A Mixed Data Dissemination Strategy for Mobile Computing Systems

Guohong Cao[1], Yiqiong Wu[1], and Bo Li[2]

[1] Department of Computer Science and Engineering,
The Pennsylvania State University
{gcao,ywu}@cse.psu.edu
[2] Department of Computer Science and Engineering,
Xi'an Jiaotong University, Xi'an, China
boblee@summit.com.cn

Abstract. Broadcasting is a very effective technique to disseminate information to a massive number of clients when the data size is small. However, if the data size is large, the broadcast cycle may be long, and hence the access delay becomes a problem. Caching frequently accessed data at the client side can reduce the access latency and improve the bandwidth utilization. However, caching techniques may not perform well when the data are frequently updated. In this paper, we propose to apply different techniques (broadcasting and caching) to deal with different components of the data based on their update frequency. Compared to previous schemes, the proposed solution not only reduces the query latency, but also improves the throughput and the bandwidth utilization.

1 Introduction

With the explosion of Internet techniques and the popularity of mobile terminals (MTs) such as laptops, personal digital assistants, people with battery powered MTs wish to access various kinds of web services over wireless networks at any time any place. However, existing wireless Internet services are limited by the constraints of mobile environments such as narrow bandwidth, asymmetric communication channels, unstable connectivity, and limitations of battery technologies. Thus, mechanisms to efficiently transmit information from the server to the massive number of clients (running on MTs) have received considerable attention [1,4,6].

There are two fundamental models [4] of providing clients with information: *data broadcasting* and *on-demand*. In the broadcasting model, data is periodically broadcasted on a communication channel. Accessing broadcasted data does not require uplink transmission and is "listen only". Since the cost of broadcasting does not depend on the number of users, this method has good scalability and can be used to address the low bandwidth issue in mobile computing systems. In the on-demand model, clients request data on the uplink channel and the server responds by sending the data to the clients.

X.S. Wang, G. Yu, and H. Lu (Eds.): WAIM 2001, LNCS 2118, pp. 408–416, 2001.

Although broadcasting has good scalability and low bandwidth requirements, it has some drawbacks. For example, since a web page may contain a large volume of data (especially in the multimedia era), the data broadcast cycle may be long. Hence, the clients have to wait for a long time before getting the required data. Caching frequently accessed data on the client side is an effective technique to improve performance in a mobile environment. It can reduce the number of uplink message for a particular data item, and the number of downloads of the same data item. However, the disconnection and mobility of the clients make cache consistency a challenging problem. Effective cache invalidation strategies are required to ensure the consistency between the cached data at the clients and original data stored in the server.

Barbara and Imielinski [1] provide a solution which is suitable for mobile environments. In this approach, the server periodically broadcasts an invalidation report (IR) in which the changed data items are indicated. Rather than querying a server directly regarding the validation of cached copies, clients can listen to these invalidation reports over wireless channels. Cao [2] proposed an UIR-based approach to address the long query latency problem of the IR-based approach. In this approach, a small fraction of the essential information (called updated invalidation report (UIR)) related to cache invalidation is replicated several times within an IR interval, and hence the client can answer a query without waiting until the next IR. However, if there is a cache miss, the client still needs to wait for the data to be delivered. In this paper, we propose a scheme to further improve the cache hit ratio. Instead of passively waiting, clients intelligently prefetch the data that are most likely used in the future. Although caching data at the client site can improve performance and conserve battery energy, caching may not be the best solution if the broadcasted data are frequently updated, in which case, a combination of different techniques may be required. We propose to apply different techniques (broadcasting and caching) to deal with different components of the data based on their update frequency. Detailed experiments are provided to evaluate the proposed methodology. Compared to previous schemes, the proposed solution not only reduces the query latency, but also improves the throughput and the bandwidth utilization.

The rest of the paper is organized as follows. Section 2 presents the necessary background. In Section 3, we propose techniques to improve the cache hit ratio, and deal with frequently updated data. Section 4 evaluates the performance of the proposed scheme. Section 5 concludes the paper.

