Energy Efficient Cache Invalidation in a Mobile Environment

Narottam Chand, Ramesh Chandra Joshi, Manoj Misra
Electronics & Computer Engineering Department
Indian Institute of Technology, Roorkee - 247 667. INDIA
Email: {narotdec, joshifcc, manojfec}@iitr.ernet.in

***Abstract**: Caching in mobile computing environment has emerged as a potential technique to improve data access performance and availability by reducing the interaction between the client and server. A cache invalidation strategy ensures that cached item at a mobile client has same value as on the origin server. To maintain the cache consistency, the server periodically broadcasts an invalidation report (IR) so that each client can invalidate obsolete data items from its cache. The IR strategy suffers from long query latency, larger tuning time and poor utilization of wireless bandwidth. Using updated invalidation report (UIR), the long query latency can be reduced. This paper presents a caching strategy that preserves the advantages of existing IR and UIR based strategies and improves on their disadvantages. Simulation results prove that our strategy yields better performance than IR and UIR based strategies.*

Categories and Subject Descriptors
C.2.1 [**Network Architecture and Design]**: Wireless Communication

General Terms
Network Architecture, Network Design, Mobile Computing

Keywords: : Mobile computing, cache invalidation, wireless, data broadcast, invalidation report, disconnection, failure.

Received 30 Oct.2004; Reviewed and accepted 30 Jan. 2005

1 Introduction

The advances in portable computing devices and wireless technology have dramatically changed the way of information access. Wireless communication permits users to access global data from any location by carrying mobile devices. However, the mobile computing paradigm presents a dramatic discrepancy relative to currently matured wired network computing. Mobile clients in wireless environments suffer from scarce bandwidth, low-quality communication, frequent network disconnections (either volunteer or involunteer), and limited local resources (computing power, battery, storage, display, etc). Caching of frequently accessed data items on the client side is an effective technique to reduce network traffic and query latency. Bandwidth and battery power are saved, as no transmission is required for clients to access data from their caches. Furthermore, the availability of data is improved because even when a mobile client is disconnected from the network, data stored in its local cache is still accessible, making disconnection operations a possibility.

Data consistency must be ensured between client and server to prevent clients from answering query by out-of-date cached data items updated by server. In order to assist mobile clients in maintaining the consistency of their caches, a number of cache invalidation techniques have been proposed [1-14]. In these approaches, the server periodically broadcasts invalidation reports (IRs) to inform the clients about which data items have been updated during the most recent past. When a mobile client receives an IR, it can use the report to identify out-dated data items in its cache and discard them before using the cache to answer queries. However, the IR based scheme suffers from long query latency and they make poor utilization of available wireless bandwidth. Cao [1, 4, 5, 15] has proposed several updated invalidation report (UIR) based caching strategies to address the problem. Each UIR contains information about most recently updated data items since the last IR. In case of cache hit, there is no need to wait for the next IR and hence the query latency is reduced. However, if there is a cache miss, the client still needs to wait for the data to be delivered. Thus, due to cache miss, the UIR strategy has same query latency as IR strategy.

To overcome the limitations of existing cache invalidation strategies, this paper presents a synchronous stateful caching strategy where cache consistency is maintained by periodically broadcasting *update reports (URs)* and *request reports (RRs).* The center design of our strategy includes reducing the query latency, improving the cache hit ratio, minimizing the client disconnection overheads, better utilization of wireless channel, and conserving the client energy. The track of cached items for each client is maintained at the home mobile support station in the form of *cache state information (CSI*). Use of CSI reduces the size of IR by filtering out non-cached items and handles long disconnection [25]. In different IR based strategies [2, 3, 6-11, 13, 14, 16], even though many clients cache the same updated data item, all of them have to query the server and get the data separately from the server. It misuses a large amount of wireless bandwidth and client battery energy. Our strategy periodically broadcasts update report (UR) [24], to minimize uplink requests and downlink broadcasts. To further reduce query latency, the strategy uses *request reports (RRs)*, where all the recently requested items are broadcast after the UIR. Selective tuning is used to conserve the client energy. Another drawback common to IR based approaches is that if the disconnection time of a client is larger than a fixed period of time, the client should discard its entire cache even if some of the cached items may still be valid. This paper demonstrates a more efficient handling of arbitrarily long client disconnection as compared to IR and UIR approaches.

The rest of the paper is organized as follows. Section 2 gives a brief review on some representative approaches on cache invalidation. In section 3, we explain our proposed caching strategy. Simulation results are presented and discussed in Section 4. Concluding remarks are given in section 5.

