A Data Partition Scheme for Wireless Broadcast Mechanisms Based on Linear Programming

Kuen-Fang Jea Chun-Le Wu

Institute of Computer Science

National Chung-Hsing University

{kfjea, cosa}@cs.nchu.edu.tw

Abstract

Due to the limited bandwidth in the wireless environment, broadcasting is the most common way to service a large number of mobile clients at the same time. There are two main issues in such an environment: saving battery energy for mobile clients and accessing data quickly. In this paper, we aim at the energy saving issue and propose a data partition scheme to improve tuning time for the broadcast mechanisms based on linear programming. By investigating the geometric properties of data distribution curves and integrating them into the linear programming technique, we create the data partition scheme and use it to construct our broadcast mechanism. With this partition scheme, the complexity of linear programming decreases and data broadcast time is predicted more accurately. As a result, the bandwidth for storing polynomial coefficients is reduced and better tuning time is achieved. Experimental results in this paper also confirm these advantages of our broadcast mechanism.

1. Introduction

In the mobile computing environment, the power consumption problem is very important because, as [SCB92] pointed out, the development of battery technology is very slow. Power consumption at mobile clients can generally be distinguished into active mode and doze mode. [IVB94c] pointed out the power consumption at mobile clients in active mode is several thousand times of that in doze mode. Therefore, the

selective tuning technique is usually adopted for energy saving. To receive data from the broadcast channel, a mobile client follows the broadcast protocol to decide the time it switches between and the duration it stays in the doze mode and active mode.

In related research, there are two major metrics to measure the broadcast scheme's performance. *Tuning time* measures the total time a mobile client tunes in active mode, while *access time* measures the latency time a mobile client receives data. The effect of tuning time is on the power consumption of mobile clients, and inserting additional information into the channel for selective tuning is widely used in current research to reduce tuning time. The effect of access time on mobile clients determines whether they can receive data rapidly, and several techniques such as [AAF95][Go95][JV98][CK99] [Wa01] have been proposed to reduce access time.

However, inserting additional information into the broadcast channel for energy saving may enlarge the broadcast cycle and thus consume more bandwidth and increase access time. It may also cause the *directory miss* problem: the desired data bucket went past as the mobile client wakes up, and it should wait and get the same data bucket in the next broadcast cycle. Directory miss is undesirable as it may increase access time enormously.

Recently, [Wa01] proposed three broadcast schemes that can avoid directory miss and have reasonable good access time. Similar to the hashing-based schemes, they provide parameters in the broadcast cycle and allow mobile clients to predict their desired data in the cycle by computing these parameters. They also use the linear programming technique to minimize the prediction errors and thus reduce access time. However, these schemes may need to solve a large number of constraint equations, which depend on the number of data being broadcast, to derive a very high degree predicting polynomial. To overcome this problem, [Wa01] proposed a partitioned scheme that partitions data into equal subsets and derives a polynomial for each of

them. Although the complexity of deriving polynomials is reduced, the even-partition scheme still does not achieve very good tuning time. Moreover, it may need to store many polynomial coefficients in the broadcast cycle, which results in longer cycles and access time. To improve the drawbacks of the even-partition scheme, this paper proposes a new data partition scheme which is based on the geometric properties of data distribution curves and can select more appropriate polynomials for prediction. With this scheme, we also propose a new broadcast mechanism, which not only has better tuning time but also saves a lot of bandwidth for polynomial coefficients.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 explains the broadcast model based on linear programming and Section 4 presents our data partition scheme. Section 5 shows our broadcast mechanism with an analysis of the mechanism. Section 6 demonstrates experimental results of its performance and comparison with other related broadcast schemes. Finally Section 7 concludes this study.

2. Related Work

In this section, we review three kinds of broadcast schemes related to our research, including hashing-based schemes, indexing-based schemes and the predictive schemes based on linear programming.

Hashing-based schemes [IVB94c] insert hash functions into the broadcast channel as additional information in order to make mobile clients access desired data directly. If a perfect hash function can be obtained, then no collision occurs and thus no overflow chain [IVB94c] is created. In this case, hashing-based schemes will in general have the optimal tuning time. But in practice, the perfect hash function is very hard to find and they always cause overflow chains, so they will have longer tuning time. Besides, hashing-based schemes will cause the directory miss problem and

increase the access time.

Indexing-based schemes add an index structure into the broadcast channel as additional information. Mobile clients just have to read the index information and follow the index step by step to get the desired data. [IVB94c] proposed Flexible Indexing which divides the broadcast file into p segments and stores the control index in the first bucket in each data segment. Mobile clients tune into these head buckets and navigate the control index to receive their desired data.

