A Strategy for Efficient Access of Multiple Data Items in Mobile Environments

ABSTRACT

Many approaches have been proposed to schedule data items for broadcasting in a mobile environment. However, the issues of accessing multiple data items on the broadcast channel are less discussed. In this paper, an efficient strategy for accessing multiple data items in mobile environments is proposed. Two issues are investigated in this paper, that is, deciding the content of the broadcast channel based on the queries from the clients, and scheduling the data items to be broadcast. Experiments are performed to compare our proposed methods for deciding the content and to show that our data scheduling methods can reduce the average access time without considerable increase on processing time.

KEYWORDS: mobile databases, information dissemination, data broadcasting, data scheduling

1. INTRODUCTION

Because of the rapid development and distribution of hardware and software techniques for mobile computing, more and more people use mobile devices to access data. Many researchers investigated the database issues of mobile computing [6][10], which include data dissemination. Because of the *asymmetry* in wireless communication (i.e., the bandwidth from server to client is much higher than that from client to server), more and more researchers focus on data broadcasting in the mobile computing environment.

The broadcast bandwidth is a very precious resource in the mobile computing environment. Hence efficient data allocation in the broadcast channel must be performed to speed up the data access for users. The *access time* is frequently used to evaluate the performance of data allocation algorithms. Access time is the period of time elapsed from the moment a mobile client makes a query to the moment when the mobile client receives all the requested data items. Acharya et al. [1][2][3][4] propose the concept of *broadcast disk.* A single broadcast channel is used to broadcast data items in different frequencies according to their relative access rates. That is, popular data items are more frequently broadcast than unpopular ones. Besides data allocation, issues on cache management, prefetching, updating, and scalability in such broadcast environments are also addressed. Leong and Si [12] suggest some data broadcasting strategies based on data replication and

partition techniques to distribute the transmitted data items over multiple broadcast channels. However, both of these approaches do not consider multiple data item queries.

In the mobile computing environment, there are three kinds of data transmission techniques: pure-push-based, pure-pull-based, and a combination of these two techniques. The first technique is that the server periodically transmits data items and the clients just passively listen to the broadcast channel to retrieve their desired data items. The major advantage of this technique is that all of the mobile clients can access data items on the broadcast channel at the same time without increasing the server workload. That is, scalable numbers of users can be supported without any performance degradation. However, the limitation of the pure-push-based technique is that mobile clients can only sequentially access data items of interest appearing on the broadcast channel. On the other hand, data items required by any of the mobile clients must be transmitted on the broadcast channel. These two disadvantages extend the average data access time. The pure-pull-based technique is that mobile clients have to explicitly send requests to the server. The server then responds to each request individually via an on-demand channel. Such technique cannot be scaled up to any number of mobile clients and the access latency depends on the server workload. The third technique is a combination of the previous two, which benefits both the server and clients. Acharya et al. [5] integrate their previous work which is a pure-push-based data dissemination scheme with the pure-pull-based approach. They use a *backchannel* to allow mobile clients to send explicit messages to the server, if the requested data items are not on the broadcast channel. In Stathatos et al. [14], a hybrid approach is proposed by using pure-push-based technique for hot data and pure-pull-based technique for cold data. In Prabhakara et al. [13], broadcast strategies and techniques supporting different mobile clients' access behaviors in multiple channel mobile environments are proposed. However, these approaches only consider the situation that a query can access only one data item.

There are many applications in which mobile clients concurrently access multiple data items. For example, a mobile client wants to know the stock prices of Cisco, Microsoft, and IBM at the same time. In previous research, this mobile client needs to issue three queries to acquire these three stock prices individually. Besides, the server schedules data items without considering the relationship between them, which extends the access time to process the clients' requests. Therefore, the access of multiple data items in a query is an important issue in mobile computing environments. In Chung and Kim [7], a broadcast schedule method called *Query Expansion Method* (QEM) to minimize the average access time is proposed. The main idea of this scheduling method is described in detail in Section

3. This method can be modified to get a better scheduling with a lower average access time. In this paper, we propose a broadcast content selection scheme. Moreover, two scheduling strategies to overcome the shortage of Chung and Kim's work are also presented.

The remaining of this paper is organized as follows. In Section 2, the formulation of the scheduling problem is introduced. We describe the problem of deciding the content of the broadcast channel and propose three greedy methods to solve it. In Section 3, the issue of data scheduling is introduced and the basic idea of QEM is described. Then our modified algorithms based on QEM are presented. We show the results of the experiments, which evaluate the three greedy methods and justify the improvement of the modified-QEM algorithms in Section 4. Finally, the conclusion and future work are presented in Section 5.

2. THE DATA BROADCASTING PROBLEM

In this section we formulate the data broadcasting problem first. Then we present three greedy methods to decide the content of the broadcast channel.