2 Related Work

A number of broadcast based cache invalidation strategies have been proposed for mobile environments. Existing strategies can be broadly classified into two types: stateful and stateless servers, distinguished by whether clients' cache information is maintained at the server or not. Barbara and Imielinski [14] have proposed the basic stateless approach called timestamp (TS), where the server broadcasts an IR every L seconds, indicating which data items are updated in the last w*L seconds, where w is the broadcast window size. Advantages of IR based approaches include high scalability and energy efficiency because the size of each IR is independent of the number of clients, and IRs are scheduled to be broadcast periodically (synchronously). As such, clients can switch to doze mode operation between successive IRs to save battery power. The major drawback, however, is: clients must flush their entire caches after long disconnection (> w*L), even if some of the cached items may still be valid. Another drawback is long query latency, as clients must at least wait for the next IR before answering a query to ensure consistency.

Various approaches have been proposed to address the long disconnection problem [3, 10]. Jing et al. [10] proposed a Bit-Sequence (BS) scheme that uses a hierarchical structure of binary bit sequences to represent IRs. The scheme is good for clients with long disconnections but has very large size of IR and assumes that update rate to the database is not high. Hu and Lee [3] have proposed a family of invalidation algorithms. The advantage of these algorithms is that the type of invalidation report to be sent is determined

dynamically based on system status. Tan [7] reexamined the BS method and studied different organizations of the invalidation report. These organizations facilitate clients to selectively tune to the portion of the reports. Hou et al. [12] proposed a scheme to reduce the false invalidation rates based on BS reports. Wu et al. [6] proposed a scheme which modifies the IR algorithm to include cache validity checks after reconnection. The method has same basic problem as in IR method (e.g. when the disconnection time is greater than W, nothing can be salvaged).

Cao [4, 15] has done some pioneering work to reduce the long query delay associated with basic IR based strategy by introducing the idea of updated invalidation report (UIR). In this approach, a small fraction of the essential information (called UIR) related to cache invalidation is replicated several times within an IR interval, and hence the client can answer a query without waiting until next IR. However, if there is a cache miss, the client still needs to wait for the data to be delivered. In [1, 15] author introduced a counter based method where clients intelligently prefetch the data that are most likely used in the future. Kahol et al. [2, 9] present an asynchronous stateful (AS) invalidation scheme where each mobile client maintains its own Home Location Cache (HLC) to deal with the disconnections. The strategy suffers from large number of uplink requests and downlink broadcasts for the same updated item.

Yuen et al. [8] proposed a scheme based on absolute validity interval (AVI) for each data item. A mobile client can verify the validity of a cached item by comparing the last update time and its AVI. To solve the problem of large size invalidations reports and duplicate uplink requests, Lai et al. [18, 19], proposed two techniques: Validation-Invalidation Reports (VIR) and Delayed Request Scheme (DRS). In VIR approach when size of IR becomes larger, the validation report can be used instead. Use of DRS reduces the number of uplink requests. Chuang et al. [20] address some of the cache invalidation issues like disconnection handling, energy consumption in a synchronous stateless environment. Wang et al. [21] proposed a hybrid invalidation strategy called Scalable Asynchronous Cache Consistency Scheme (SACCS). Unlike stateful algorithms, SACCS maintains only one flag bit for each data item in MSS and unlike the existing synchronous stateless approaches, it does not require periodic broadcast of IRs. Because of asynchronous nature of SACCS approach, it does not provide any guarantee on waiting time of clients and hence they can rarely switch to power save mode. Authors in [22] describe the cache consistency maintenance for intra- and inter-roaming clients. Three strategies: homogenous IR, inhomogeneous IR without roaming check and inhomogeneous IR with roaming check are applied to TS and SACCS strategies.

3 Proposed Caching Strategy

In this section, we present our synchronous stateful caching strategy.

3.1 Mobile Computing Environment

Figure 1 illustrates the mobile computing environment assumed in this study. The model consists of two distinct sets of entities: *Mobile Hosts (MHs)* and *Fixed Hosts (FHs)*. Some of the fixed hosts called *Mobile Support Stations (MSSs)*, are augmented with a wireless interface in order to communicate with the mobile hosts, which are located within a radio coverage called a cell. We use the terms client and mobile host interchangeably. Wireless cells cover the entire surface of installation (e.g., a university campus, industry). MSSs are also known as *Base Stations (BSs)* and are interconnected by means of fixed LAN infrastructure. An MSS acts like a gateway between wired LAN and wireless network. Individual LANs are interconnected by means of routers; one of them is acting as an Internet gateway. An MH communicates with the server via an MSS over a wireless communication link. A fixed network has a large bandwidth (order of Mbps or Gbps); while the bandwidth of the wireless channel is scarce (19.2 Kbps – 10 Mbps). An MH can move within a cell or between cells while retaining its network connection. When an MH moves from one cell to another (called handoff), its wireless connection is switched to the new cell. An MH either connects to an MSS through a wireless link or disconnects from the MSS by operating in a power save mode [2].