2 Preliminaries

To ensure cache consistency, the server broadcasts invalidation reports (IRs) every L seconds. The IR consists of the current timestamp T_i and a list of tuples (d_x, t_x) such that $t_x > (T_i - w * L)$, where d_x is the data item id, t_x is the most recent update timestamp of d_x, and w is the invalidation broadcast window size. In other words, IR contains the update history of the past w broadcast intervals. Every client, if active, listens to the IRs and invalidates its cache accordingly. To

answer a query, the client listens to the next IR and uses it to decide whether its cache is valid or not. If there is a valid cached copy of the requested data item, the client returns the item immediately. Otherwise, it sends a query request to the server through the uplink.

In order to reduce the query latency, Cao [2] proposed to replicate the IRs m times; that is, the IR is repeated every $(\frac{1}{m})^{th}$ of the IR interval. As a result, a client only needs to wait at most $(\frac{1}{m})^{th}$ of the IR interval before answering a query. Hence, latency can be reduced to $(\frac{1}{m})^{th}$ of the latency in the previous schemes (when query processing time is not considered).

Since the IR contains a large amount of update history information, replicating the complete IR m times may consume a large amount of broadcast bandwidth. In order to save the broadcast bandwidth, after one IR, $m-1$ *updated invalidation reports* (UIRs) are inserted within an IR interval. Each UIR only contains the data items that have been updated after the last IR was broadcasted. In this way, the size of the UIR becomes much smaller compared to that of the IR. As long as the client downloads the most recent IR, it can use the UIR to verify its own cache. The idea of the proposed technique can be further explained by Figure 1. In Figure 1, $T_{i,k}$ represents the time of the k^{th} UIR after the i^{th} IR. When a client receives a query between $T_{i-1,1}$ and $T_{i-1,2}$, it can answer the query at $T_{i-1,2}$ instead of T_i. Thus, to answer a query, the client only needs to wait for the next UIR or IR, whichever arrives earlier. However, if there is a cache miss, the client still needs to fetch data from the server, which increases the query latency. Also, if the data are frequently updated, the latency will be significantly increased. Next, we propose a scheme to improve the cache hit ratio and dealing with the frequently updated data.

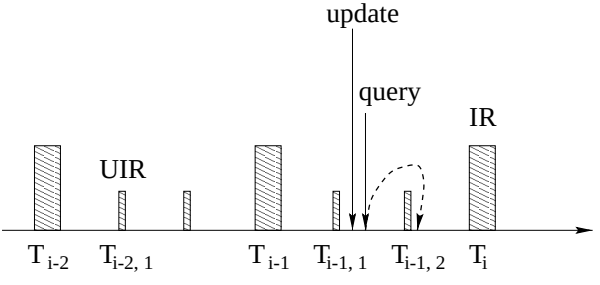

Fig. 1. Reducing the query latency by replicating UIRs

3 A Mixed Data Dissemination Strategy

3.1 Improving the Cache Hit Ratio

To improve the cache hit ratio, clients prefetch data that may be used in the near future. For example, if a client observes that the server is broadcasting a

data item which is an invalid entry of its local cache, it is better to download the data; otherwise, the client may have to send another request to the server, and the server will have to broadcast the data item again in the near future. To save power, clients may only wake up during the IR broadcasting time, and then how to prefetch data becomes an issue. As a solution, after broadcasting the IR, the server first broadcasts the *id* list of the data items whose real data will be broadcasted next, and then broadcasts the data items whose *id*s are in the *id* list. Each client should always listen to the IR if it is not disconnected. At the end of the IR, the client downloads the *id* list and finds out when the interested data will come, and wakes up at that time to download the data. With this approach, power can be saved since clients stay in the doze mode most of the time; bandwidth can be saved since the server may only need to broadcast the updated data once.

