Adaptive Cache Invalidation Methods in Mobile Environments

Qinglong Hu Dik Lun Lee
Department of Computer Science
University of Science and Technology
Clear Water Bay, Hong Kong
qinglong@cs.ust.hk, dlee@cs.ust.hk

Abstract

Caching of frequently accessed data items can reduce the bandwidth requirement in a mobile wireless computing environment. Periodically broadcast of invalidation reports is an efficient cache invalidation strategy. However, this strategy is severely affected by the disconnection and mobility of the clients. In this paper, we present two adaptive cache invalidation report methods, in which the server broadcasts different invalidation reports according to the update and query rates/patterns and client disconnection time while spending little uplink cost. Simulation results show that the adaptive invalidation methods are efficient in improving mobile caching and reducing the uplink and downlink costs without degrading the system throughput.

1. Introduction

Wireless mobile computing is gaining more and more popularity in recent years, because it can satisfy people's information needs at any time and anyplace. The advent of laptop and notebook computers further induces growing demands for mobile and nomadic computing. Unlike traditional client-server architectures, where the locations and connections of clients do not change, mobile computers will often be in a disconnected state for long periods of time due to limitations on battery technologies. They may also frequently move between cells and connect to different data servers at different time.

In [12], packet efficiency and power efficiency are proposed as two criteria in designing various algorithms for mobile environments. Packet efficiency means the ability of the algorithm to minimize the total number of packets sent on a wireless link, whereas power efficiency means the ability to minimize the battery power consumed by the client using the algorithm. The communication channel consists of a downlink channel and an uplink channel. The server can have two basic modes of information dissemination with corresponding information access methods for the clients:

1. Publishing Mode: Data is multicasted periodically on the downlink channel. Accessing multicast data does not require uplink transmission and is "listen only". Querying involves simple filtering of the incoming data stream according to a user specified filter.

2. On-Demand Mode: The client requests a piece of data through the uplink channel and the server responds by sending the data to the client on the downlink channel.

On-demand mode is the traditional client-server approach, while publishing mode is more suitable for mobile environments. Broadcasting the most frequently accessed data items saves bandwidth since the large number of users requesting the data items can be served at no additional cost. Thus, it is packet efficient. It also saves the client's energy by avoiding energy-consuming uplink transmissions and thus is power efficient. On the other hand, on-demand channels allow clients to have immediate access to data. Further, the server can keep an accurate picture of the kind of information accessed by the clients.

Caching of frequently accessed data items on the client side is an important technique to reduce the number of uplink messages for a particular data item, and the number of downloads of the same data item. On the other hand, the disconnection and mobility of the clients make cache consistency a challenging problem. Effective cache invalidation strategies must be developed to ensure the consistency between the cached data in the mobile computers and the original data stored in the server. Generally, there are three basic ways to design invalidation strategies:

1. The server is stateful. It knows which data are cached by which mobile clients. Each time a data item is changed, the server will send an invalidation message to those clients which cached that particular item. This method requires the server to locate the clients. Since disconnected mobile clients cannot be contacted by the

server, the disconnection of a mobile client automatically means its cache is no longer valid upon reconnection. Once a mobile client moves to another cell, it has to notify the server of this relocation. The mobility, disconnection of the clients and updating of data items will incur uplink and downlink messages.

2. The clients query the server to verify the validity of their caches whenever any cached data is used. This method generates a lot of uplink traffic in the network.

3. The server is stateless, since it is not aware of the state of the client's cache and the client itself such as location and disconnectivity of the clients. The server simply periodically broadcasts an invalidation report containing the data items that have been updated recently. The client determines if its cache is valid by listening to the report, going uplink only if the cache validity is no longer guaranteed.

D. Barbara and T. Imielinski provided three cache invalidation schemes, namely, Broadcasting Timestamps (TS), Amnesic Terminals (AT) and Signatures (SIG), which use different invalidation reports for a stateless server [4],[5]. These algorithms are effective only for clients who have not been disconnected for a period that exceeds an algorithm-specified parameter (e.g. broadcast window w and broadcast period L) or if the number of updated items during the period is not greater than an algorithm-specified parameter. However, it is possible in these methods that some of the cached objects are still valid after a long disconnection period or a large number of updates at the server. Thus, these methods doesn't utilize the bandwidth efficiently.

Jin Jing proposed a bit-sequence (BS) scheme where the invalidation report consists of a set of binary bit sequences with an associated set of timestamps [13]. A client will invalidate its entire cache only if more than half of the data items have been updated in the server since its last validation time. Since no algorithm-specified parameter is required, changing workload parameters has little impact on the performance. However, this method requires much greater invalidation report size than that in TS or AT methods, especially when the database size is large.

K.L Wu [16] proposed a scheme which modifies the TS or AT algorithms to include cache validity checks after reconnection. The side effect of this method is that it requires uplink bandwidth and an update history window of the past W broadcast intervals ($W \geq w$ in GCORE) to be specified, which introduces the same basic problem as in the TS method (e.g. when the disconnection time is greater than W, nothing can be salvaged). The effectiveness of the algorithm also depends heavily on the update pattern on the data items.

