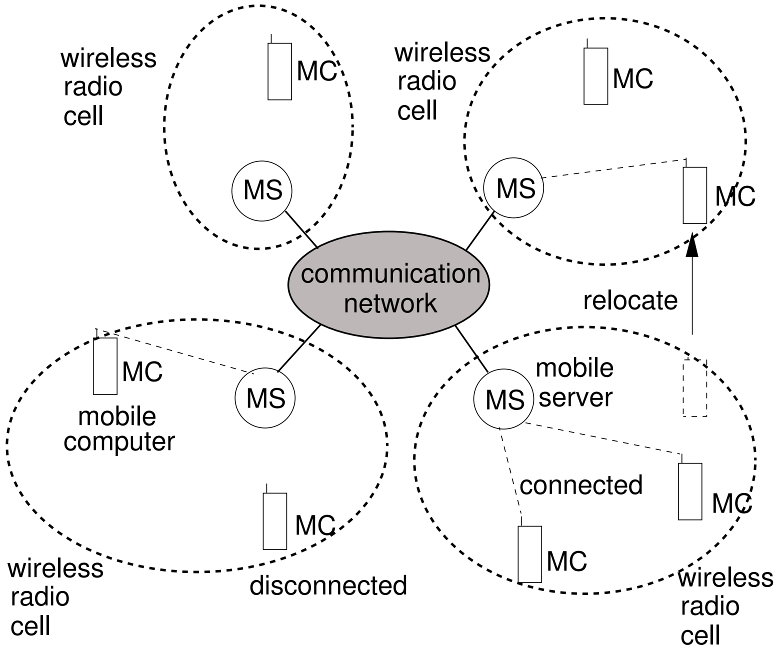


Figure 1: A wireless mobile computing environment.

wireless communication capability. The mobile server station is simply called mobile server in this paper and it stores a complete copy of the database. The mobile servers are connected through a communication network (typically wired). A mobile computer can connect to a server (uplink) through a wireless communication channel. It can disconnect from the server by operating in a dose mode (consumes significantly less energy) or a power-off mode. It can move from one cell to another cell. The server can communicate with a particular mobile computer (downlink) also through a wireless channel if the mobile computer does not power itself off. We assumed that data are only updated in the servers. Mobile computers only read the data and do not update them. To ensure cache consistency, the server periodically 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 receives similar invalidation reports.

Frequently accessed data objects by queries are cached in a mobile computer's cache. Here, we assumed that the mobile cache is nonvolatile memory such as the hard disk on the mobile computer. The cache contents can be retrieved after a power off. The server keeps track of the object ids that are recently updated and broadcasts an invalidation report every L seconds. The most recent invalidation report broadcasted is denoted as IR in this paper. In IR, it contains the current timestamp T and a list of (o_i, t_i) such that $t_i > (T - w \times L)$, where o_i is an object id and t_i is

Query Processing Using the No-Checking Scheme:

```
if (T_lb < (T − w × L))
   invalidate the entire cache;
else
   {
      for ∀o_i ∈ IR
         {
            if (o_i is in the cache) and (t_i > t_i^c)
               invalidate o_i;
         }
   }
∀q ∈ QL
   {
      if all the objects referenced by q are in the cache
         process q;
      else
         send the missed object ids to the server;
   }
T_lb = T;
```

Figure 2: Algorithm for query processing using the no-checking caching scheme.

its corresponding most recent update timestamp. In other words, IR contains the update history of the past w broadcast intervals. A mobile computer listens to the broadcast and invalidates its cache contents accordingly.

2.1 No-checking caching scheme

Due to the constraint of limited energy in a mobile computer, it 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. Once missing an invalidation message, a mobile client may have to discard the entire cache contents because some of them may have become obsolete and it does not know which. 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 receiving 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 report. Also, the timestamp of the latest invalidation report received, denoted by T_{lb}, is also reliably maintained so that after a mobile computer wakes up from a disconnection, it knows what the timestamp of the latest received report is.

In Fig. 2, t_i^c is the timetamp of the cached copy of object 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 timestamps to the mobile computer. They will be again cached in the mobile computer. For the no-checking scheme shown in Fig. 2, a mobile computer often must be turned off in order to save battery energy. As a result, it usually must start with an empty cache each time it reconnects with the server after an energy-saving disconnection. This can significantly increase the wireless bandwidth requirement between the mobile computer and the server as most of the objects referenced by queries result in cache misses.