3.2 Dealing with Frequently Updated Data

The IR-based approach is very useful in applications where data items do not change frequently, and then clients can cache these data items and use them to serve queries locally. However, if the data are frequently updated, caching may not be helpful. In this case, broadcasting the data on the air may be a good solution. Following this idea, many indexing techniques [4,6] have been proposed to address the tradeoff between query latency and power consumption. In most of the indexing techniques, the index and the real data are both broadcasted. Since some data items may contain a large amount of data (especially in the multimedia era), the clients may have to wait for a long time before getting the required data. In short, the indexing (broadcasting) techniques are good for small data size while the IR-based approach is good for large data size with less update frequency. However, in real life, most applications may not work well with either approach. For example, although the stock price of a company is updated frequently, the company related news such as the company profile, financial news, new product release, and broker coverage, may only be updated several times in a day. Since the stock price is updated too often, the IR-based approach is not suitable. Similarly, broadcasting techniques should not be used to maintain company related news due to large data sizes that need to be updated. We propose to apply multiple techniques to deal with the problem. The central idea is to differentiate the frequently updated data part from others. In other words, a data item can be divided into two *data components*: the hot component and the cold component. The hot component is the data part which is frequently updated, while the cold component is the data part which is not frequently updated. Indexing techniques are used to access those data components that are frequently updated, whereas IR-based techniques are used to access those data components which are not frequently updated. Considering the above example, broadcasting techniques are used to access the stock prices, whereas IR-based techniques are used to access the company news.

To implement the idea, we modify the UIR-based approach so that it can be used to deal with frequent updates. The idea is to broadcast the frequently

updated data components multiple times during an IR interval. This can be done by broadcasting them after each UIR or IR. Since most of the frequently updated data components have small data size, broadcasting them should not add too much overhead. If the client access pattern is known, the hot data components should be broadcasted more often than the cold data components to reduce the average query latency. If multiple channels are available, the server can use one channel to deliver the frequently updated data components using broadcasting techniques, and use another channel to deliver the cold components using the UIR-based techniques. The process of dividing a data item into two components should be mutually agreed by the clients and the server. When a client needs to serve a query, it has to know where to locate the components of the data item. It may have to access one component from the local cache and download the other one from the broadcast channel.

4 Performance Evaluation

In order to evaluate the efficiency of various cache management algorithms, we develop a model which is similar to that employed in [2,3,5]. It consists of a single server that serves multiple clients. The database can only be updated by the server whereas the queries are made on the client side. From the server point of view, the database is divided into two subsets: the *hot* data subset and the *cold* data subset. The hot data subset includes data items from 1 to 100 (out of 2000 items) and the cold data subset includes the remaining data items of the database. From the client point of view, the database is divided into three subsets: the *hot* data subset, the *medium* data subset, and the *cold* data subset. The hot data subset includes 20 randomly (based on the client *id*) chosen items from the first 100 items. The medium data subset includes the remaining 80 items in the first 100 items. The cold data subset includes the remaining data items of the database. Table 1 lists the parameters used in our model.

4.1 The Query Latency

The left graph of Figure 2 shows the query delay as a function of the data size for the broadcast approach under different update arrival time. As can be seen, the query delay is proportional to the data size. When the data size is small, the delay is small. When the data size increases to 2000 bytes, the delay is too high to be tolerable. Thus, the broadcast approach is not suitable for applications which have very large data sizes and have strict delay requirements. As can be seen, the query delay of the broadcast approach is not affected by the mean update arrival time.