In this paper, we introduce the adaptive cache invalidation scheme which is a hybrid of the above algorithms and a method to adjust window size according to query and update patterns on the data items. The purpose is to make better use of the algorithms proposed in these papers to achieve an optimized overall performance in terms of packet and power efficiency. The strategy is to use as little uplink communication as possible, without degrading the system throughput (the number of queries answered in unit time). Low uplink cost is important because the uplink capacity from clients back to servers is much smaller than the downlink capacity from servers to clients [1]. This situation is also referred to as *Asymmetric Communication Environments*. Furthermore, uplink transmission requires much higher power from clients than downlink reception does — the power needed for transmission is proportional to the forth power of the distance between the two communicating entities [12] — and thus consumes valuable battery power in the clients. It is important to avoid transmission from clients as much as possible while maintaining the system throughput.

The rest of the paper is organized as follows. Section 2 discusses the cache invalidation algorithms for stateless server. Section 3 presents the two cache invalidation methods. Section 4 describes the simulation model. Section 5 does the simulation study. Section 6 gives conclusion of the paper.

2. Cache Invalidation Algorithms for Stateless Server

A mobile computing environment consists of two sets of entities: clients and fixed hosts. Some of the fixed hosts, called *Mobile Support Station* (MSS), are augmented with a wireless interface to communicate with clients. The whole geographic area is partitioned into *cells*. Each cell is covered by a MSS with a large number of clients residing in a cell. Each mobile host can move within a cell or between cells while retaining its network connection. A mobile host can either connect to a server (uplink) through a wireless communication channel or disconnect from the server by operating in the doze mode (power-off). Each MSS can only service users who are currently located in the cell. We assume that the database is updated only by the server. The database is a collection of N named data items and is replicated at each server. A data item is the basic unit for update and query. Mobile hosts only issue simple requests to read the most recent copy of a data item.

To ensure cache consistency, the server periodically broadcasts invalidation reports. Every client, if active, listens to the reports and invalidates its cache accordingly. Data items frequently referenced by queries are cached in the mobile hosts. To answer a query, the client on a mobile host will listen to the next invalidation report and use the report to decide whether its cache is still valid or not. If there is a valid cached copy of the requested data item, the client will return the item immediately. Otherwise, it will go uplink with the query. The server keeps track of the recently updated information and broadcasts an invalidation report every L second. In this paper, we denote the most recent invalidation report broadcasted as IR.

2.1. No-Checking Caching Strategy

The no-checking scheme (Figure 1) is similar to the broadcasting timestamp (TS) proposed in [4],[5]. The IR consists of 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, t_i is the corresponding most recent update timestamp and w is the invalidation broadcast window. In other words, IR contains the update history of the past w broadcast intervals. In order to save energy, clients are usually required to operate in a doze mode or even to completely disconnect for a prolonged period of time. In the latter, a client in the mobile host may miss some invalidation reports. The timestamp of the latest invalidation report received by the client (denoted by T_{lb}) is maintained on the client side so that after a mobile host wakes up from a disconnection, it knows the timestamp of the latest report received and decides on whether the reports it missed are all included in the current window w or not. In the no-checking scheme, if a mobile host has been disconnected for more than w broadcast intervals, the entire cache contents has to be discarded upon reconnection, since the client can't tell which parts of the cache are valid, even though some of the cached contents may still be valid. This will significantly increase the required bandwidth and is neither packet efficient nor power efficient.

2.2. Checking Caching Strategy

It is possible that the client may still have many valid data items in its cache after a long period of disconnection. In this case, dropping the entire cache as it is done in the no-checking algorithm will degrade the system performance significantly.

In order to retain valid data items in the cache, we must identify which part of the cache is still valid. There are several approaches to this problem with different trade-offs. One possible solution presented in [16] is that the mobile host sends back to the server the ids of all the cached data items and their corresponding timestamps. Then the server identifies which data items are valid and returns a validity report to the client. However, this requires a lot of uplink bandwidth and thus is not power efficient.

$if\ (T_{lb} < (T_i - L \times w))$

drop the entire cache

$else\ for$ **every item** o_j **in the cache**

if **(**$o_j \in IR$**) and (**${t_j}^c < t_j$**)**

throw o_j **out of the cache**

$else\ {t_j}^c = T_i$

$for\ \forall o_j \in Q_i$

$if\ o_j$ **is in the cache**

use the cache's value to answer the query

$else$ **go uplink with the query**

$T_{lb} := T_i$

Figure 1. TS **Client Cache Invalidation Algorithm.**

2.3. Bit-Sequences Algorithm

The problem with TS (or AT) algorithm is that it requires a fixed window size w, which cannot be adapted according to workload parameters. In practice, the client reconnecting to the server after long disconnection needs a larger update history window w to tell which parts of the cache are valid. Large window size w means that IR includes more messages and takes up more bandwidth of the broadcast channel. On the other hand, if the client is connected most of the time, a small broadcast window w is sufficient.