2.2 Motivation for checking cache validity

Note that even after a long period of disconnection, if most of the cached data still have not been updated by the server, they should be retained in the cache after the mobile computer turns on. This can significantly reduce the bandwidth requirement because the mobile computer can use its local cached data. But, in order to retain cached data, an approach is needed to identify which cached data are still valid and which should be purged.

There are several possible approaches to identifying valid cache entries after a disconnection, ranging from the most straightforward to the more sophisticated such as GCORE proposed in this paper. They involve different trade-offs. To accurately identify the valid cache entries, a mobile computer can simply send to the server all the cached object ids and their corresponding timestamps. However, this requires a big demand on the uplink bandwidth as well as battery energy to send the large number of cached objects. On the other hand, the mobile computer can send group ids and group timestamps and the validity can be checked in the group level. This reduces the uplink bandwidth requirement. But, a single object updated in a group essentially invalidates the entire group. As a result, the amount of cached objects salvaged after a disconnection may be quite small, resulting in many future cache misses. GCORE tries to combine the advantages of both schemes by salvaging as many cached objects as close to the simple checking scheme while consumes as little uplink costs as close to the simple grouping scheme.

Note that a straightforward way of checking cache validity, either in an object level or a group level, is to simply send the object or group ids and their timestamps to the server. This is typically used for object version checking in a general distributed system. However, such a scheme can be unnecessarily costly in a wireless mobile computing environment because it requires a large uplink communication costs as both object ids and timestamps need to be sent. Instead, it is sufficient to just send T_{lb} with object ids or group ids to the server. 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}, the timestamp of the latest broadcast report received by a mobile computer. Therefore, in all the three checking schemes discussed in this paper, the object timestamps or the group timestamps are not sent during validity checking.

2.3 Simple-checking caching scheme

For the simple checking scheme of this paper, 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 compares T_{lb} with the object update timestamp stored in the server and sends a validity report back to the mobile computer. This validity report can simply be a bit vector, with each bit indicating yes or no for the corresponding object. Fig. 3 shows the algorithm for query processing after a mobile computer receives an invalidation report. It first invalidates its cache according to IR. Then, if the mobile computer just wakes up from a disconnection and $T_{lb} < (T - w \times L)$, it sends the cached objects that are not yet invalidated to the server for validity checking. Note that since IR contains the object ids that were updated during the past w broadcast intervals, no validity checking is necessary if $T_{lb} \geq (T - w \times L)$. After receiving the validity report from the server, the mobile computer then processes the queries. As mentioned above, the uplink communication overhead can be very large for the simple checking scheme, especially if the cache size is large and the mobile computer is frequently disconnected.

2.4 Simple-grouping caching scheme

In order to reduce the uplink communication costs for validity checking, the entire database can be partitioned into a number of groups and a mobile computer checks its cache validity in the group level. The grouping function can be simply a modulo function, resulting in equal-sized groups. Or it can be different for different types of objects, resulting in different group sizes. Whatever grouping function is chosen, it must be agreed upon between the server and the mobile computer. Data objects belonging to a group may or may not be in a mobile cache. But, the group validity checking is not affected by the grouping function or by the fact that some of the group objects are not in a mobile cache.

The 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 result requires much less uplink bandwidth. Fig. 4 shows the query processing algorithm using simple grouping scheme. It is the same as GCORE,

Query Processing Using the Simple-Checking Scheme:

```
for ∀o_i ∈ IR
   {
      if (o_i is in the cache) and (t_i > t_i^c)
         invalidate o_i;
   }
if (T_lb < (T − w × L))
   {
      send T_lb and all the object ids that are not yet invalidated to the server;
      wait for the validity report from the server;
      invalidate the cache according to the validity report;
   }
∀q ∈ QL
   {
      if all the objects referenced by q are in the cache
         process q;
      else
         send the missed object ids to the server;
   }
T_lb = T
```

Figure 3: Algorithm for query processing using the simple-checking scheme.

to be described next, in sending the group ids and T_{lb} to the server for group validity checking. However, it differs from GCORE in the way the server determines whether or not a group is valid. In the simple 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, resulting in large amount of downlink costs due to cache misses from future query processing.