The right graph of Figure 2 shows the query delay as a function of the mean update arrival time for the UIR approach under different data item sizes ($S = 2000$ bytes, $S = 200$ bytes, and $S = 20$ bytes). When the mean update arrival time decreases, the data is updated more frequently, and more clients have cache misses. As a result, the query delay increases. As the mean update arrival

Table 1. The Simulation Parameters

Database items D	2000 items
Hot data items D_h	20 items
Medium data items D_m	80 items
Cold data items D_c	(2000-100)=1900 items
Number of clients n	200
Cache size c	20 to 200 items
Broadcast interval L	20 seconds
The UIR replicate times m	5
Broadcast window size w	10 intervals
Mean hot component update arrival time T_h	0.001s to 10000s
Mean cold component update arrival time T_c	0.1s to 10000s
Mean info. query generate time T_q	10s to 300s
Hot data update prob. p_u	0.33
Hot data access prob. q_h	0.75
Medium data access prob. q_m	0.10
Cold data access prob. q_c	0.15
Hot component data size S_h	16 bits
Cold component data size S_c	10 bytes to 2048 bytes
The timestamp size S_{tmp}	32 bits
The id size S_{id}	16 bits
Broadcast bandwidth R	20000 bits/s

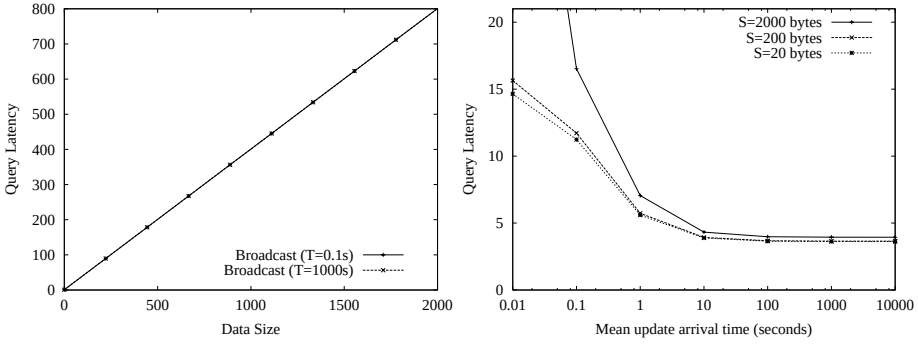

Fig. 2. A comparison of the query latency. The left figure shows the query latency as a function of data size for the broadcast approach. The right figure shows the query latency as a function of the mean update arrival time for the UIR approach ($T_q = 100s, c = 200, n = 100$).

time becomes very small, many clients have cache misses and their requests form a queue at the server. Since it takes a longer time to send a large data item than a small data item, clients may need to wait for a longer time if the data size is large. For example, the query delay of $S = 2000$ is much higher than that of $S = 20$ as the mean update arrival time drops below $1s$.

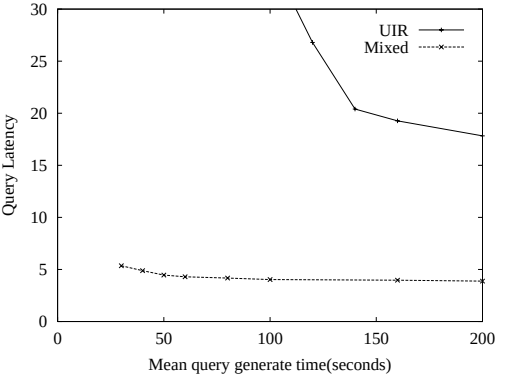

Fig. 3. A comparison of the query delay ($T_c = 100s, T_h = 0.01s, c = 200, n = 100$)

Each client generates queries according to the mean query generate time. The generated queries are served one by one. If the queried data is at local cache, the client can serve the query locally; otherwise, the client has to request the data from the server. If the client cannot process the generated query due to waiting for the server reply, it queues the generated queries. Since the broadcast bandwidth is fixed, the server can only transmit a limited amount of data during one IR interval, and then it can only serve a maximum number (α) of queries during one IR interval. If the server receives more than α queries during one IR interval, some queries are delayed to the next IR interval. If the server receives more than α queries during each IR interval, many queries may not be served, and the query delay may be out of bound. Due to the difference of the data size in different approach, the value of α varies. For example, with $T_h = 0.01s$, the query delay of $S = 200$ is still around $15s$, but the the query delay of $S = 2000$ becomes infinitely high. Note that when $T_h = 0.01s$, broadcasting the IR and UIR occupies a large amount of bandwidth since many data items are updated and their *id*s must be added to the IR and UIR.