In the bit-sequences approach, IR consists of a set of bit sequences, each of which has a corresponding timestamp. Each bit represents a data item in the database. A "1" bit in a sequence means that the data item represented by the bit has been updated since the time specified by the timestamp of the sequence. A "0" bit means that the item has not been updated since that time. The set of sequences is further organized in a hierarchical structure with B_n in the structure having N bits, which correspond to N data items in the database (i.e. each bit in B_n represents a data item). As many as half of the bits ($N/2$) in B_n can be set to "1" to indicate that $N/2$ items have been updated (initially, the number of "1" bits may be less than $N/2$). The timestamp of the sequence B_n, denoted by $TS(B_n)$, indicates the time after which these $N/2$ items have been updated. The next sequence, denoted by B_{n-1}, in the structure has $N/2$ bits. The kth bit in B_{n-1} corresponds to the kth "1" bit in B_n. $N/2^2$ bits can be set to "1" to indicate that $N/2^2$ items have been updated since $TS(B_{n-1})$. Generally, bit sequence B_k ($k = 1, \ldots, n, 2^n = N$) in the structure consists of $|B_k|$ $=N/2^{n-k}$ bits of which as many as $N/2^{n-k+1}$ data items have been updated after the timestamp $TS(B_k)$. An additional dummy sequence B_0 is used, where $TS(B_0)$ indicates the time after which no data item has been updated.

$if\ TS(B_0) \leq T_{lb}$

no cache is to be invalidated and stop.

$if\ T_{lb} < TS(B_n)$

the entire cache is invalidated and stop.

Locate bit sequence B_j with timestamp $TS(B_j) \leq T_{lb} \leq TS(B_{j-1})\ \forall j\ (1 \leq j \leq n)$.

Invalidate all the data items represented by the "1" bits in B_j.

Figure 2. BS Client Cache Invalidation Algorithm.

Bit-sequence structure contains more update history information than updating window w. For example, it can record as many as half of the data items of the database being updated. Since it does not require algorithm specified parameter w, it is more flexible. A client will invalidate its entire cache only if more than half of the data items have been updated in the server since its last validation time. In general, changing workload parameters such as disconnection time, update rates, query rates, etc., has little impact on the performance of the BS algorithm. However, there is one disadvantage of BS compared to TS and AT. When disconnection is rare or short, the update history contained in the invalidation report for TS is enough for cache invalidation. The BS algorithm actually wastes downlink broadcast bandwidth and causes clients to spend more time in active mode for receiving invalidation reports.

3. Adaptive Invalidation Report

To avoid wasting bandwidth of the downlink broadcast channels and to save bandwidth of uplink broadcast channels, we proposed two methods which combine the advantages of the previous methods by dynamically adjusting the invalidation reports according to system workload (e.g. update, query pattern and disconnection frequency and time).

3.1. Adaptive Invalidation Report with Fixed Window

The first method is called Adaptive Invalidation Report with Fixed Window, AFW. In AFW, IR consists of two types of information:

1. Update history for window w, denoted by $IR(w)$. The report size for $IR(w)$ is $n_w(log(N) + b_T)$.
2. Bit-sequences structure, denoted by $IR(BS)$. The report size for $IR(BS)$ is $2N + b_T log(N)$.

In the above, n_w is the number of data items updated within w broadcast periods, b_T is the size of timestamp, and N is the number of data items in the database. In the following, we denote the database as D, clients set as C and T^i_{lb} as the latest timestamp at which client i received its invalidation report.

The server begins to broadcast the invalidation report periodically at time $T_i = iL$. The server keeps a list $IR(w_i)$ and a set of bit sequences $IR(BS)$ defined as follows.

- $IR(w_i) = \{(o_j, t_j) \mid o_j \in D$ and $T_i - w \times L \leq t_j \leq T_i\}$ [1]
- $IR(BS) = \{B_n, TS(B_n), \ldots, B_0, TS(B_0)\}$

Queries from long disconnected clients which reconnect during the intervals between T_{i-1} and T_i can be processed as follows. If the next invalidation report at T_i is $IR(BS)$, then client k runs BS cache invalidation algorithm after receiving the report. If the next invalidation report at T_i is $IR(w)$, the client simply sends its T_{lb} to the server. The server in return decides whether to use $IR(BS)$ or $IR(w)$ as the next invalidation report at T_{i+1} by comparing $TS(B_n)$ with T_{lb}. If there is one T_{lb} received by the server such that $TS(B_n) \leq T_{lb}$, the server will choose $IR(BS)$. Otherwise, it will choose $IR(w)$.

Our method guarantees that BS is broadcast as the next invalidation report only if there is at least one client which needs more update history information than the window w can provide. Otherwise, window w is broadcast whenever it is possible to save the bandwidth of the broadcast channel. Obviously changing workload parameters such as clients disconnection time has less impact on the performance of the system. The algorithms running on the server and clients are depicted in Figure 3. Since only the latest timestamp T_{lb} needs to be sent to the server, the uplink bandwidth required for this method is much smaller compared to that for TS with checking method.

[1] t_j is the timestamp of the last update of data item o_j.

Server:

if ($\exists i \in C$ **and** $TS(B_n) \leq T_{lb}^i < T - w \times L$)

the server selects $IR(BS)$ as the next invalidation report

$else$ **the server broadcasts $IR(w_i)$ as the invalidation report.**

Clients:

if (**report type is** $IR(BS)$)

run BS client cache invalidation algorithm

$else$ if ($T_{lb} < T_i - w \times L$ **and not yet sent T_{lb} to server = TRUE**)

send T_{lb} to the server

$else$ **invalidate its cache by running TS cache invalidation algorithm.**

Figure 3. Adaptive Client Cache Invalidation Algorithm AFW.