2.5 Grouping with cold update-set retention (GCORE)

To avoid unnecessarily discarding of a group, we present a grouping with cold update-set retention scheme to improve the caching effectiveness for wireless mobile computing. Similar to the simple grouping scheme, the server partitions the entire database objects 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 if possible so that the future downlink costs due to cache misses can be significantly reduced. The query processing algorithm for a mobile computer using GCORE is shown in Fig. 4.

In addition to grouping, the server also dynamically identifies hot update set that has been updated recently in a group and excludes it from the group when checking the validity of the group. If all the

Query Processing Using Simple-Grouping and GCORE schemes:

```
for ∀o_i ∈ IR
   {
      if (o_i is in the cache) and (t_i > t_i^c)
         invalidate o_i;
   }
if (T_lb < (T − w × L))
   {
      send T_lb and all the group ids in the cache to the server;
      wait for the group validity report from the server;
      invalidate the cache according to the group validity report;
   }
∀q ∈ QL
   {
      if all the objects referenced by q are in the cache
         process q;
      else
         send the missed object ids to the server;
   }
T_lb = T
```

Figure 4: Algorithm for query processing using simple-grouping and GCORE schemes.

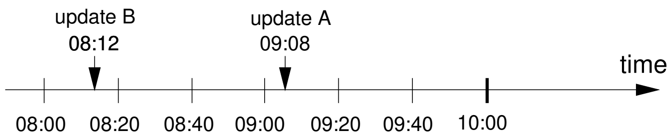
updated object ids in a group have already been included in IR, these objects will be invalidated by the mobile computer when it receives IR. With the hot update set excluded from a group, the server can conclude that the objects that are not updated in the group can be retained in the cache and validate the rest of the group as a result. This scheme is therefore referred to as grouping with cold update-set retention in this paper. GCORE is energy efficient because it incurs both low uplink costs for validity checking and low downlink costs as it retains more cached objects.

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 sophisticated data structure for the group update history. In this paper, we present an efficient implementation for GCORE. It maintains for each group the object update history of the past W broadcast intervals ($W \geq w$), 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. 5 for its definition). In addition, it also maintains the number of distinct objects that were most recently updated between $(T - W \times L)$ and $(T - w \times L)$ to speed up the group validity checking.

As an example, Fig. 6 shows a snapshot of a group table entry, `group_table[1]`. In Fig. 6, group 1 contains objects A, B, C, D, E, and F. A broadcast interval is 20 minutes, $w = 3$ and $W = 6$ in this example. Object A was updated at 09:08 and object B was updated at 08:12. The server

```
struct group_table_entry
{
    double time;  /* most recent update timestamp of a group          */
    int total_wW; /* total number of distinct objects most recently   */
                  /* updated between (T - W * L) and (T - w * L)      */
    struct pair *uplist; /* pointer to a list of pairs (o, t)'s for   */
                         /* objects updated in the previous W         */
                         /* broadcast intervals                       */
} group_table[];
```

Figure 5: Data structure for maintaining group update history.



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

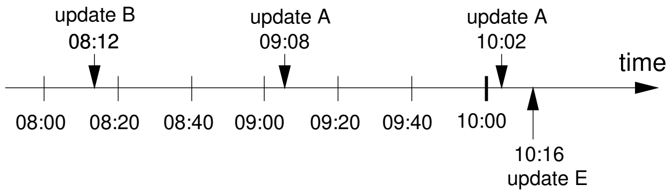
Current time = 10:00

T = 10:00

Figure 6: Example of a `group_table[]`.

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 $(T - W \times L)$ and $(T - w \times L)$ is 1. Namely, `group_table[1].tot_wW` at the moment is 1 (the number of distinct objects most recently updated between 08:00 and 09:00 is 1).

For every object updated, the server updates `group_table[]`. Assume that object a is updated at time t and it belongs to group g. The server first sets `group_table[g].time` to t. Next, it checks to see if this object has already been in `group_table[g].uplist`. If yes, its corresponding update time is changed to t. Otherwise, a new pair is created for this object. As a continuing



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

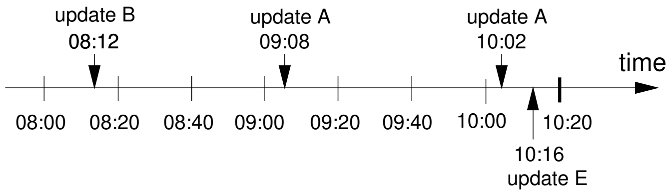
T = 10:00

Figure 7: Changes to `group_table[]` due to an object update.

example, Fig. 7 shows the changes to `group_table[1]` at time 10:19 after object A and E were updated at 10:02 and 10:16, respectively. 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 reflect the most recent update to the group.