3.2 Cache Invalidation Strategy

Server database D is a collection of N data items with ids: d_1, d_2, ..., d_N. D_i denotes the actual data of an item with id d_i. Each data item d_i is of same size S_{data} (in bits) and has two timestamps: t_i is the most recent timestamp when d_i got updated at the server and t_i^r, called *latest request time*, represents the most recent time when d_i was last requested by any client. Clients only issue simple requests to read the most recent copy of data items. In order to serve a request sent from a client, the MSS needs to communicate with the database server to retrieve the database items. Caching techniques may also be applied at MSS. Since the communication between the MSS and the database server is transparent to the clients, from the client point of view, the MSS is the same as the database server.

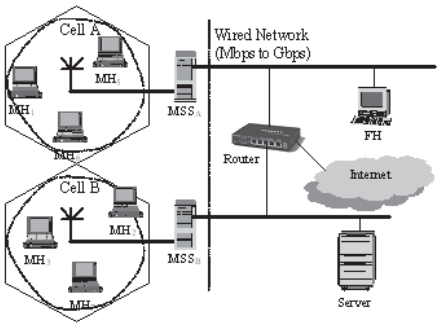


Figure 1 Mobile computing environment.

Frequently accessed data items are cached on the client side. Each client has same cache capacity of C items. When caching is used, data consistency issues must be addressed. We assume the *latest value* consistency model [9]. To ensure cache consistency, the server broadcasts UR every L seconds and it also broadcasts (m-1) RRs between two URs. The structure of UR_i at time T_i is as follows:

IR_i	UR_INDEX_i	UR_DATA_i

$IR_i = \{(d_x, \tau_x) | (D_x \in D) \wedge (n_x > 0) \wedge (T_i - w*L < \tau_x \leq T_i)\}$

$UR\_INDEX_i = \{d_x | ((T_{i-1} < t_x \leq T_i) \wedge (n_x > 0)) \vee (T_{i-1,m-1} < t_x^r \leq T_i)\}$

$UR\_DATA_i = \{D_x | d_x \in UR\_INDEX_i\}$

IR contains the update history of past w broadcast intervals whereas UR_DATA contains actual data value for the items which have been updated during previous UR interval and the items that have been requested during the latest RR interval. At interval time $T_{i,k}$, $RR_{i,k}$ is constructed as follows:

$UIR_{i,k}$	$RR\_INDEX_{i,k}$	$RR\_DATA_{i,k}$

$UIR_{i,k} = \{d_x | (d_x \in D) \wedge (T_{i,0} < t_x \leq T_{i,k}) \wedge (n_x > 0)\}$ $(0 < k < m)$
$RR\_INDEX_{i,k} = \{d_x | (T_{i,k-1} < \leq T_{i,k})\}$
$RR\_DATA_{i,k} = \{D_x | d_x \in RR\_INDEX_{i,k}\}$

To enable selective tuning, the MSS broadcasts the index information before the broadcast of actual data. An index (UR_INDEX/RR_INDEX) is list of the items broadcast as part of UR_DATA/RR_DATA. It is notable that use of index in our strategy does not increase size of the report. For instance, for an item d_x, the number of bits in both the approaches (with and without index) remains $S_{id}+S_{data}$ (where $S_{id}=|d_x|$, is size of the item id and $S_{data}=|D_x|$, is number of bits in actual data of the item). Within index, the items are arranged in nondecreasing order of their cache counts to further reduce the query latency [23]. To answer a query, the client listens to the IR/UIR part of report (UR/RR) and uses it to decide cache validity. If there is a valid cached copy of the requested item, the client returns the item immediately;

otherwise, it sends a query request to the server.

For each client MH_x, the home MSS maintains cache state information CSI_x (i.e. list of cached items) and request timestamp (i.e. last time the client communicated with the MSS). The MSS also maintains cache count n_i for each item d_i. For a client request, the MSS updates the relevant counters, corresponding CSI and timestamp, and forwards the request to the server. To save energy a client may power off and only turns on during the report broadcast. Our strategy reduces the size of IR by filtering out non-cached items, thus enhancing the overall performance [25]. Reduced size of IR/UIR reports improves the downlink channel utilization. Further, it enhances the uplink channel utilization by adopting *delayed uplink (DU)* technique [25]. To illustrate the working of proposed strategy consider an example as shown in Figure 2 for the mobile scenario of Figure 1.

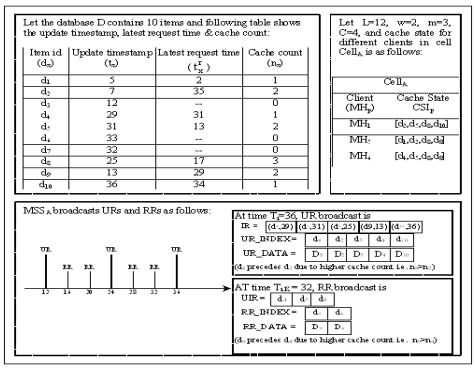


Figure 2 Working of our strategy

In UIR scheme, the requested data are broadcast after the next IR, thus, due to cache miss the expected query latency is L/2 seconds. To reduce the query latency due to cache miss, our strategy broadcasts the recently requested data items after the next report (IR/UIR) as part of UR/RR whichever arrives earlier, such that the expected query latency is L/(2*m) seconds instead of L/2 seconds. For example, when a client receives a cache miss request between $T_{i,1}$ and $T_{i,2}$, it cannot answer the query until T_{i+1} in the UIR approach, but it can answer the query at $T_{i,2}$ in our approach (see Figure 3).