3.2. Adaptive Invalidation Report with Adjusting Window

Since the bit-sequences structure contains the update information of the whole database, it is good for mobile hosts with long disconnections to invalidate their caches. This is because the longer the disconnection time is, the more update history the mobile host requires in order to salvage the data items cached. However, for disconnections that are barely longer than the window w, the use of BS invalidation reports is still quite wasteful. It is sufficient just to adjust the window size w of the TS algorithm a little bit so that the TS report can be used for a mobile host to invalidate its cache, thus saving the downlink cost for invalidation reports. For very long disconnection, adjusting the window size alone may result in a large number of update information and consequently large invalidation reports to be broadcast. Thus, the BS report should be a better choice. Integrating these three types of reports (i.e., default TS, varying window size, and BS) yields another adaptive cache invalidation algorithm, which we call adaptive with adjusting window (AAW).

AAW adapts the window size w in the TS algorithm according to workload parameters. In this method, adaptive TS has window size w or the smallest update window history as the default size. It will be enlarged for the next invalidation report whenever there is at least one client with its T_{lb} greater than the default w. Otherwise, it retains its default size. To keep the invalidation report size small, we do not explicitly include in each report the window size for the client to be able to correctly invalidate its cache. Instead, if the server wants to enlarge the next report, in addition to including all (o_i, t_i) such that $t_i > T_{lb}$ ($T_{lb} < (T - w \cdot L)$) in the report, it adds one more dummy record $(dummy_{id}, T_{lb})$ in the next invalidation report where $dummy_{id}$ is a special id not used by any data as ids. Upon receipt of the report, client k checks to see whether its T_{lb}^k is within w or not. If it is, then TS does the normal invalidation check. Otherwise, it checks to see whether the report includes the dummy $(dummy_{id}, T_{lb})$ and whether $T_{lb} < T_{lb}^k$. If this is the case, the normal invalidation check is performed. Otherwise, it sends back its T_{lb}^k to the server to request more updating information. On the server side, when T_{lb}^k is received, the server determines what is to be used as the next report based on whether the extended window is larger than BS or not. The algorithm is given in Figure 4, where $IR(w')$ contains all the update information since timestamp T_{lb}^i.

Obviously the adaptive window size of the TS algorithm is an effective cache invalidation approach. The approach in [5] adjusts the window size for each data item according to changes in update rates and reference frequencies of the item. It requires uplink communication for the server to obtain feedback information about the access patterns from the clients. Further, the window size w must be contained in each report for the client to be able to correctly invalidate its cache. However, no detailed algorithm was given in [5] to show how the window size information is included in the invalidation reports.

4. Simulation Model

The performance analysis given in this section is designed to obtain the communication costs of a set of clients under the coverage of the same server. The costs include the downlink cost of broadcasting invalidation reports, the downlink cost of sending data and invalidation reports to the clients, and the uplink costs of validity checking and requesting missing objects for the basic BS algorithm proposed in [13], simple-checking caching scheme given in [16] and the two adaptive cache invalidation schemes. The study does not include TS and AT methods proposed in [4],[5] and grouping with cold update-set retention (GCORE) proposed in [16], because they are not applicable to clients with long disconnections. The effectiveness of a cache scheme is measured by the throughput (i.e., number of queries answered in unit time) and uplink cost due to cache validation processing.

The model used in this paper is similar to that employed

Server:

if ($\exists i \in C$ **and** $TS(B_n) \leq T^i_{lb} < T - w \times L$)

if (**size of** $IR(BS) >$ **size of** $IR(w')$)

the server selects $IR(w')$ **as the next invalidation report**

$else$ **the server selects** $IR(BS)$ **as the next invalidation report**

$else$ **the server broadcasts** $IR(w)$ **as the invalidation report.**

Clients:

if (**report type is** $IR(BS)$)

run BS **client cache invalidation algorithm**

$else$ if ($T_{lb} < T_i - w \times L$ **and** $T_{lb} < t$ **with** $dummy_{id}$ **as its id and not yet sent** T_{lb} **to server =TRUE**)

send T_{lb} **to the server**

$else$ **invalidate its cache by running** TS **cache invalidation algorithms.**

Figure 4. Adaptive Client Cache Invalidation Algorithm AAW.

in [13]. It consists of a single server that services multiple mobile clients. The database objects are only updated in the server. Updates are separated by an exponentially distributed update interarrival time. The server also broadcasts an invalidation report periodically. The queries are read-only and generated in each client. The arrival of a new query is separated from the completion of the previous query by either an exponentially distributed think time or an exponentially distributed disconnection time. Our model assumes that each client may enter into a disconnection mode with a probability p in each broadcast interval. If the referenced data items are not cached in the mobile host, it sends the requested object ids to the server to fetch the data item.

The size of the client buffer pools is specified as a percentage of the database size. Cached data items are managed using an LRU replacement policy. The network is modeled with invalidation reports having the highest priority, checking requests and validity reports coming next and followed by all the other messages which are of equal priority and served on a first-come first-served basis. This strategy ensures that invalidation reports will always be broadcast at the exact broadcast period as specified in the model and short response time for queries.