Every time when the server broadcasts an invalidation report, it also updates `group_table[]` accordingly. For each group, the server removes the pairs in `group_table[].uplist` that have update times less than $T - W \times L$. This would eliminate any objects that were updated before $T - W \times L$, and limit the amount of history that the server must maintain under GCORE. In addition, the server also updates `group_table[].total_wW`, which keeps the number of distinct objects that were most recently updated between $T - W \times L$ and $T - w \times L$. The maintenance of `group_table[].total_wW` is for a fast group validity checking. If no object was most recently updated during $T - W \times L$ and $T - w \times L$ and $T_{lb} > (T - W \times L)$, it means if there are any objects in the group updated since T_{lb}, their ids have been included in IR. Notice that in GCORE (see Fig. 4), the cache validity checking is done after the mobile computer first invalidates its cache based on IR, the most recent invalidation 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. 6 and 7, we show the changes to `group_table[1]` at time 10:20



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 8: Changes to `group_table[]` due to invalidation a new invalidation broadcast.

when a new IR is broadcasted in Fig. 8. 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 updated between 08:20 and 09:20 is zero.

The server checks the validity of a group by examining whether or not all the objects updated since the mobile computer becomes disconnected have been included in the latest invalidation report IR. If yes, then the group can be retained by the mobile computer. Otherwise, the entire group is invalid. This can be achieved by first 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 $T_{lb} > (T - W \times L)$. If yes, this group is also valid since the updated object ids are all included in IR. Otherwise (either `group_table[].tot_wW` is greater than zero or $T_{lb} < (T - W \times L)$), this group is invalid. Based on the example shown in Fig. 8, if a mobile computer was disconnected at time 09:02 and woke up at 10:18, the server can valid group 1 if the mobile computer checks for its validity after 10:20. In this case, the T_{lb} sent to the server by the mobile computer would be 09:00. All the object ids that are updated between 09:20 and 10:20 would be included in the IR. As a result, the rest of group 1 can still be retained in the cache since there is no object that was most recently updated between 09:00 and 09:20. Note that object A has been further updated at 10:02 and has been included in the IR.

3 Simulation model

In order to evaluate the performance of GCORE, an event-driven 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 effectiveness of caching is measured by the communication bandwidth requirement for query processing. This communication requirement includes the uplink communication for validity checking and asking for missed objects and the downlink communication for sending the data to the mobile computer. High bandwidth requirement means less effective caching and more energy consumption.

There are D total objects in the database. Of the D objects, β portion of them are hot update set, while $(1-\beta)$ portion of them are cold update set. Data in the hot update set are randomly chosen from the D objects. The data objects in the hot update set are updated by α fraction of the transactions and the objects in the cold update set are updated by $(1-\alpha)$ fraction of the transactions. Update transaction arrival is a Poisson process with rate λ_u. The objects updated by a transaction are randomly chosen from either the hot update set or the cold update set. The number of objects updated is uniformly distributed between $U/2$ and $3U/2$ objects, where U is the average.

In order to focus on the cache invalidation effect, we assumed that a cache miss is resulted only from invalidation. 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 computer reference to a fixed subset of objects that are initially 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 invalidated 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 distributed between $Q/2$ and $3Q/2$, where Q is the average.

The probability of a mobile computer becoming disconnected in the next broadcast interval given that it is active now is denoted as P_{disc}. The length of disconnection is exponentially distributed with mean L_{disc}. When a mobile computer is active, query interarrival times are exponentially distributed with mean $1/\lambda_q$ seconds. Queries are batched in QL and are not processed until a mobile computer receives a broadcast invalidation report. We accumulated the communication costs for query processing in a mobile computer for a period of time (40,000 broadcast intervals),

Notation	Definition (Default values)
D	database size in the server (100,000 objects)
B	cache size in a mobile computer (5000 objects)
λ_u	update transaction arrival rate in the server, Poisson interarrival time distribution (0.01 jobs/sec)
λ_q	query arrival rate in a mobile computer, Poisson interarrival time distribution (0.1 jobs/sec)
U	mean number of objects updated by a transaction (5 objects)
Q	mean number of objects referenced by a query (10 objects)
$\alpha - \beta$	reference skew by the update transactions (90%-5%)
G	group size (100 objects)
w	number of broadcast intervals for an invalidation report (5)
W	number of broadcast intervals for maintaining the update history by GCORE (15)
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 for the current broadcast invalidation report
T_{lb}	the timestamp of the latest broadcast invalidation report received by a mobile computer 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 (0.1)
L_{disc}	mean length of a disconnection (100 seconds)

Table 1: System and workload parameters.

and then compute its average bandwidth requirement. Since broadcast costs are always required whether or not a mobile computer is on or off, we don't include the broadcast costs in our results. 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 definition for all the simulation parameters are summarized in Table 1. The default values used in the simulations, if not otherwise specified, are included in the parentheses.