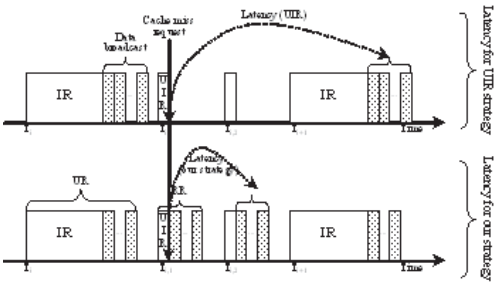


Figure 3 Reducing the query latency

3.2.1 Handling client disconnection

Since a UR broadcasts information about the items that have been updated during past w*L time, our strategy handles the client disconnection less than w*L without any additional overhead. When a client reconnects after a disconnection time longer than w*L, it sends an uplink request with the last received UR timestamp T_l (before disconnection) to the MSS. On receipt, the MSS constructs a binary vector *DIV* called *disconnection information vector*. DIV is of size C bits and contains the validity information about the cached items by the client.

In comparison to our previous disconnection handling strategy [25], here we introduce the idea of *disconnection report (DR)*, that further enhances the expected value of disconnection overhead. The rationale behind the use of DR is that, there may be two extreme cases of all the cached items being valid (DIV=0) or invalid (DIV=1) after the long disconnection. When all the bits in DIV are same (i.e. 0 or 1) then it is better to send one bit to the client instead of C bits. The MSS computes DR (of size 1 or C bits) from DIV and sends it to the client. After downloading DR, the client invalidates the cache. The whole process is reflected in two algorithms: Algorithm 1 runs on the MSS and Algorithm 2 runs on the client side.

Algorithm 1. The algorithm at the MSS

The MSS receives reconnect(T_l) from client MH_j;
$DIV_j = 0$; Set DR_j to empty;
T^r_j = current_time;
for each item $d_x \in CSI_j$ do
 if ($t_x > T_l$) /* item has become invalid */
$DIV_j[k] = 1$ where $CSI_j[k] = d_x$; /*d_x is the kth cached item */
if all bits in DIV_j are same then $DR_j = DIV_j[1]$; /* DR_j has one bit 0/1 */
 else append DIV_j to DR_j;
send DRj to MHj;

Algorithm 2. The algorithm at the client

The MH_j reconnect after the disconnection;
if (disconnection time $\leq$ w*L)
 receive next UR to invalidate the cache;
else /* long disconnection */
 {
  send reconnect(T_l) to the MSS;
  wait until receiving DR_j;
  if DR_j has single bit, use it to invalidate the cache;
  else
   {
    for each kth bit b_k (starting from MSB) in DR_j
     if $b_k = 1$ then d_k = invalid
      else d_k = valid;
   }
 }

As compared to UIR strategy [1, 4, 5, 15], which handles disconnection by sending the ids for updated items, our strategy uses only one bit for an item, thus reducing the reconnection overheads tremendously. For our strategy the maximum reconnection overhead is C bits which is very low as compared to UIR. Because of smaller size of overhead, our strategy is also very much effective in terms of bandwidth utilization, client tuning time and energy consumption.

Consider a caching scenario of Figure 2. Let client MH_5 disconnects at time 14 and reconnects at time 47 as shown in Figure 4.

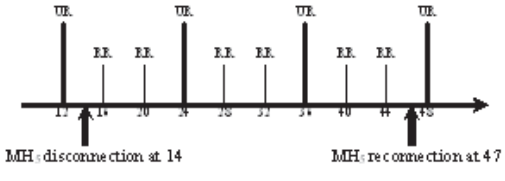


Figure 4 Disconnection example

Disconnection period 47-14 = 33 (i.e. > w*L). MH_5 has no information about updates between (12, 47], and it misses URs at 24 and 36. On wake up it sends T_l = 12 to the MSS_A. The MSS_A constructs DR as shown below:

d_1	d_2	d_8	d_9

0	0	1	0

When MH_5 receives DR_5 = 0010, it is interpreted as: items d_1, d_2 & d_9 are still valid, whereas item d_8 has become invalid. The reconnection overhead for our strategy is 4 bits. For UIR, the overhead = min {invalid items, valid items}*item id size (S_{id}). Generally S_{id} = 32 bits, therefore the overhead value = 32 bits.