In the experiment, we assume that the bandwidth is always fully utilized so that the throughput represents the true efficiency of the algorithm and is not influenced by the mobile host disconnection time and frequency. The default parameters and settings employed in the simulation are listed in Table 1. All simulations were performed on Sun Sparc Workstations running SunOS using the CSIM simulation package [14].

Parameter	Setting
Simulation Time	100000 seconds
Number of Clients	100 mobile client hosts
Database Size	1000 to 80000 data items
Data Item Size	8192 bytes
Client Buffer Size	1 % or 2 % of database size
Broadcast Period	20 seconds
Network Downlink Bandwidth	10000 bits per second
Network Uplink Bandwidth	1 % to 100 % that of downlink bandwidth
Control Message Size	512 bytes
Mean Think Time	100 seconds
Mean Data Items Ref. by a Query	10 data items
Mean Data Items Updated by a tran.	5 data items
Mean Update Arrive Time	100 seconds
Mean Disconnect Time	200 to 8000 seconds
Prob. of Client Disc. per Interval	0.1 to 0.8
Window for Broadcast Invalidation	10 intervals

Table 1. System Parameter Settings.

Parameter	UNIFORM	HOTCOLD
HotQueryBounds	-	from 1 to 100 for each client
ColdQueryBounds	all DB	remainder of DB
HotQueryProb	-	0.8
HotUpdateBounds	-	-
ColdUpdateBounds	all DB	all DB
HotUpdateProb	-	-

Table 2. Query/Update Pattern.

5. Simulation Results

Experiments were run using different data access patterns and workloads, such as client query patterns and server update patterns. For each client, two database regions, hot query set and cold query set, can be specified. Hot query probability refers to the probability that a query will address a data item in the client's hot database region. Data items in the hot query region are randomly chosen. For the data server, the hot update region and cold update region can be specified. The hot update probability q specifies the probability that an update will be performed on a data item in the server's hot database region.

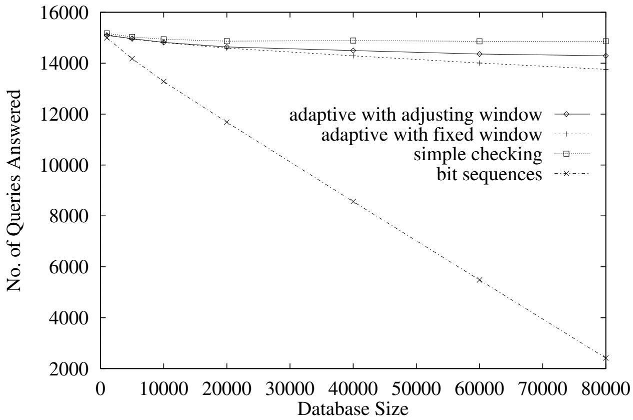
The performance analysis presented here is designed to compare the effects of different workload parameters such as query/update patterns, probability of disconnection and system parameters such as database size, number of the clients, etc., on the relative performance of the BS algorithm in [13], TS with checking in [16] and the adaptive methods. In each experiment, we compare the total number of queries answered by the system within the given simulation time under the condition that the network is fully utilized, and uplink bandwidth cost for validity checking. Unless otherwise specified, the default values for most of the simulations are provided in Table 1. Table 2 presents the workloads associated with query and update patterns employed in this study.

The first experiment is for UNIFORM workload, where each query reads a data item chosen uniformly from among all the data items in the database. Also, each update request chooses a data item uniformly from all the data items. Both query and update parameters display no locality of access. Caching has little effect in improving system performance because of low hit ratio.

The throughput is obtained for various database sizes. From Figure 5, we see that the throughput for BS method goes down rapidly as the database size increases. The other three methods are much less influenced by the database size, with TS method performing better than the two adaptive methods. Also, AAW has better throughput than AFW. However, when we consider the uplink cost for validity reports (see Figure 6), BS method consumes no cost. The two adaptive methods spend similar uplink cost, which is much less than that of the TS method. TS's uplink cost also goes up as the database size increases, while the adaptive methods have rather stable uplink costs. This indicates that the adaptive methods achieve an excellent balance between throughput and uplink cost.

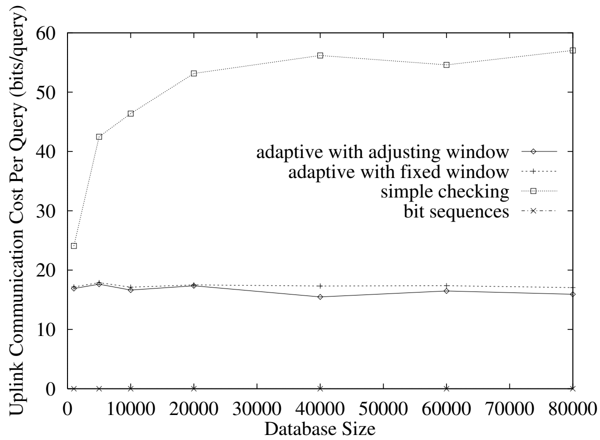
Figures 7 and 8 show the result obtained using the same setting but fixing the database size at 10000 and varying the disconnection frequency. Fig. 7 shows that the throughput for all algorithms except the BS method goes down slightly as the disconnection frequency becomes high and that the relative throughput for the method is the same as that in Figure 5 and 6. Increasing disconnection frequency will induce more uplink cost for both TS with checking and the adaptive methods. However, Figures 7 and 8 indicate that AAW is better than AFW, showing the effect of the adaptive window size on both throughput and uplink cost. When we vary the mean disconnection time as shown in Figure 9 and 10, we obtain similar performance result. Again, this experiment shows the superior balance of the adaptive methods on throughput and uplink costs.