Figure 3 compares the query delay of the UIR approach and the mixed approach. As can be seen, the mixed approach outperforms the UIR approach in terms of query delay. In the mixed approach, most of the queries can be served after an UIR is broadcasted, and then the query delay is very low. The query delay of the UIR approach becomes infinitely high when T_q drops below $100s$. This is due to the fact that the cache hit ratio of the UIR approach is much smaller than the mixed approach. Since the UIR approach does not differentiate between the hot component and the cold component, the mean update arrival time equals the hot component update arrival time $T_h = 0.01s$, and then its cache hit ratio is near 0. In the mixed approach, the mean update arrival time is $T_c = 100s$, and then their cache hit ratio is pretty high.

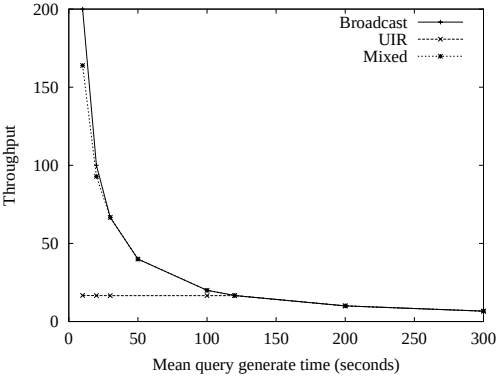

Fig. 4. The number of queries served per IR interval ($T_c = 100s, T_m = 0.01, c = 200, n = 100$)

4.2 The Throughput

As explained in the last subsection, with limited broadcast bandwidth, the server can only serve a maximum number (α) of client requests during one IR interval. However, the throughput (the number of queries served per IR interval) may be larger than α since some of the queries can be served by accessing the local cache. Since the mixed approach has much higher cache hit ratio than the UIR approach, the throughput of the mixed approach is much higher than the UIR approach. As shown in Figure 4, the broadcast approach has the highest throughput, while the UIR approach has the lowest throughput. The mixed approach has the same throughput as the broadcast approach when $T_q > 40s$. When the mean query generate time drops below 40s, the throughput of the mixed approach becomes lower than the broadcast approach.

5 Conclusions

We proposed a mixed approach to deal with the limitations of the broadcasting techniques and cache techniques. In our approach, based on the data update frequency, each data item is divided into two components: the hot component and the cold component. The UIR-based approach is used to deal with the cold component and broadcasting techniques are used to deal with the hot component. In this way, the proposed mixed data dissemination approach can be extended to many different application environments. Simulation results showed that the proposed solution can not only reduce the query latency, but also improve the throughput and the bandwidth utilization.

References

1. D. Barbara and T. Imielinski: Sleepers and Workaholics: Caching Strategies for Mobile Environments. ACM SIGMOD. (1994) 1–12
2. Cao, G.: A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments. ACM Int'l Conf. on Mobile Computing and Networking (MobiCom). (2000) 200–209
3. Hu, Q. and Lee, D.: Cache Algorithms based on Adaptive Invalidation Reports for Mobile Environments. Cluster Computing. (1998) 39–48
4. Imielinski, T., Viswanathan, S., Badrinath, B.: Data on Air: Organization and Access. IEEE Transactions on Knowledge and Data Engineering, vol. 9, no. 3, (1997) 353–372
5. Jing, J., Elmagarmid, A., Helal, A., Alonso, R.: Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments. Mobile Networks and Applications. (1997) 115–127
6. W. Lee, Q. Hu, and D. Lee: Lee, W., Hu, Q., Lee, D.: A Study on Channel Allocation for Data Dissemination in Mobile Computing Environments. ACM/Baltzer Mobile Networks and Applications. (1999) 117–129