2.1. Problem Formulation

Given a set of queries $\mathbf{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ and a set of data items $\mathbf{D} = \{d_1, d_2, \ldots, d_m\}$. Each query Q_i accesses a set of data items called *Query Data Set*, represented by $QDS(Q_i)$, where $QDS(Q_i) \subset \mathbf{D}$. The frequency of query Q_i is denoted by $freq(Q_i)$ and is named as *query frequency* in the rest of this paper. We denote a broadcast schedule on a single broadcast channel by $\sigma = \{d_i, d_j, \ldots, d_k\}$, which is a permutation of **D** (or a permutation of a subset of **D**). There are two issues to be investigated. First, the server has to choose queries and broadcast their associated data items in order to serve a maximal number of clients accessing data items from the broadcast channel. Therefore, only some queries of **Q** are chosen for broadcast. Second, the server needs to schedule the data items of these chosen queries on the broadcast channel to minimize the average access time. The measure *Query Distance* (QD) defined in [7] is used to show the coherence degree of a query's QDS in a schedule which also represents the time needed to access the data items for this query. The definition of QD is as follows:

Definition of QD: Suppose $QDS(Q_i)$ is $\{d_1,d_2,\ldots,d_n\}$, and δ^i is the interval between d_i and d_{i+1} in schedule σ. Then the QD of Q_i on σ is defined as: $QD(Q_i,\sigma) = BC - MAX(\delta^i)$ where BC is the length of a broadcast cycle.

For example, we assume a schedule $\sigma = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}\}$ where each data item is of equal size. There is a query Q_t and its QDS is $\{d_2, d_4, d_5, d_8\}$. Then the QD of Q_t is 10-3=7 in this schedule. The summation of $QD(Q_i)*freq(Q_i)$ of all queries is

called *Total Query Distance* (TQD), which can represent the average access time under the corresponding schedule [8]. In order to minimize the average access time, we have to find a schedule that minimizes TQD.

2.2. Deciding the Content of the Broadcast Channel

We propose three greedy methods to decide queries from the Query_List to form the content of the broadcast. We call them *Query Selection Methods* throughout the rest of the paper.

The first method is intuitive in which the server only broadcasts data items of queries with higher frequencies. That is, the server selects the queries to broadcast according to the rank of frequencies. If two queries have the same frequency then it chooses the query with the lower number of data items. Because queries with lower numbers of data items will occupy less space. The remaining BC can be reserved for data items of other queries. The detailed description for Method 1 is shown in Figure 1.

Method 1:

Step 1: Sort queries by their frequency from high to low in the Query_List

Step 2: Current node = Head of the Query_List

```
while(not end of Query_List)
{
    If (no. of new data items in current node + no. of data items in BC ≤ BC)
    {
       allocate new data items in current node to the broadcast channel
       no. of data items in BC += no. of new data items in current node
    }
    current node = next node from the Query_List
}
```

Figure 1: Method 1 for selecting the content of the broadcast channel

We give an example to show how this method chooses queries for broadcasting. We assume that the capacity is 5 data items and we have six candidate queries Q_1~Q_6 as shown in Figure 2.

QDS(Q_1)={d_1, d_2}, frequency (Q_1)=120
QDS(Q_2)={d_2, d_5, d_6}, frequency (Q_2)=105
QDS(Q_3)={d_3, d_4}, frequency (Q_3)=90
QDS(Q_4)={d_8, d_9, d_{10}, d_{11}}, frequency (Q_4)=124
QDS(Q_5)={d_2, d_4, d_7}, frequency (Q_5)=66
QDS(Q_6)={d_{12}}, frequency (Q_6)=80

Figure 2: Queries Q_1~Q_6

According to Method 1, the order of these six queries will be Q_4{d_8, d_9, d_{10}, d_{11}; 124}, Q_1{d_1, d_2; 120}, Q_2{d_2, d_5, d_6; 105}, Q_3{d_3, d_4; 90}, Q_6{d_{12};80}, Q_5{d_2, d_4, d_7; 66} (where d_i's are data items and the last number within brackets is the query frequency.). Since the capacity limitation is 5 data items, we can only broadcast data items of Q_4 and Q_6 on the channel. Therefore, 124+80=204 requests are served through the broadcast channel.

The first method only considers the query frequency for selecting queries. According to Method 1, higher frequency queries have higher precedence. However, a high frequency query with a large no. of acquired data will be selected and a large space of the broadcast channel will be used. As a consequence, fewer queries can be selected for broadcasting. Method 2 in Figure 3 is based on the concept of the knapsack algorithm [9] to overcome this disadvantage. The main idea of Method 2 is to sort queries by the associated ratios $\frac{f_i}{n_i}$ in decreasing order where f_i is the frequency of Q_i and n_i is the number of data items of Q_i. For the last example we have the following order for the six queries: Q_6{d_{12};80}, Q_1{d_1, d_2; 60}, Q_3{d_3, d_4; 45}, Q_2{d_2, d_5, d_6; 35}, Q_4{d_8, d_9, d_{10}, d_{11}; 31}, Q_5{d_2, d_4, d_7; 22} where the numbers inside the brackets are the ratio value of the corresponding query. In this case, we can select Q_6, Q_1, and Q_3 for broadcasting which serves 80+120+90=290 requests.