In the HotCold workload, we use the same setting as in the experiment shown in Figures 5 - 10. The only difference is the query pattern. In other words, an update transaction chooses a data item uniformly from all data items and, for simplicity, we assume data items from 1 to 100 are hot query items to which 80% of each client's requests are directed; the other 20% of the requests are directed to



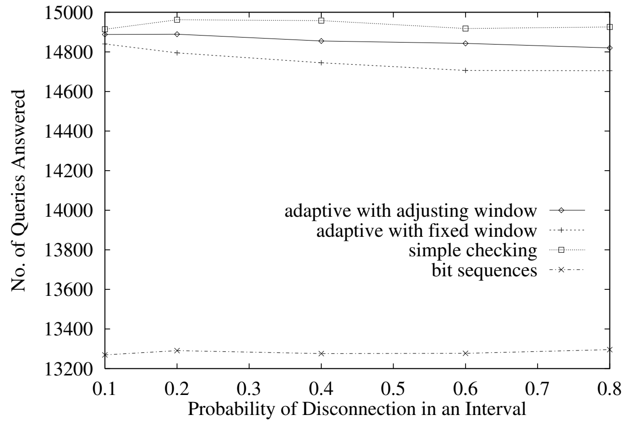
Prob of Disc=0.1, Mean Disc Time=4000, Client Buffer Size=2%

Figure 5. UNIFORM Workload.



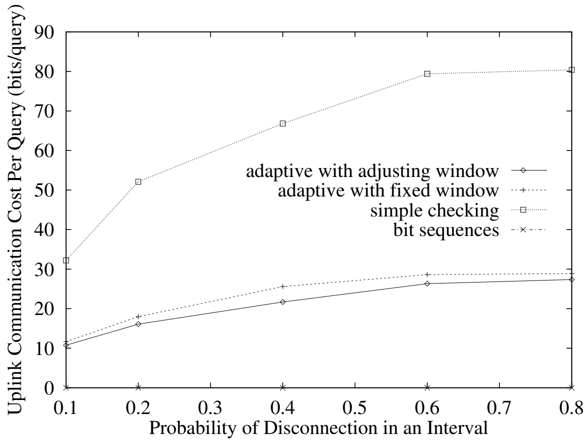
Prob of Disc=0.1, Mean Disc Time=4000, Client Buffer Size=2%

Figure 6. UNIFORM Workload.



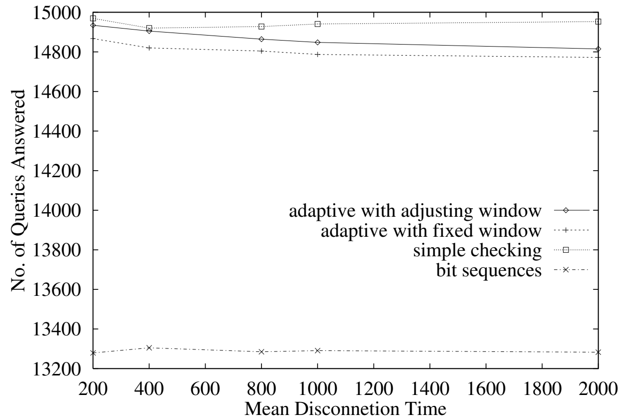
Database Size=10^4, Mean Disc Time=400, Client Buffer Size=2 %

Figure 7. UNIFORM Workload.



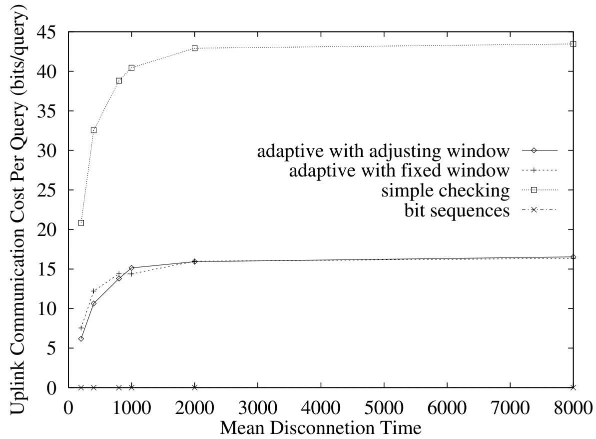
Database Size=10^4, Mean Disc Time=400, Client Buffer Size=2 %

Figure 8. UNIFORM Workload.



Database Size=10^4, Prob of Disc=0.1, Client Buffer Size=1 %

Figure 9. UNIFORM Workload.



Database Size=10^4, Prob of Disc=0.1, Client Buffer Size=1 %

Figure 10. UNIFORM Workload.