For the no-checking scheme, the communication costs for query processing are dominated by the downlink costs resulted from the server sending the data to the mobile computer due to cache misses. Besides the uplink costs to request missed objects in the local cache, there are no additional uplink costs due to cache validity checking. Note that for all the caching schemes there are always uplink costs due to a mobile computer requesting missed objects in its cache. For the other checking schemes, the uplink costs for cache validity checking are added to the total bandwidth costs. For the simple-checking scheme, the uplink costs for validity checking is due to the mobile computer sending the object ids and T_{lb} to the server. For the simple grouping and GCORE schemes, the uplink costs for validity checking is due to the mobile computer sending the group ids and T_{lb}. Depending on the group size, the uplink costs of the simple checking scheme can be substantially larger than those of the simple grouping and GCORE schemes.

4 Simulation results

This section evaluates and compares the effectiveness of various caching schemes described in Section 2. We compute the communication costs due to query processing in the mobile computer, including the uplink costs of checking the cache validity and requesting missed objects, and the downlink costs of the server sending the validity report and the data objects due to cache misses. The bandwidth requirement presented is the average of the communication costs over 40,000 broadcast intervals. Since the bandwidth requirement would be smaller if the mobile computer is often disconnected (similar to the case of a sleeper in [4]), we also plotted the bandwidth requirement per 1000 queries. By doing this, the true effectiveness of caching is manifested in bandwidth requirement. In addition, since the energy consumptions for sending requests uplink and receiving data from downlink may be different, we also show the relative energy consumption between the simple grouping scheme and GCORE. Unless otherwise specified, the default values for most of the simulations are provided in Table 1.

First, we examine the impact of L_{disc} on the bandwidth requirement. Fig. 9 shows the total bandwidth requirement for the four different mobile caching schemes. The left-hand side shows the average bandwidth requirement over 40,000 broadcast intervals. The right-hand side shows the average requirement per 1000 queries. As indicated from this figure, it is beneficial to do cache validity checking if $L_{disc} < 200$ seconds. Also, GCORE and simple grouping are much more effective than the simple checking scheme. As L_{disc} increases, i.e., a mobile computer is off for a long period of time, the number of cached objects that are still valid decreases. Thus, it may not pay off for the simple checking scheme to incur the large uplink costs for validity checking. However, by using simple grouping or GCORE, it is still beneficial to do checking since the uplink costs is relatively small. Finally, as the disconnection interval increases, the total number of queries issued during the simulation interval decreases as well. Therefore, the bandwidth requirement during the simulation interval for the no-checking case decreases. However, it is not true if we consider the bandwidth