3.2.2 Handling client failure

From a client perspective a failure is treated as a long disconnection. When a client recovers from a failure longer than w*L seconds, it sends an uplink request with the last received UR timestamp T_l to the MSS. Due to client failure, the CSI may not represent the actual state of client cache. Moreover, all cache counters may grow very large after some amount of time and fail to represent the number of caching clients. For a client MH_x, the last request timestamp T_x^r maintained at the home MSS is used to handle such situation. If the value (current_time-Txr) is larger than some *fail_threshold*, the MSS tries to communicate with MH_x. If the client does not respond in a certain time period, the MSS assumes that it has failed. Hence, the MSS decreases the relevant counters and removes the associated CSI_x. On receiving T_l from the client MH_x, the MSS updates the T_x^r, creates a new CSI_x list and responds to the client with DR_x=1 indicating that whole cache is invalid.

4 Performance Evaluation

4.1 Methodology

To evaluate the performance of the proposed scheme, we have developed a simulation model. Table 1 shows system parameters and their values. The model consists of a single server serving multiple clients. The database can only be updated by the server, while the queries are generated by the clients following an exponential distribution. The database items are distributed into two categories: the *hot* data subset and the *cold* data subset. The hot data subset includes data items from 1 to 50 and the cold data subset includes the remaining data items of the database. Clients have a large probability (80%) to access the data in the hot subset and a low probability (20%) to access the data in the cold subset. The server broadcasts IRs (URs in our algorithm) every L seconds and UIRs (RRs in our algorithm) every L/m seconds. The server generates a single stream of updates separated by an exponentially distributed update arrival time. All updates are randomly distributed inside the hot data subset and the cold data subset, while 40% of the updates are applied to the hot data subset. The mean inter-arrival time of queries generated by all clients is T_q. The inter-arrival time of updates at the server is distributed exponentially with a mean of T_u.

A client follows exponential distributed disconnection with mean time T_d and has probability p_d to enter the disconnection mode only when the outstanding query has been served.. The client processes generate queries one by one. If the referenced data items are not cached on the client side, the data ids are sent to the server for fetching. Once the requested data items appear on the channel, the client brings them into its cache. Client cache management follows I-LRU (invalid LRU) replacement policy, where invalid cache items are first replaced. If there is no invalid cache item, the valid items are replaced in LRU fashion.

Parameter	Value
Server database size (N)	1000 items
Item size (S_{data})	4096 bits
Client cache size (C)	30-100 items
Number of clients (M)	50
Item id size (S_{id})	32 bits
Update timestamp size (T_{data})	32 bits
Broadcast interval (L)	20 sec
Number of UIR/RR broadcasts (m-1)	4
Broadcast window (w)	10 L
Hot data items (D_H)	1-50
Cold data items (D_c)	Remainder of D
Percentage of access on hot data subset	80
Percentage of updates on hot data subset	40
Mean query generate time (T_q)	10-300 sec
Mean update arrival time (T_u)	1-10000 sec
Mean disconnection time (T_d)	0-400 sec
Client disconnection probability (p_d)	0.10
Bandwidth of uplink channel (B_{up})	19.2 Kbps
Bandwidth of downlink channel (B_{down})	100 Kbps

Table 1 Simulation parameters

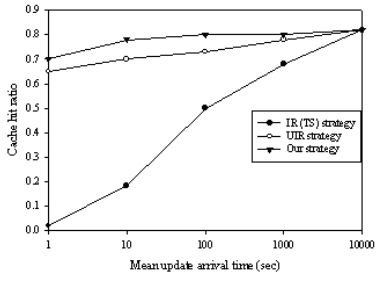
4.2 Results

The effects of different workload parameters such as mean update arrival time, mean query generate time, etc. on the relative performance of the IR (TS), UIR and our strategy have been presented.

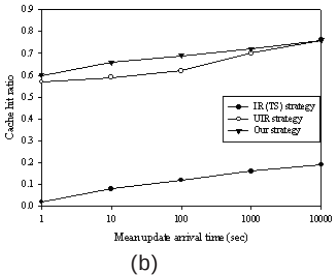
4.2.1 The cache hit ratio

Figure 5 shows that the cache hit ratio of our strategy is always higher than IR and UIR strategies. In the IR algorithm, a client only downloads the data items that it has requested from the server, whereas in UIR and our strategy, a client also downloads the updated data items that are broadcast by the server, thus, increasing the cache hit ratio. Due to cache miss, the UIR strategy broadcasts the requested data items only after the next IR, whereas our strategy also broadcasts the requested data items after every UIR (as part of RR). Because of such distributed query replies, cache hit ratio of our strategy is higher than UIR strategy. Figure 5a shows that cache hit ratio drops with the decrease in mean update arrival time.