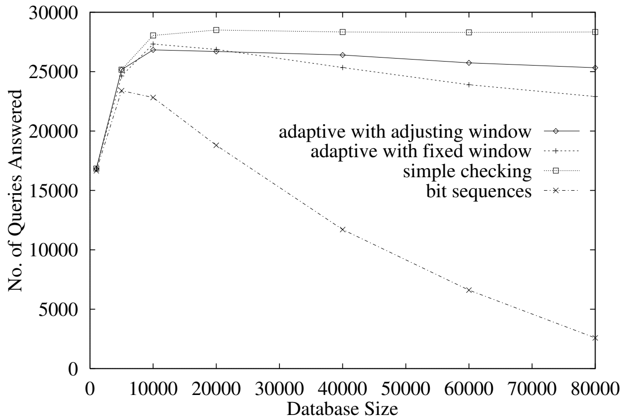
elsewhere in the database. Thus, significant benefits can be expected from caching, which is shown in Figures 11 - 14.

Note in Fig. 11 that the throughput for all four methods is low when the database size is less than 5000. This is because the cache size is chosen to be 2% of the database size. Thus, when the database size is less than 5000, the cache size is smaller than the hot query region and adversely affects the caching efficiency. The throughput for the four algorithms has similar relationship with each other, with BS method being the worst and the other three comparable to each other. TS with checking always has best throughput but requires highest uplink cost. AAW comes in second in throughput and the third is AFW. These two adaptive methods require similar amount of uplink cost for validity reports, which is much less than that for TS.

So far, we assume that the uplink channel has the same bandwidth capacity as the downlink channel. In real world, it may not be the case — the channel available to clients for transmission is typically much smaller than that available to the server. Figures 15 and 16 show the performance of the four methods under uniform and hotcold workloads when the uplink bandwidth is only a fraction of the downlink bandwidth. We observe that when the uplink bandwidth is lower than certain value (200 bits/second), the two adaptive methods have better throughput performance than TS for both uniform and hotcold workload.

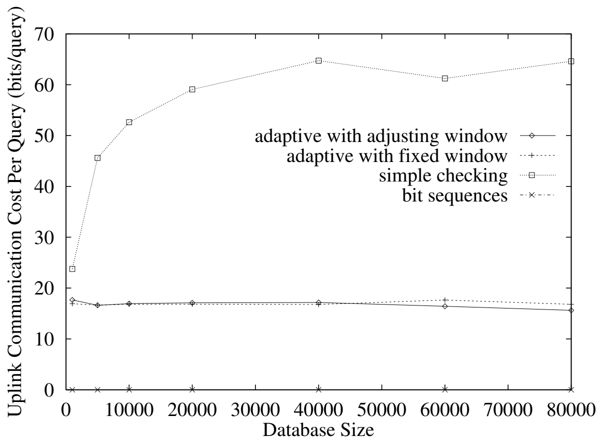
6. Conclusion

In a mobile environment, an ideal caching algorithm should be intelligent in the sense that it must be able to adapt dynamically according to information available from the clients (e.g., client disconnection time and frequency, and query patterns), and from the server (e.g., data item up-



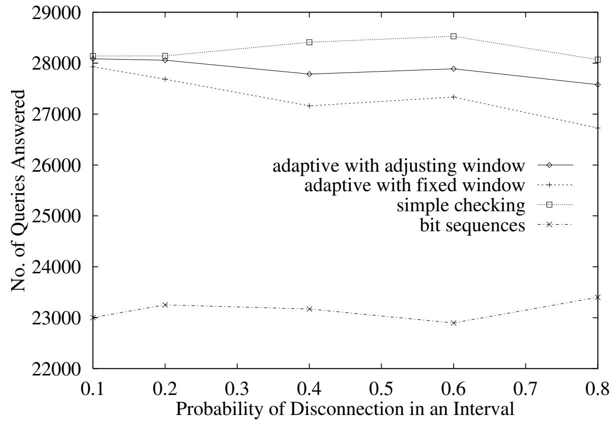
Database Size=10^4, Mean Disc Time=400, Client Buffer Size=2 %

Figure 11. HotCold Workload.



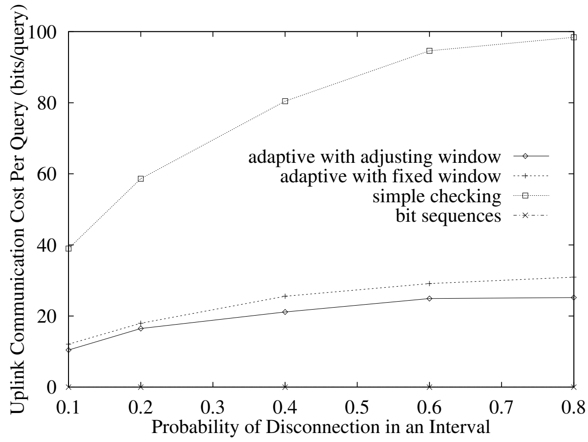
Database Size=10^4, Mean Disc Time=400, Client Buffer Size=2 %

Figure 12. HotCold Workload.



Database Size=10^4, Mean Disc Time=400, Client Buffer Size=2 %

Figure 13. HotCold Workload.



Database Size=10^4, Mean Disc Time=400, Client Buffer Size=2 %

Figure 14. HotCold Workload.

date pattern). BS algorithm adapts itself dynamically as update rate/pattern varies, but it does not make use of the disconnection state information from the clients side. On the other hand, TS with checking method utilizes the information from the client side, but it does not adapt itself intelligently as update rate/pattern and clients disconnection time change. This is because it requires a fixed specified update history window w, which cannot be adjusted according to unexpected workload changes.