In [IVB94a], (1,m) Indexing and its extension Distributed Indexing were presented. The index structure is built by B^+-tree and the leaf contains pointers to data items. (1,m) Indexing partitions data into m equal data segments. The whole index is broadcast followed by each partition of data. The main difference between (1,m) Indexing and Distributed Indexing is that only the portion of index tree related to a data segment is broadcast in front of that data segment in Distributed Indexing for better access time.

[Wa01] proposed schemes that predict data location by polynomials which are derived from linear programming. The coefficients of polynomials are stored in each bucket in the broadcast channel and mobile clients find data by calculating the polynomial using these coefficients.

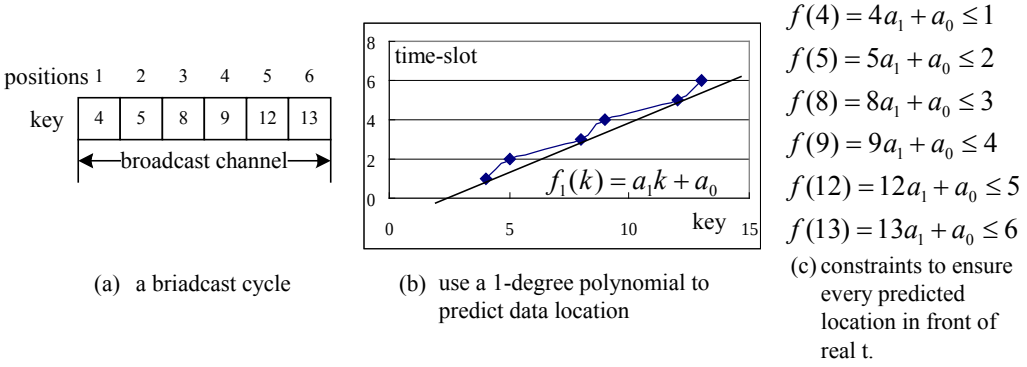
In [Wa01], three different schemes were presented, simple scheme, variation scheme and partitioned scheme. The simple scheme has the same predictive polynomial in each bucket and can predict the whole data set in the broadcast cycle. The variation scheme, on the other hand, has a different predictive polynomial in each bucket whose predicting scope is from the next bucket to the last bucket of the broadcast cycle. The partitioned scheme divides the data set into p equal segments and classifies the predictive polynomial into global and local types. The global polynomial predicts which data segment contains the desired data, while the local polynomial

predicts which time-slot (data bucket) contains the desired data. Although the partition scheme reduces the complexity of deriving polynomial and has better tuning time, it still leaves much space for improvement (as discussed in Section 1).

3. Linear Programming-based Broadcast Model

The broadcast problem studied in [Wa01] can be formulated as follows: Find a predictive function that can predict a data item's time-slot on the broadcast channel and always make the predicted position appear in front of the item's actual time-slot. The idea of linear programming-based broadcast model is to transform this broadcast problem into a linear programming problem and solve it by the linear programming technique.

Figure1 shows an example. Assume that each bucket only contains one data item and data are broadcast over a single channel in the ascending order of their key. If we consider a coordinate system where the vertical axis indicates the time slot a data item being broadcast (*t*-axis) in a broadcast cycle and the horizontal axis indicates the key value of the broadcast data item (*k*-axis), then each data item can be represented as a point (*k*, *t*) in this system. For the six data items in the broadcast cycle in Figure 1(a), they can be replaced by six points: (4,1), (5,2), (8,3), (9,4), (12,5) and (13,6). To predict the time-slot t when the data item *k* is broadcast, we may use a polynomial $f_1(k)$ of degree 1 (as shown in Figure 1(b)) with one constraint $f_1(k) \le t$ for each *k* (as shown in Figure 1(c)) to guarantee every predicted position always appears in front of the real broadcast time-slot *t*. Under these constraints, we can have the least prediction error by maximizing $\sum f_1(k)$, i.e. make $f_1(k)$ very close to the real t.



(a) a briadcast cycle

(b) use a 1-degree polynomial to predict data location

(c) constraints to ensure every predicted location in front of real t.

Figure 1. Example to illustrate the broadcast problem

On the other hand, the standard form of linear programming is as follows.

$$\text{maximize (or minimize) } z = c_1a_1 + c_2a_2 + \cdots + c_na_n \qquad (1)$$

$$\text{subject to} \quad c_{11}a_1 + c_{12}a_2 + \cdots + c_{1n}a_n (>, =, <) b_1$$
$$c_{21}a_1 + a_{22}a_2 + \cdots + c_{2n}a_n (>, =, <) b_2 \qquad (2)$$
$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$
$$c_{m1}a_1 + c_{m2}a_2 + \cdots + c_{mn}a_n (>, =, <) b_m$$
$$a_1, a_2, \cdots, a_n \ge 0 \qquad (3)$$

In the standard form, (1) is the objective function, (2) is a set of constraints, and (3) are nonnegative constraints. The meaning of the standard form is that under the constraints (2) and (3), the objective function (1) will be maximized or minimized.