As shown in Figure 5b, the cache hit ratio reduces due to client disconnections. All the three strategies react differently to the disconnections. In the IR algorithm, if a client disconnects longer than w*L, it invalidates all cached data items and deletes them from the cache although some of them may be valid. This results in many cache misses for the client before it fills up the cache again.



(a)

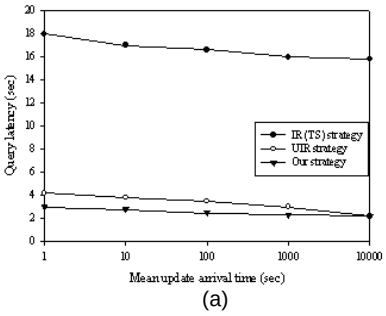


(b)

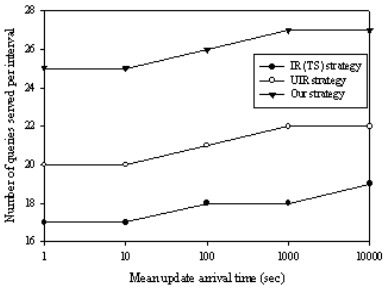
Figure 5 Cache hit ratio as a function of mean update arrival time (T_q = 100 sec, N = 1000 items, C = 50 items). (a) no client disconnection (b) client disconnection probability (p_d) is 0.10 (T_d = 400 sec).

4.2.2 The effect of mean update arrival time

We measure the query latency and throughput (the number of queries served per IR/UR interval) as a function of the mean update arrival time. Figure 6a shows the effect of mean update arrival time over the query latency. Our strategy outperforms the IR and UIR strategies, because a client only needs to wait for the next RR to serve the queries and hence the query latency is about 20/(2*5) = 2 seconds (L/(2*m)).



(a)



(b)

Figure 6 The query latency and throughput as a function of the mean update arrival time (T_q = 100 sec, T_d = 400 sec, N = 1000 items, C = 50 items).

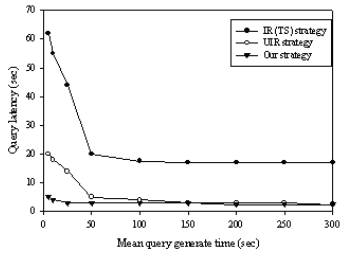
In our simulation model, a query is generated only when there is no pending request from the client. As the query latency drops, the next query may arrive earlier and be served earlier. Thus, the server can serve more queries within one broadcast interval. Since three strategies have different query latencies, they have different throughput. As shown in Figure 6b, our strategy has the highest throughput, whereas IR strategy has the lowest, and UIR strategy in the middle.

4.2.3 The effect of mean query generate time

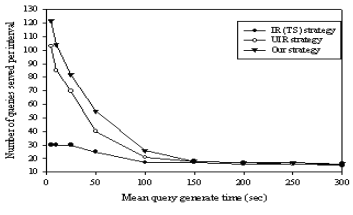
As shown in Figure 7a, the query latency of the IR strategy grows longer than the IR interval when the mean query generate time drops below 50 seconds. Due to the distributed broadcast of requested data, our strategy has always less query latency as compared to UIR strategy. Since our strategy has higher cache hit ratio than the IR and UIR strategies, it can serve more queries locally. Due to *delayed uplink (DU)*, our strategy further reduces the number of uplink requests as compared to UIR and IR strategies. As a result, it has always higher throughput than the IR and UIR strategies when the mean query generate time is very low. For example, as shown in Figure 7b, when the query generate time reduces from 20 seconds to 5, the throughput of the IR algorithm remains at about 30, whereas the throughput of UIR strategy increases from about 70 to about 103. At the same time the throughput of our strategy increases from 82 to 122. When the mean query generate time is very high i.e., around 300 seconds, the broadcast channel has enough bandwidth to serve the client request, and hence three strategies have almost similar throughput.

4.2.4 The number of uplink requests

To evaluate the effectiveness of our strategy to uplink bandwidth utilization, we study the effect of mean update arrival time and mean query generate time over number of uplink requests. As shown in Figure 8, our strategy has the lowest uplink cost, whereas the IR strategy has the highest uplink cost and the UIR strategy is in the middle. This can be explained by the fact that the three strategies have different cache miss ratios. In the IR strategy, when a cached data item becomes invalid, the client has to send the uplink request. Thus, a large number of uplink requests are also resulted due to the invalidation of cache contents. In our strategy, there is no uplink due to updates at the server. As the mean query generate time increases the number of uplink requests decreases because there will be less number of cache misses. In Figure 8b, when the mean query generate time drops below 50 seconds, because of the use of DU technique our strategy behaves significantly better than IR and UIR strategies.