Method 2:

Step 1: Calculate the ratio $\frac{f_i}{n_i}$ of each query in the Query_List

Step 2: Sort queries by the associated ratios in decreasing order

Step 3: Current node = Head of the Query_List

```
while(not end of Query_List)
{
    If (no. of new data items in current node + no. of data items in BC ≤ BC)
    {
        allocate new data items in current node to the broadcast channel
        no. of data items in BC += no. of new data items in current node
    }
    current node = next node from the Query_List
}
```

Figure 3: Method 2 for selecting the content of the broadcast channel

Although the ratio of the frequency to the number of data items is considered in Method 2, the overlapping data items, i.e., a data item may be contained in more than one query, is not considered. In Method 3, the query whose number of data items not exceeding the number of data items in BC and has a maximal $\frac{f_i}{n_i}$ is first selected to put into the broadcast channel. Then, the remaining queries are evaluated by $\frac{f_i}{m_i}$ where m_i is the number of data items of Q_i, which have not been allocated in the broadcast channel. Notice that when m_i equals to zero, it means all the data items of Q_i have been allocated on the broadcast channel. The data items of Q_i with a maximal $\frac{f_i}{m_i}$, which are not yet allocated and whose size can fit in the remaining broadcast cycle will be allocated on the broadcast channel. For the remaining queries, m_i may need to be updated and $\frac{f_i}{m_i}$ recomputed. We repeat this process until no query can be further processed. Using the above example, Q_6 (80) will be first put into the broadcast channel, then Q_1 (60) will be selected, finally, Q_2 (52.5) will be put into the broadcast channel. In the example, the server will broadcast data items of Q_6, Q_1, and Q_2 which serve 80+120+105=305 requests. The description for Method 3 is shown in Figure 4.

Method 3:

Step 1: remove the query whose no. of data items not exceeding the no. of data items in BC and has a maximal $\frac{f_i}{n_i}$ from Query_List and put it into the broadcast channel

no. of data items in BC = no. of data items in the selected query

updated=1

Step 2: While (updated ==1)

{

remove the queries whose m_i equals zero from the Query_List

evaluate $\frac{f_i}{m_i}$ of the queries in the Query_List

if there exists the queries whose no. of new data items + no. of data items in BC ≤ BC

{

remove the one with maximal ratio from the Query_List and put it into the broadcast channel

no. of data items in BC+ = no.of new data items in the selected query

}

else

updated = 0

}

Figure 4: Method 3 for selecting the content of the broadcast channel

3. DATA SCHEDULING

In this section, we discuss the issue of data scheduling. A scheduling method called Query Expansion Method (QEM) proposed in [7] is introduced. Then, we propose two new data scheduling methods to enhance QEM.

3.1. Data Scheduling Issues

Assume the server periodically broadcasts a set of data items $\{d_1, d_2, d_3, d_4, d_5, d_6\}$ and a client wants to retrieve d_2 and d_5. In Figure 5(a) the client has to wait for the next broadcast cycle to access d_2 because d_2 has passed when the client tunes in. However, if the data items are scheduled as shown in Figure 5(b), then the client can access d_2 and d_5 in the same broadcast cycle. In that case the access time is lower than for previous schedule. Therefore, a good broadcast schedule should place the data items accessed in a query close to each other to reduce access time.

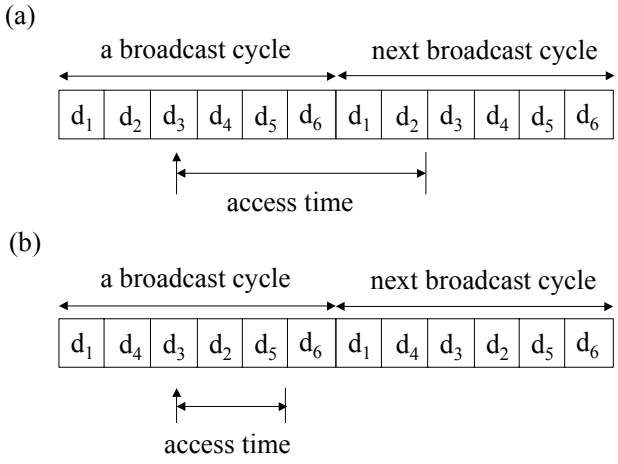

Figure 5: An example for access times in different schedules

3.2. Query Expansion Method

The QEM proposed in [7] is a greedy method. It expands the QDS of each query in a greedy manner after sorting all queries according to their corresponding access frequencies. This algorithm has the following policies.

Policy 1: Higher-frequency queries have higher precedence for expansion.

Policy 2: During expanding the QDS of a query, the QD of the queries which had been previously expanded, remain unchanged.

Policy 3: When expanding query q_i into the current schedule, the proposed method always minimizes the QD of q_i as much as possible.

Now we show an example to explain how this algorithm constructs the broadcast schedule. In the example, there are three queries that clients submit to the server and 7 data items to be transmitted on the broadcast channel. The access frequency of each query is freq(Q_1)=100, freq(Q_2)=80, and freq(Q_3)=50. All data items are assumed to be of equal size and the QDS of each query is depicted in Figure 6.