By using the constraints in Figure 1(c), giving the objective function $Maximize \quad Z = \sum f_1(k) = 51a_1 + 6a_0$ and letting $a_i = a_{i1} - a_{i2}$, $a_{i1}, a_{i2} \ge 0$, we can transform the broadcast problem in Figure 1 into the following standard form of linear programming.

$$Maximize \quad Z = \sum f_1(k) = 51(a_{11} - a_{12}) + 6(a_{01} - a_{02}) \qquad (1)$$

$$Subject \ to \quad 4(a_{11} - a_{12}) + (a_{01} - a_{02}) \le 1$$
$$5(a_{11} - a_{12}) + (a_{01} - a_{02}) \le 2$$
$$8(a_{11} - a_{12}) + (a_{01} - a_{02}) \le 3 \qquad (2)$$
$$9(a_{11} - a_{12}) + (a_{01} - a_{02}) \le 4$$
$$12(a_{11} - a_{12}) + (a_{01} - a_{02}) \le 5$$
$$13(a_{11} - a_{12}) + (a_{01} - a_{02}) \le 6$$
$$a_{11}, a_{12}, a_{01}, a_{02} > 0 \qquad (3)$$

Based on this idea, the linear programming-based broadcast model, **LPB-Model**, can be defined as follows. Assume n data items are broadcast over a single channel in the ascending order of their keys k_i. In the (k, t) coordinate system, they form n coordinate points, $\{(k_1,t_1),(k_2,t_2),\cdots,(k_n,t_n)\}$, $k_i,t_i>0 \quad \forall i$, $k_1<k_2<\cdots<k_n$ and $t_1 \le t_2 \le \cdots \le t_n$. Let $c_{i,q}=(k_i)^q$ and $a_i=a_{i1}-a_{i2}$, $a_{i1},a_{i2}\ge 0$. We may obtain a q-degree polynomial $f_q(x)=\sum_{i=0}^{q} a_i x^i$ with $f_q(k_i) \le t_i \forall\, i = 1 \sim n$ such that mobile clients can use it to predict the time slot of their desired data k_i with the least prediction error by solving the following linear programming problem:

$$Maximize \quad Z=(\sum_{i=1}^{n} c_{i,q})(a_{q1}-a_{q2})+\cdots+(\sum_{i=1}^{n} c_{i,2})(a_{21}-a_{22})+(\sum_{i=1}^{n} c_{i,1})(a_{11}-a_{12})+n(a_{01}-a_{02})$$

Subject to

$$\begin{gathered}
c_{1,q}(a_{q1}-a_{q2})+\ldots+c_{1,2}(a_{21}-a_{22})+c_{1,1}(a_{11}-a_{12})+(a_{01}-a_{02}) \le t_1 \\
c_{2,q}(a_{q1}-a_{q2})+\ldots+c_{2,2}(a_{21}-a_{22})+c_{2,1}(a_{11}-a_{12})+(a_{01}-a_{02}) \le t_2 \\
\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
c_{n,q}(a_{q1}-a_{q2})+\ldots+c_{n,2}(a_{21}-a_{22})+c_{n,1}(a_{11}-a_{12})+(a_{01}-a_{02}) \le t_n \\
a_{q1},a_{q2},\ldots,a_{21},a_{22},a_{11},a_{12},a_{01},a_{02}>0
\end{gathered}$$

In this model, the goal of least prediction error indicates the shortest distance that mobile clients should navigate (stay in active mode) from the predicted location to the real location. As a result, tuning time is largely reduced and more battery energy is saved. According to this model, we define a function **LP()** which takes n data keys as input and generates the q-degree polynomial $f_q(x)=\sum_{i=0}^{q} a_i x^i$ in the LPB-Model for prediction. In fact, LP() implements the linear programming solution technique.

Complexity of LPB-Model

Karmarkar algorithm [Ka84] is the most common algorithm for linear programming and its time complexity is $O(v^{3.5}L^2 \ln L \ln\ln L)$, where v is the number of variables and L is the number of bits in the input. Generally we denote $O(v^{3.5}L^2)$ as its time complexity. In LPB-Model, there are $2(n+q+1)$ variables, n is the

number of data items and q is the degree of polynomial. Thus, the time complexity of LP() is $O((2n)^{3.5}L^2)$.