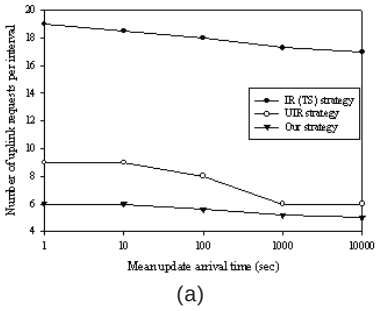


(a)

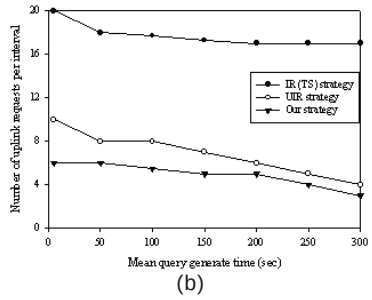


(b)

Figure 7 The query latency and throughput as a function of the mean query generate time (T_u = 10 sec, N = 1000 items, C = 50 items, no client disconnection).



(a)



(b)

Figure 8 The number of uplink requests per interval (N = 1000 items, C = 50 items, no client disconnection). (a) T_q = 50 sec (b) T_u = 100 sec.

4.2.5 The disconnection overhead

Figure 9 shows the effect of different disconnection times on the disconnection overhead. The overhead considered is the additional number of bits transmitted on downlink channel due to cached items those have become invalid during the disconnection. Since our strategy has no overheads due to disconnection less than 200

seconds (w*L), the comparison between UIR and our strategies has been shown for disconnection time longer than 200 seconds. The figure shows that the UIR strategy has always higher overhead than our strategy. As our strategy uses disconnection report (DR), the maximum overhead is C bits. When disconnection time is slightly higher than w*L or has very large value, the overhead is less than C bits as all the cached items may be valid/invalid. In UIR strategy, the number of invalid items increases with the increase of disconnection time, thus increasing the reconnection overhead. For IR strategy, if the disconnection time is longer than 200 seconds, a client assumes that all the cached data items have become invalid and deletes them from the cache. To serve the future requests the client will download the items from the server and thus, the overhead is a very large value as compared to our scheme and hence not shown in the figure.

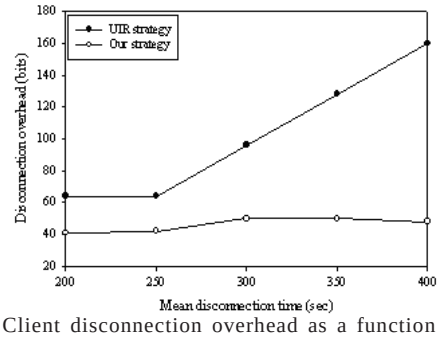
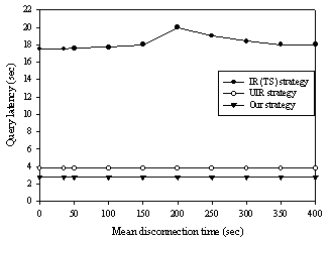


Figure 9 Client disconnection overhead as a function of the mean disconnection time (T_u = 10 sec, T_q = 50 sec, N = 1000 items, C = 50 items).

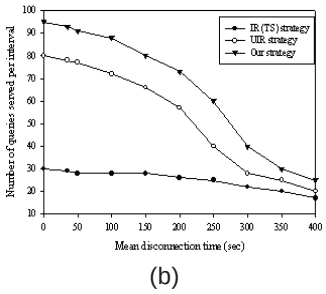
4.2.6 The effect of client disconnection time

Figure 10 shows the effects of the client disconnection time. In our strategy, the client keeps the valid cache items even though it has been disconnected for a long time (larger than w*L seconds). Thus, the query latency of our strategy does not change too much as the client disconnection time changes. Since a client generates less queries when the mean disconnection time increases, the throughput of UIR and our strategies drops as the mean disconnection time increases.

In the IR strategy, if a client disconnects longer than w*L seconds, it has to discard the whole cache. Since the client disconnection time is exponentially distributed, the cache hit ratio of IR strategy drops as the disconnection time grows until w*L = 200 seconds. Since the query latency increases as the cache hit ratio drops, the query latency of the IR strategy grows to almost 20 seconds as the disconnection time increases to 200 seconds.



(a)



(b)

Figure 10 The query latency and throughput as a function of the mean disconnection time (T_u = 100 sec, T_q = 25 sec, N = 1000 items, C = 50 items).

5 Conclusions and Future Work

With the increased use of portable computing devices, the mobile computing has reached a stage that enables foreseeing easy access to information anywhere, anytime. However, it demands solutions to mask unstable connectivity and limited energy. Caching at mobile client can enhance the data access performance in such environments. To overcome the demerits of existing IR and UIR strategies, this paper presents a synchronous stateful cache invalidation strategy that reduces the query latency, disconnection overheads, use of wireless channel and conserve the client battery energy. Simulation experiments show that our strategy performs better than IR and UIR schemes. Use of prefetching in integration with caching is a consideration during our future research.

Acknowledgements

The authors would like to thank the anonymous reviewers and guest editors whose insightful comments helped to improve the presentation of the paper.