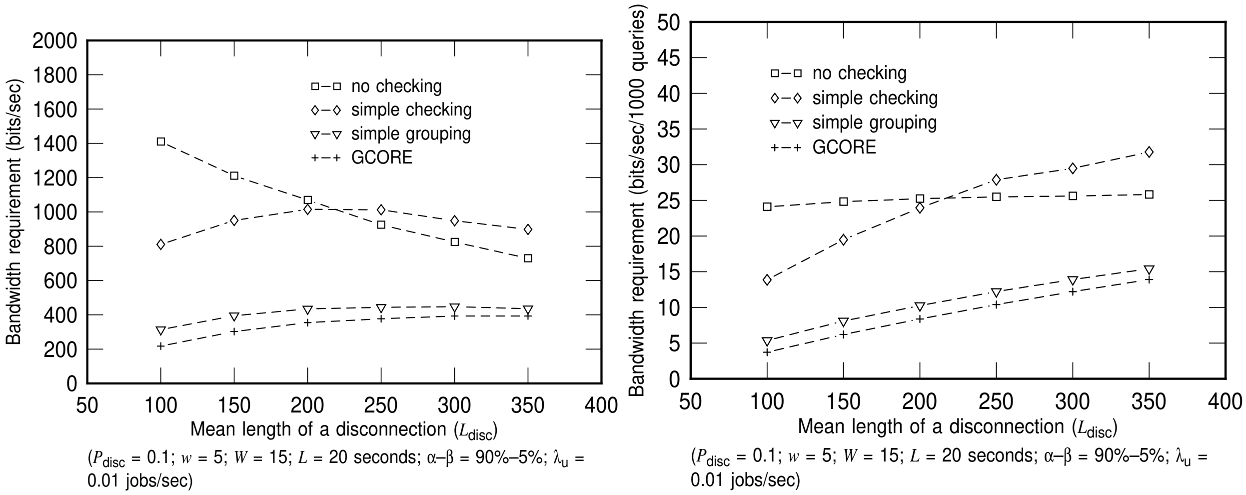


Figure 9: Impact of disconnection length on the bandwidth requirement for query processing.

requirement per 1000 queries.

Note that the total bandwidth requirement would have been much higher if a mobile computer sends both the object or group ids and their timestamps for validity checking, especially for the simple checking scheme where the uplink costs dominate the total costs. Here, as explained in Section 2.2, only a single T_{lb} is sent with the object ids or group ids. It is important to pay special attentions to communication overhead in wireless mobile computing as communication (uplink or downlink) consumes energy as well as wireless bandwidth.

Next, we look at the performance impact of P_{disc}, the conditional probability of a mobile computer disconnected in the next broadcast interval given that it is active now. The larger P_{disc} is, the shorter each connection interval is for a mobile computer. Fig. 10 shows both the average bandwidth requirement during the simulation interval (40,000 broadcast intervals) and the average bandwidth requirement per 1000 queries. For the case of no checking, the costs of query processing is dominated by the downlink costs due to cache misses, as there is no uplink cost for cache validity checking.

The bandwidth requirement of the simple checking scheme increases rapidly as the probability of disconnection increases. This is due to the fact that every disconnection results in the mobile computer sending a lot of the cached object ids to the server for validity checking. Thus, it requires large uplink bandwidth. On the other hand, Fig. 10 also clearly demonstrates that both simple grouping and GCORE requires significantly less bandwidth than the simple checking scheme. Additionally, GCORE performs better than the simple grouping scheme. Moreover, as P_{disc} becomes very large, checking cache validity may not be beneficial even if it is done in a group level. Under such an occasional connected condition, a mobile computer may be better off if it does not cache

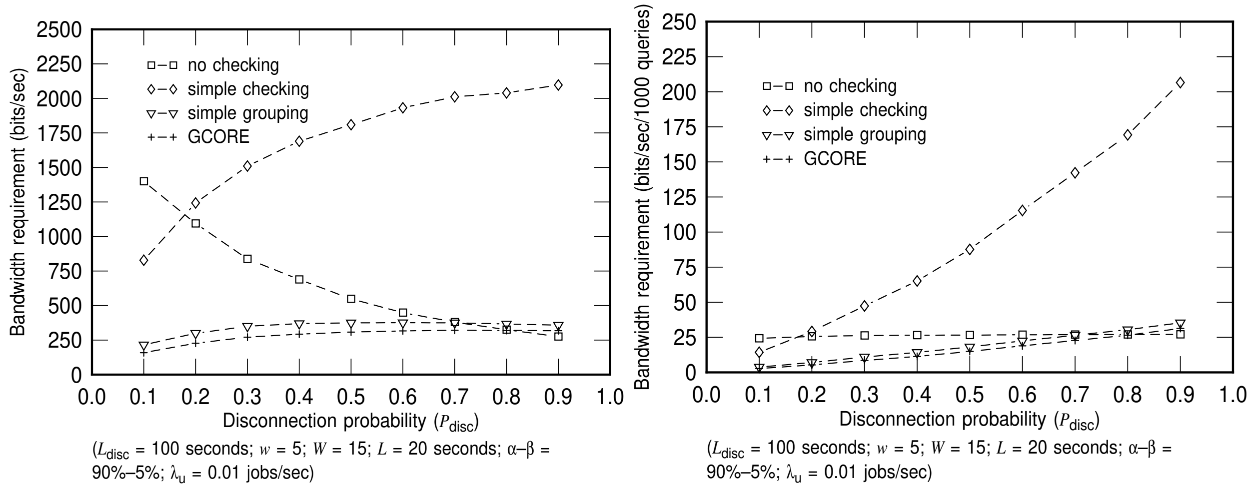
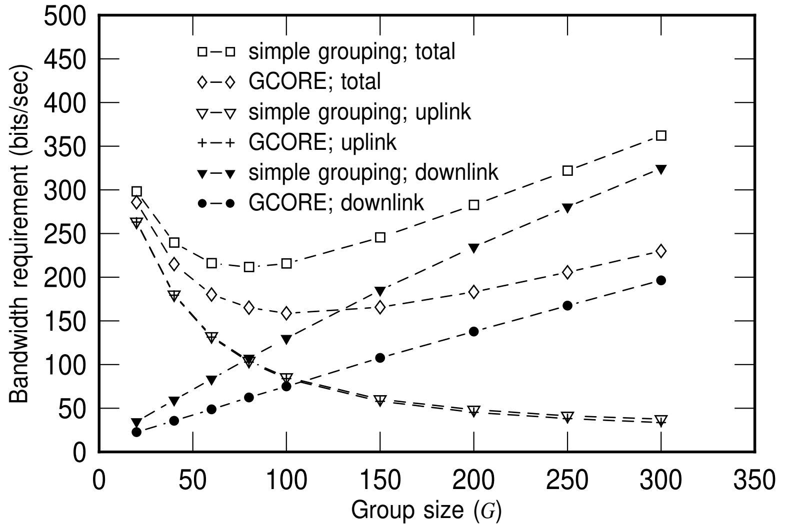