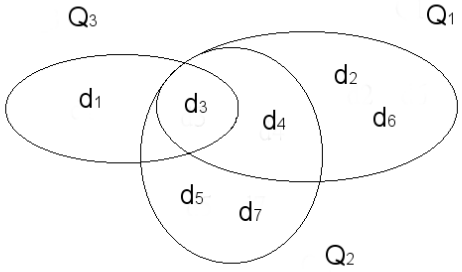

Figure 6: QDS of queries Q_1~Q_3

Initially the schedule σ is empty. According to Policy 1, QEM finds the query with the highest access frequency (query Q_1) to expand its data items. Then the current broadcast schedule σ is formed as follow:

$$\sigma = [\ d_2, d_3, d_4, d_6\ ]$$

We use the symbols "[" and "]" to represent the schedule in which data items are free to interchange without affecting the QD.

The next query to be scheduled is query Q_2, whose QDS is { d_3, d_4, d_5, d_7 }. Since d_3 and d_4 are also data items of query Q_1, which had been expanded in the current schedule σ. Therefore we have two choices to expand QDS of query Q_2 as follows:

$$\sigma_{\text{Right-append}} = [\ d_2, d_6\ ]\ [\ d_3, d_4\ ]\ [\ d_5, d_7\ ]$$

$$\sigma_{\text{Left-append}} = [\ d_5, d_7\ ]\ [\ d_3, d_4\ ]\ [\ d_2, d_6\ ]$$

The first schedule is the result of appending Q_2 at the rightmost position of the previous schedule, whereas the second one is the result of appending Q_2 at the leftmost position. Both choices minimize the QD of Q_2 (Policy 3) and preserve the QD of Q_1 (Policy 2). Since data items inside "[" and "]" are interchangeable, we have 2*2*2=8 possible ways to arrange these data items in the above two schedules and all of them have the same TQD. In this example, we choose $\sigma_{\text{Left-append}}$ to continue the expanding process.

Finally we consider QDS(Q_3) = { d_1, d_3 }. For these two data items, only d_1 does not appear in the previous two queries. Therefore we only have to append data item d_1 into the schedule at the rightmost or leftmost position. While appending data item d_1, we need to move data items d_3 and d_5 as close to d_1 as possible (if they are interchangeable inside "[" and "]") to minimize QD of Q_3. This case is depicted as follows.

$$\sigma_{\text{Right-append}} = [\ d_5, d_7\ ]\ [\ d_4, d_3\ ]\ [\ d_2, d_6\ ]\ [\ d_1\ ]$$

$$\sigma_{\text{Left-append}} = [\ d_1\ ]\ [\ d_5, d_7\ ]\ [\ d_3, d_4\ ]\ [\ d_2, d_6\ ]$$

Since both of the above schedules have the same TQD, we choose the schedule with left-append as our final schedule. Then the TQD of this broadcast schedule is

$$100*(7-3)+80*(7-3)+50*(7-3)=920.$$

3.3. Modified Query Expansion Methods

In this section, we will present two broadcast data scheduling methods based on QEM to improve its performance.

In Method 1, the similar expanding policies in QEM are applied on the data items from high-frequency queries to low-frequency queries. The main differences are on Policy

2 and Policy 3 in QEM. In QEM, the QD of the previously expanded queries can not be changed. However, if we loosen this restriction, TQD may become smaller. The change of the QD of the previously expanded queries means that the data items in the previous schedule can be moved. These data items are the overlapping QDS in previous schedule and currently being expanded query. We move the data items to the rightmost position or leftmost position of previous schedule such that they can be adjacent to the new data items of the currently expanding query. However, some QD's of the previously expanded queries may become larger after the moving operation, and some may become smaller. The moving operation is performed only if the TQD after the moving operation is smaller than QEM, which does not apply the moving operation.

In QEM a *segment* is defined, which contains some queries with overlapping data items. For example, σ_i has one segment. However σ_j has two segments that are connected by a symbol "$\oplus$". Because no query in σ_j accessing data items d_6, d_7, d_{10} will also access any one of the data items d_1, d_3, d_4, d_8, d_9, there are two segments in this schedule:

$$\sigma_i \quad [\, d_1, d_8 \,] [\, d_3, d_4, d_9 \,] [\, d_6, d_7, d_{10} \,]$$

$$\sigma_j \quad [\, d_1, d_8 \,] [\, d_3, d_4, d_9 \,] \oplus [\, d_6, d_7, d_{10} \,]$$

We focus on the situation that the currently expanded query intersects only one segment as shown in Figure 7 and other cases are expanded as QEM. Data items in the previous schedule are presented by (a), the shadow portion of (a) are the overlapping data items between (a) and the currently expanded query. Figure 7 (b) are data items of currently expanded query that do not appear in (a). If we apply QEM to schedule this query we get schedule (c). However, we can move the shadow portion of (a) to concentrate data items of the currently expanded query like (d) when TQD of (d) is less than TQD of (c).