References

1. Cao, G. (2002). On Improving the Performance of Cache Invalidation in Mobile Environments. ACM/Kluwer Mobile Networks and Applications, 7(4), 291-303.
2. Kahol, A., Khurana, S., Gupta S.K.S., & Srimani, P.K. (2001). A Strategy to Manage Cache Consistency in a Disconnected Distributed Environment. IEEE Transaction on Parallel and Distributed Systems, 12(7), 686-700.
3. Hu, Q., & Lee, D.K. (1998). Cache Algorithms Based on Adaptive Invalidation Reports for Mobile Environments. Cluster Computing, 1(1), 39-50.
4. Cao, G. (2001). A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments. ACM International Conference on Computing and Networking (Mobicom), 200-209.
5. Cao, G. (2002). Proactive Power-Aware Cache Management for Mobile Computing Systems. IEEE Transactions on Computers, Vol. 51, No. 6.
6. Wu, K.L., Yu, P.S., & Chen, M.S. (1998). Energy-Efficient Mobile Cache Invalidation. Distributed and Parallel Databases, Kluwer Academic Publishers, Vol. 6, 351-372.
7. Tan, K.L. (2001). Organisation of Invalidation Reports for Energy-Efficient Cache Invalidation in Mobile Environments. Mobile Networks and Applications, 6, 279-290.
8. Yuen, J.C., Chan, E., Lam, K., & Lueng, H.W. (2000). Cache Invalidation Scheme for Mobile Computing Systems with Real-Time Data. SIGMOD.
9. Kahol, A., Khurana, S., Gupta S.K.S., & Srimani, P.K. (2000). An Efficient Cache Maintenance Scheme for Mobile Environment. Proceedings of Int. Conf. on Distributed Computing Systems, 530-537.
10. Jing, J., Elmagarmid, A., Helal, A., & Alonso, R. (1997). Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/ Server Environments. Mobile Networks and Applications, 115-127.
11. Yao, J.F., & Dunham, M.H. (2001). Caching Management of Mobile DBMS. Journal of Integrated Computer-Aided Engineering, Vol. 8, No. 2.
12. Hou, W.C., Su, M., Zhang, H., & Wang, H. (2001). An Optimal Construction of Invalidation Reports for Mobile Databases. Proceedings of CIKM, 458-465.
13. Nam, S.H., Chung, Y., Cho, S.H., & Hwang, C.S. (2002). Asynchronous Cache Invalidation Strategy to Support Read-Only Transactions in Mobile Environments. IEICE Trans. Inf. and Syst., Vol. E85-D, No. 2.
14. Barbara, D., & Imielinski, T. (1994). Sleepers an d Workaholics: Caching Strategies in Mobile Environments. Proceedings of the ACM SIGMOD Conference on Management of Data, 1-12.
15. Cao, G. (2003). A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments. IEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 5, 1251-1265.
16. Lee, S.K. (2002). Caching and Concurrency Control in a Wireless Mobile Computing Environment. IEICE Trans. Inf. and Syst., Vol. E85-D, No. 8.
17. Tan, K.L., Cai, J., & Ooi, B.C. (2001). An Evaluation of Cache Invalidation Strategies in Wireless Environments. IEEE Transactions on Parallel and Distributed Systems, Vol. 12, No. 8.
18. Lai, K. Y., Tari, Z., & Bertok, P. (2003). Cost Efficient Broadcast Based Cache Invalidation for Mobile Environments. SAC, 871-877.

19. Lai, K. Y., Tari, Z., & Bertok, P. (2003). An Analytical Study of Broadcast Based Cache Invalidation in Mobile Computing Networks. CoopIS/DOA/ODBASE, 554-572.
20. Chuang, P.J., & Hsu, C.Y. (2004). An Efficient Cache Invalidation Strategy in Mobile Environments. IEEE International Conference on Advanced Information Networking and Application (AINA).
21. Wang, Z., Das, S. K., Che, H., & Kumar, M. (2003). Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments. ICDCS Workshops, 797-802.
22. Wang, Z., Kumar, M., Das, S. K., & Shen, H. (2003). Investigation of Cache Maintenance Strategies for Multi-cell Environments. Mobile Data Management (MDM), 29-44.
23. Chand, N., Joshi, R. C., & Misra, M. (2004). Broadcast Based Cache Invalidation and Prefetching in Mobile Environment. Intl. Conf. on High Performance Computing (HiPC), Springer LNCS 3296, 410-419.
24. Chand, N., Joshi, R. C., & Misra, M. (2005). Energy Efficient Cache Invalidation in Wireless Mobile Environment. Proceedings of IEEE International Conference on Personal Wireless Communications (ICPWC), 244-248.
25. Chand, N., Joshi, R. C., & Misra, M. Energy Efficient Cache Invalidation in a Disconnected Wireless Mobile Environment. International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC), to appear.