The adaptive methods proposed in this paper make use of workload information from both clients and server so that the system workload has less impact on the system performance while maintaining low uplink and downlink bandwidth requirement. Further, they achieve very good balance between throughput and uplink costs.

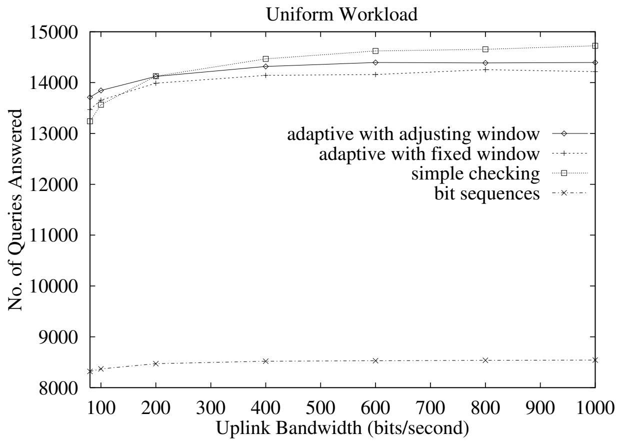
In this paper, we assume only one channel for downlink and one channel for uplink communication. As future work, we will develop caching strategies for the multiple-channel environment, where some channels are assigned as broadcast channels while others are point-to-point channels, and investigate their performance.

7. Acknowledgment

The authors would like to thank the members of the Ubiquitous Information Systems, Drs. Bo Li, Manhoi Choy, Tin-Fook Ngai, and D.Y. Yeung, for their valuable comments.

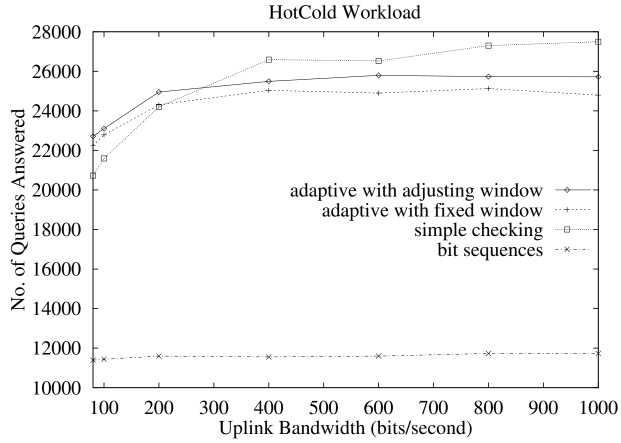
References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric



Database Size=4×10^4, Mean Disc Time=4000, Client Buffer Size=2 %

Figure 15. Asymmetric Communication Environment.



Database Size=4×10^4, Mean Disc Time=4000, Client Buffer Size=2 %

Figure 16. Asymmetric Communication Environment.

communications environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, San Jose, California, 1995.

[2] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval. *ACM Transactions on Database Systems*, 15(3):359–384, September 1990.

[3] R. Alonso and H. Korth. Database issues in nomadic computing. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 388–392, 1993.

[4] D. Barbara and T. Imielinksi. Sleepers and workaholics: Caching strategies for mobile environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1–12, 1994.

[5] D. Barbara and T. Imielinksi. Sleepers and workaholics: Caching strategies for mobile environments (extended version). *MOBIDATA: An Interactive journal of mobile computing*, 1(1), Nov. 1994.

[6] A. K. Elmargarmid, J. Jing, and T. Furukawa. Wireless client-server computing for personal information services and applications. *ACM Sigmod Record*, 1995.

[7] G.H. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, April 1994.

[8] Y. Hung, P. Sistla, and O. Wolfson. Data replication for mobile computers. In *Proceeding of the ACM SIGMOD Conference on Management of Data*, Minneapolis, Minessota, 1994.

[9] Y. Hung, R. H. Sloan, and O. Wolfson. Divergence caching in client-server architectures. In *Proceedings of the third International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 131–139, Austin, TX, Sept. 1994.

[10] T. Imielinski and B. R. Badrinath. Data management for mobile computing. *SIGMOD RECORD*, 22(1):34–39, March 1993.

[11] T. Imielinski and B. R. Badrinath. Wireless mobile computing : Challenges in data management. *Communication of ACM*, 37(10), 1994.

[12] T. Imielinski and S. Viswanathan. Adaptive wireless information systems. In *proceedings of SIGDBS (Special Interest Group in DataBase Systems) Conference*, Tokyo - Japan, October 1994.

[13] J. Jing, O. Bukhres, A. K. Elmargarmid, and R. Alonso. Bit-sequences: A new cache invalidation method in mobile environments. Technical Report CSD-TR-94-074, Computer Sciences Department, Purdue University, May 1995.

[14] H. Schwetman. *Csim user's guide (version 17).* MCC Corporation, 1992.

[15] D. B. Terry. Caching hints in distributed systems. *IEEE Trans. Software Eng.*, January 1987.

[16] K.-L. Wu, P. S. Yu, and M.-S. Chen. Energy-efficient caching for wireless mobile computing. In *20th International Conference on Data Engineering*, pages 336–345, Feb. 26-March 1 1996.