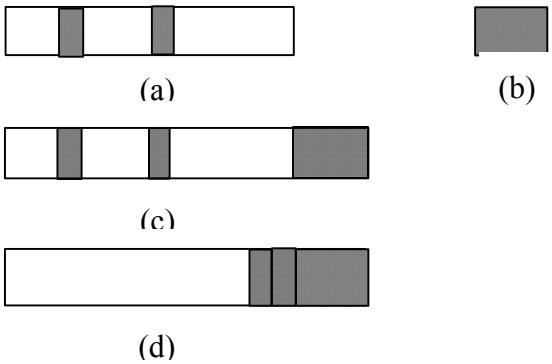

Figure 7: An example of a query intersecting one segment

If we allow data item d_3 in the previous example to be moved in the schedule and concentrate d_3 with the new data item d_1 of query Q_3 together, then we will have the following schedule:

$$\sigma_{Change\_QD} = [\ d_5, d_7\ ]\ [\ d_4\ ]\ [\ d_2, d_6\ ]\ [\ d_3\ ]\ [\ d_1\ ]$$

In the above schedule the QD of query Q_1 is the same as before. However, the QD of query Q_2 increases from 7-3=4 to 7-2=5 and the QD of query Q_3 decreases from 7-3=4 to 7-5=2, which benefits users accessing Q_3. Then TQD equals 100*(7-3)+80*5+50*2=900. This schedule will lower the TQD of the previous one by 20. That is, the average access time of our algorithm is less than that of QEM.

In this method, additional calculation of the TQD is required to decide whether performing the moving operation or not will prolong the processing time. We can set a threshold to filter out some queries where we need not perform the moving operation when we expand them. The probability that the moving operation will reduce the TQD is proportional to the frequency of the currently expanded query. When we set the threshold to one, we have to check TQD when expanding each query to determine whether we should apply the moving operation or not. If we set the threshold to zero, the moving operation will never be performed and the scheduling result is the same as QEM. If the threshold is set to 0.5, only queries with first 50% high frequency will need to check TQD to decide whether should be performed the moving operation.

In our first scheduling method, sometimes the moving operation will not benefit the TQD after scheduling all queries. The reason is that we only consider the QDs of the previously expanded queries and the currently expanded query Q_c. Some queries with lower frequencies than Q_c may also request the (or part of) data items, which are the overlapping data items of the previous schedule and the currently expanded query. The benefit of the moving operation when expanding QDS(Q_c) may not be larger than the loss of it after scheduling these queries. Therefore, we propose the second scheduling method in which an additional inequality needs to be checked before applying the moving operation. If the inequality is true then the moving operation is applied; otherwise not. The inequality is shown as follows:

$$freq(Q_c) + \sum_{\substack{QDS(Q_i)\cap Overlap(Q_c)\neq\phi \ \ \& \\ QDS(Q_i)\cap(QDS(\sigma_{Q_c})-Overlap(Q_c))=\phi}} freq(Q_i) \ > \sum_{\substack{QDS(Q_j)\cap Overlap(Q_c)\neq\phi \ \ \& \\ QDS(Q_j)\cap(QDS(\sigma_{Q_c})-Overlap(Q_c))\neq\phi}} freq(Q_j)$$

Where Q_i (or Q_j) is the query with smaller frequency than the currently expanded query Q_c, QDS(σ_{Qc}) is the set of data items of the previous schedule and Overlap(Q_c) is the set of data items of the intersection of QDS(Q_c) and QDS(σ_{Qc}). The left side of the inequality is the summation of the frequencies of the remaining queries which will benefit by applying the moving operation and the right side is the summation of the frequencies of the remaining queries. For instance, assume two queries Q_4 and Q_5 with access frequencies freq(Q_4)=30

and freq(Q_5)=25 are further considered in the previous example. The QDS of each of them is depicted in Figure 8.

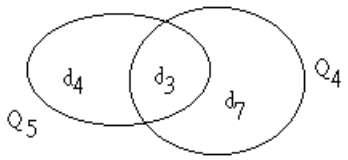

Figure 8: QDS of queries Q4 and Q5

Since both of them access the data item d_3 which is one of the data items in Q_3 and d_4, d_7 already exist in the previous schedule, freq(Q_3)=50 < freq(Q_4)+freq(Q_5)=30+25=55, we need not apply the moving operation when we expand QDS(Q_3). In this case, TQD of applying the moving operation equals 900+30*(7-3)+25*(7-3)=1120, but TQD of not applying the moving operation equals 920+30*(7-5)+25*(7-5)=1030.

4. PERFORMANCE EVALUATION (增加RELATIVE WORK)

4.1. Simulation Models

The performance evaluation consists of two parts, one for the query selection methods and the other for the scheduling methods. The simulation is run on pentium III 700 processor with 512k cache and 128 M memory. To evaluate the performance of the scheduling methods, a set of simulations is performed by generating different query data sets. We compare the average access times for the broadcast program generated by the two proposed methods and QEM with the lower bound on the average access time for the optimal broadcast program of the queries. The lower bound on the average access time for the optimal broadcast program of the queries is derived in the following Lemma.

Lemma ? : Given a set of queries Q and let the summation of the request frequency of the query in Q is 1. Assume the number of data items acquired by a query is at least 2. The average access time for retrieving the QDS(Q_i) for all the Q_i in Q is no less than $\frac{2}{BC} + BC - 1$, where BC is the number of data items on the broadcast channel.