Figure 10: Impact of disconnection probability on the bandwidth requirement for query processing.

objects at all since all the objects are likely to have been changed when occasionally it wakes up. To compare and contrast the simple grouping scheme and GCORE in a more detailed level, we plotted the performance of both schemes with different group sizes in Fig. 11. In addition to the total bandwidth, we also show the uplink and downlink bandwidths in Fig. 11. In general, GCORE requires much less total bandwidth than the simple grouping scheme. For example, in Fig. 11, the optimal bandwidth requirements are 158.8 bits/sec and 211.5 bits/sec for GCORE and simple grouping, respectively. The optimal group size for each scheme is also different, where GCORE has a larger optimal group size. In Fig. 11, the optimal group sizes are 100 objects and 80 objects for GCORE and simple grouping, respectively. Since the uplink costs for both schemes are essentially the same, the performance advantage of GCORE is due to its effectiveness in retaining the cold update set of data objects in a group. This is demonstrated in the much less downlink bandwidth requirement of GCORE, as compared with the simple grouping scheme. As the group size increases, the uplink costs of both schemes decrease simply because less group ids need to be sent. But, with a larger group size, it is more likely to invalidate a group and thus increasing the downlink costs due to cache misses.

Sending requests uplink by a mobile computer consumes not only wireless bandwidth but also battery energy. So does receiving data from the server. However, the energy requirement for a mobile computer for uplink communication and receiving data from downlink may be quite different. Uplink communication may consume significantly more energy than downlink communication. Here we compare simple grouping and GCORE in relative energy consumption for query processing during 40,000 broadcast intervals. For sensitivity analysis, we examined cases of energy requirement ratio



(*disc* = 0.2; *w* = 5; *W* = 15; *L* = 20 seconds; α–β = 90%–5%; λ_u = 0.01 jobs/sec)

Figure 11: Detailed comparisons of simple grouping and GCORE.

for uplink over downlink being 1, 5 and 10. Fig. 12 shows the relative energy consumption for both schemes. The energy requirement of GCORE with group size of 100 objects for uplink/downlink energy ratio of 1 was assumed to be 1. For the three different ratios, GCORE is more energy efficient than simple grouping. As the ratio increases, the optimal group size becomes larger. This is because larger group size means there are less number of group ids to be sent uplink. If uplink costs are much higher than downlink costs, then larger group size can minimize the energy consumption. Note that comparing Fig. 12 with 11, if the uplink/downlink energy ratio is greater than 1, then the optimal group size for total bandwidth requirement and that for energy consumption is different. For example it is 100 object for GCORE in Fig. 11 but it is 300 objects in Fig. 12 if the uplink/downlink energy ratio is 10.