Proof: As discuss in section 2, to minimize the average access time of Q_j is to allocate the QDS(Q_j) adjacently on the broadcast channel. Therefore, the minimum average

access time for retrieving the QDS(Q_j), denoted as $MAAT_{Q_j}$ can be computed as follows:

$MAAT_{Q_j} = \frac{1}{BC} \times |QDS(Q_j)| + \frac{BC-1}{BC} \times BC$, where |QDS($Q_j$)| denotes the number of data items acquired by Q_j.

Therefore, the average access time for retrieving the QDS(Q_i) for all the Q_i in Q, denoted as AAT_Q is no less than $\sum_{Q_j \in Q} freq(Q_j) \times MAAT_{Q_j}$, i.e.

$$
\begin{aligned}
AAT_Q \geq & \sum_{Q_j \in Q} freq(Q_j) \times (\frac{1}{BC} \times |QDS(Q_j)| + \frac{BC-1}{BC} \times BC) \\
= & \frac{1}{BC} \times \sum_{Q_j \in Q} freq(Q_j) \times |QDS(Q_j)| + \sum_{Q_j \in Q} freq(Q_j) \times (BC-1) \\
= & \frac{1}{BC} \times \sum_{Q_j \in Q} freq(Q_j) \times |QDS(Q_j)| + (BC-1) \times \sum_{Q_j \in Q} freq(Q_j) \\
= & \frac{1}{BC} \times \sum_{Q_j \in Q} freq(Q_j) \times |QDS(Q_j)| + (BC-1) \\
\geq & \frac{1}{BC} \times \sum_{Q_j \in Q} freq(Q_j) \times 2 + (BC-1)
\end{aligned}
$$

(The number of data items acquired by Q_j is at least 2. Therefore, |QDS(Q_j)| ≥2.)

And finally, we get $AAT_Q \geq \frac{2}{BC} + BC - 1$.

4.2. Experiment Setup

In the simulation, there are 1000 distinct data items and among the 1000 distinct data items, we select 100 distinct data items to form the *overlapping data set*. The overlapping data set is used to model the overlap data items among the queries. Each query has 2~20 data items with request frequency between 0~1 Moreover, the summation of request frequency is equal to 1. The following parameter is used to generate different broadcast data sets.

PARAMETERS

- *Selectivity*: The ratio of the number of data items in the query contained in the overlapping data set to the number of data items in the query, which is the degree of the data items acquired by a query overlap to the data items acquired by other queries.

- N: The number of queries.
- θ: The parameter of Zipf distribution.

Parameters	Default value	Ranges
Selectivity	0.3	0.1~0.6
Number of queries (N)	300	100 ~ 600
Zip parameter (θ)	0.5	0 ~ 0.99

Table 1: Parameter Settings

The parameter settings for our experiments are listed in Table 1. The access frequencies of queries are based on the Zipf distribution [GSE94]. In the Zipf distribution, the access frequencies for the queries follow the 80/20 rule that 80 percent clients are usually interested in 20 percent data items.

4.3. Experiment Results

Comparison of Query Selection Methods

The experimental results are depicted in Figure 9. The comparison is based on the summation of the access frequencies of the queries served through the broadcast channel under different Broadcast Capacities. The results show that Method 3 serves more requests than Method 2, which on the other hand serve more requests than Method 1.

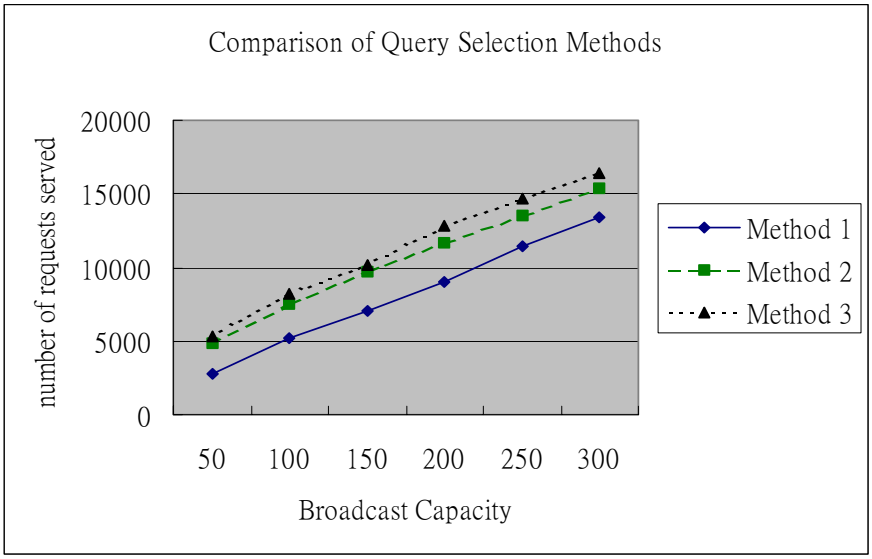

Figure 9: Comparison of Query Selection Methods for 100 queries

Figure 10: Comparison of Query Selection Methods for 150 queries

4.4. Comparison of QEM and Modified-QEM

In this experiment, the performance is evaluated by averaging *TQD Reduction* of 10 runs for each number of queries. TQD Reduction is the difference of TQD between QEM and our two data scheduling methods and is defined as follows:

TQD Reduction = TQD(QEM) –TQD(Modified-QEM)

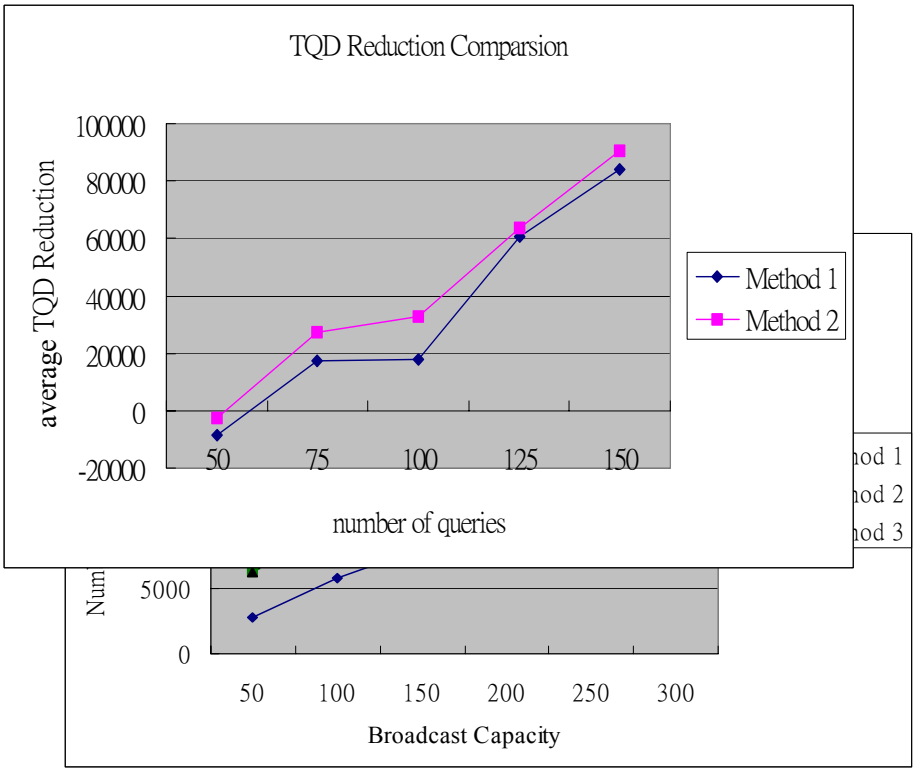

Figure 11: Comparison of QEM and Modified-QEM

As shown in Figure 11, TQDs of Method 1 and Method 2 are larger than that of QEM when the number of queries is 50. Besides, there are also some cases in the 10 runs that our Modified-QEM has a larger TQD than QEM when the number of queries varies from 75 to 150. The reason is that our Modified-QEM only considers the interaction between some of the affected queries during the expansion of data items and the global effect of the moving operation is not considered. Therefore, there are some cases that QEM will outperform our Modified-QEM. In the 10 runs, the benefit of better cases will be larger than the loss of worse ones. On the other hand, the experimental result shows that TQD Reduction of Method 2 is larger than that of Method 1 because an additional inequality is considered to determine whether applying the moving operation or not in Method 2. Hence Method 2 is better than Method 1.

4.5. Effect of the Threshold

As mentioned in section 3.3, the server will take much time to compute the TQD. Hence, a threshold p is set to let the first (p*no. of queries) high frequency queries check TQD to decide whether applying the moving operation or not. We run this experiment on 100 queries. Figure 12 shows that when the threshold p becomes large, it takes much time to compute TQD. Moreover, the processing time is small when p<=0.6 and it exceeds 10 seconds when p=0.8.

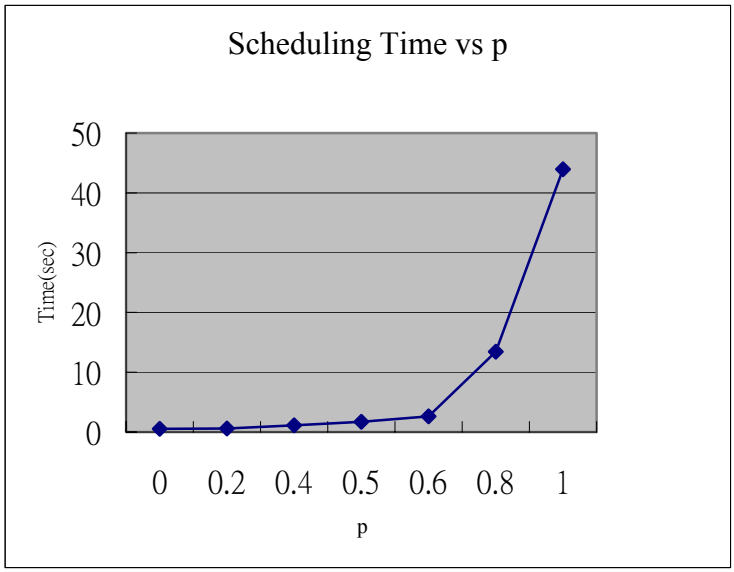

Figure 12: Effect of the threshold on processing time

In Figure 13, we compare TQD under different threshold values. It is observed that the value of TQD for p=0.5 (or 0.6) are similar to the value of TQD for p=1. According to Figure 12 and 13, we can set the threshold p to be 0.5, so that we can get a similar performance as p=1 and the scheduling time will be low.

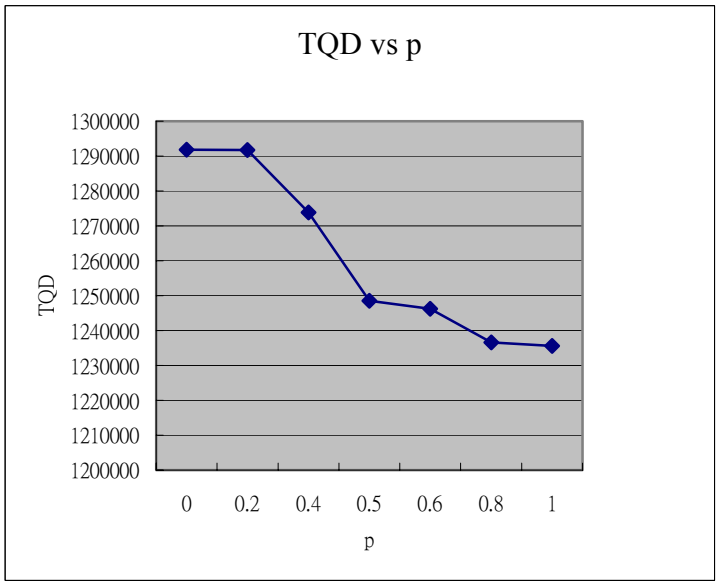

Figure 13: Effect of the threshold on TQD

5. CONCLUSION

In this paper, an efficient strategy for answering multiple data item queries in mobile environments is proposed. Two issues are investigated in this paper. We first propose three Query Selection Methods to select new queries for selecting the content of the broadcast channel according to queries from the clients. The first method is based on the frequency of the query. This method does not consider the number of data items in a query. The second method takes both the frequency and the number of data items into consideration. However, this method does not consider that a data item may be contained in several queries. Hence we propose the third method to overcome this problem. Our experiments show that Method 3 can serve more requests than Method 2, which is better than Method 1. Second, we propose two methods based on QEM to schedule the data items of the chosen queries on the broadcast channel. The experiment results show that our data scheduling methods can reduce the average access time without considerable increase of the scheduling time.

In the future, we will extend this work on the environment with multiple broadcast channels. Besides, we will involve index techniques to reduce the tuning time.

REFERENCE:

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proc. ACM SIGMOD Conference*, pages 199~210, 1995.

[2] S. Acharya, R. Alonso and S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", *IEEE Personal Communications*, 2(6), 1995.

[3] S. Acharya, R. Alonso and S. Zdonik, "Prefetching from a Broadcast Disk", *Proc. IEEE International Conference on Data Engineering*, pages 276~285, 1996.

[4] S. Acharya, R. Alonso and S. Zdonik, "Disseminating Updates on Broadcast Disks", *Proc. VLDB Conference*, pages 354~365, 1996.

[5] S. Acharya, M. Franklin and S. Zdonik, "Balancing Push and Pull for Data Broadcast", *Proc. ACM SIGMOD Conference*, pages 183~194, 1997.

[6] D. Barbara, "Mobile Computing and Databases – A Survey", *IEEE Transactions on Knowledge and Data Engineering*, 11(1), pages 108~117, 1999.

[7] Y. D. Chung and M. H. Kim, "QEM: A Scheduling Method for Wireless Broadcast Data", *Proc. International Conference on Database Systems for Advanced Applications proceedings*, pages 135~142, 1999.

[8] Y. D. Chung and M. H. Kim, “On Scheduling Wireless Broadcast Data”, Technical Report CS-TR-98-134, KAIST, Department of Computer Science, 1998.

[9] T. H. Cormen, C. E. Leiserson and R. L. Rivest, “Introduction to Algorithms”, *McGraw-Hill Book Company*, pages 335~336, 1990.

[10] T. Imielinski and R. Badrinath, “Wireless mobile computing: Challenges in data management”, *Communications of the ACM*, 37(10), 1994.

[11] T. Imielinski and S. Viswanathan, “Adaptive Wireless information Systems”, *Proc. SIGDBS Conference*, 1994.

[12] H. V. Leong and A. Si, “Data Broadcasting Strategies over Multiple Unreliable Wireless Channels”, *Proc. International Conference on Information and Knowledge Management*, pages 96~104, 1995.

[13] K. Prabhakara, K. A. Hua and J. Oh, “Multi-Level Multi-Channel Air Cache Designs for Broadcasting in a Mobile Environment”, *Proc. IEEE International Conference on Data Engineering*, pages 167~176, 2000.

[14] K. Stathatos, N. Roussopoulos and J.S. Baras, “Adaptive Data Broadcast in Hybrid Networks”, in *Proc. VLDB Conference*, pages 326~335, 1997.