5 Summary

In this paper, we studied the issues of caching frequently accessed data in a wireless mobile computer. Because of the limited battery power, a mobile computer may be forced to power off for saving energy. Once disconnected, it may miss some of the cache invalidation reports broadcasted by the server. As a result, when it wakes up again, the entire cache may have to be discarded, significantly reducing the benefits of caching. We presented an energy-efficient caching scheme, called grouping with cold update-set retention GCORE, that allows a mobile computer to disconnect to save energy, but still retains most of the caching benefits by salvaging most of the cached objects

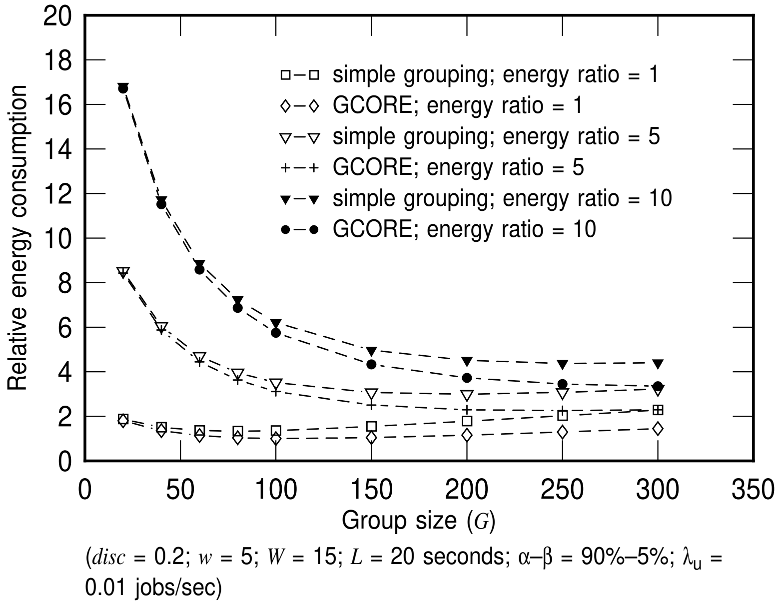


Figure 12: Comparisons of energy consumption between simple grouping and GCORE.

if they are still valid. An efficient implementation of GCORE was presented in the paper. It uses a simple data structure to facilitate the dynamic exclusion of hot update set from a group so that the cold update set can be retained. Upon waking up, a mobile computer checks its cache validity with the server in a group level to save uplink costs. The server determines that a group is valid if the hot update set has already been included in the most recently broadcasted invalidation report. Simulations were conducted to evaluate the performance of GCORE. We compared GCORE with a no-checking, a simple-checking and a simple grouping schemes. The results show that, compared with no-checking and simple checking, both simple grouping and GCORE requires much less bandwidth for processing queries. Lower bandwidth requirement also consumes less energy and thus more energy efficient. Furthermore, GCORE is more energy efficient than the simple grouping scheme because it can retain more objects in a mobile cache after a disconnection.

References

[1] A. Acharya and B. R. Badrinath. Checkpointing distributed applications on mobile computers. In *Proc. of Int. Conf. on Parallel and Distributed Information Systems*, pages 73–80, 1994.

[2] R. Alonso and H. Korth. Database system issues in nomadic computing. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 388–392, 1993.

[3] B. R. Badrinath and T. Imielinski. Replication and mobility. In *Proc. of the 2nd Workshop on the Management of Replicated Data*, 1992.

[4] D. Barbara and T. Imielinski. Sleepers and workholics: Caching in mobile distributed environments. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, 1994.

[5] Y. Huang, P. Sistla, and O. Wolfson. Data replication for mobile computers. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 13–24, 1994.

[6] Y. Huang and O. Wolfson. Ojbect allocation in distributed databases and mobile computers. In *Proc. of Int. Conf. on Data Engineering*, pages 20–29, 1994.

[7] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *Proc. of Very Large Data Bases*, pages 41–52, 1992.

[8] T. Imielinski and B. R. Badrinath. Mobile wireless computing. *Communications of the ACM*, 37(10):18–28, Oct. 1994.

[9] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 25–36, 1994.

[10] R. H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, pages 6–17, First Quarter 1994.

[11] J. Kistler and M. Satyanarayanan. Disconnected operation in the CODA file system. *ACM Trans. on Computer Systems*, 10(1):3–25, Feb. 1992.

[12] P. Krishna, N. H. Vaidya, and D. K. Pradhan. Location management in distributed mobile environments. In *Proc. of Int. Conf. on Parallel and Distributed Information Systems*, pages 81–88, 1994.