LPB-Model has the following properties, whose proofs can be found in [Wu02].

Theorem 1. LP() always generates the optimal polynomial $f_q(k) = \sum_{i=0}^{q} a_i k^i$.

Theorem 2. The polynomial of higher degree will have less error than the polynomial of lower degree for the same set of points (key, time-slot) in the (k, t) coordinate system.

As shown in [Wa01], the broadcast mechanism generated by this model may result in a lot of space reserved for polynomial coefficients in each bucket, especially when n is large and the degree q of the polynomial is very high. This may waste a lot of channel bandwidth and lead to longer tuning and access time. Therefore, in this study, we will apply a data partition scheme to solve this problem. In contrast to the even-partition scheme in [Wa01], this scheme partitions the data set into small subsets (groups) based on the properties of polynomial and data distribution in the (k, t) coordinate system, which generates better effect than the even-partition scheme.

4. Data Partition Scheme

The basic idea of our data partition scheme is to divide data into several groups (subsets) and make each group fit a q-degree polynomial. There are two kinds of predictive polynomials, GLPF (Global Linear Programming predictive Function) and LLPF (Local Linear Programming predictive Function). Each group has a LLPF to predict data items in the group. There is only one GLPF, which is used to predict the whole data set and makes sure that every predicted position always appears in front of the first bucket of the group containing the desired data.

Our data partition scheme works as follows. We first find out the candidate

points from which we can choose the boundary points for partition (as described in Section 4.1). Then, we repeatedly split the approximation intervals (defined by two consecutive candidate points) until the estimated error is smaller than a pre-specified bound (as discussed in Section 4.2). After that, by merging segments and minimizing the total estimated tuning time, we obtain the boundary points for partition (as stated in Section 4.3). Finally we construct predictive polynomials based on these points (as described in Section 4.4).

4.1 Select Candidate Points

Consider Figure 2 as an example, which shows the graph of a 4-degree polynomial $f(x) = x^4 - 6x^3 + 3x^2 + 26x - 24$, where there are two inflection points c and e. Line segments $\overline{bd}$, $\overline{df}$ pass through c, e respectively, and their slopes are the derivative of f(x) at c, e respectively. We can find two tangents $\overline{ab}$ and $\overline{bd}$ of f(x) at point a, g such that the distance between the tangent and f(x) is the smallest. Four segments $\overline{ab}$、$\overline{bd}$、$\overline{df}$ and $\overline{fg}$ are used to fit (approximate) f(x). Usually a q-degree polynomial can be fit (or approximated) by q line segments, i.e. we can treat each line segment as one degree of the polynomial.

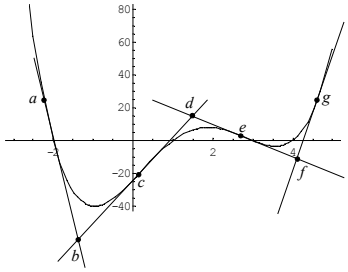


Figure 2. Properties of polynomial

To approximate data distribution curves, we define several parameters in Table 1.

Table 1. Parameters for data curves

Parameter	Description
RS_i	Slope of the line segment passing point (k_i, t_i) and point (k_{i+1}, t_{i+1}).
VS_i	Slope variation of RS_i and RS_{i-1}.

CP_i	Candidate points, from which we choose boundary points to partition data into groups.

These parameters are determined as follows. $RS_i = \frac{y_{i+1} - y_i}{k_{i+1} - k_i}$, where k_i is the key of the i[th] data item, y_i is the time-slot of the i[th] data item. $VS_i = \frac{RS_i}{RS_{i-1}}$, $i \geq 2$. Candidate points CP_i include the 1[st] data point, last data point and those points i that satisfy the condition $VS_i \geq 1$. Among these points, if $VS_i, VS_{i+1}, \cdots, VS_{i+j} > 1$ or $VS_i = VS_{i+1} = \cdots = VS_{i+j} = 1$, we choose the two points i and $i + j$ as candidate points (CP_i and CP_{i+j}). According to this rule, we may generate candidate points for further selection of the boundary points.

Figure 3 shows a data distribution curve where there are 7 candidate points CP_1, CP_4, CP_7, CP_9, CP_{11}, CP_{14} and CP_{15} in total. Let S_i denote the line segments connecting CP_i and CP_{i+j}, IS_i denote the interval between CP_i and CP_{i+j}, SS_i be the slope of S_i, respectively. There are three kinds of curves in IS_i, concave up (IS_1), concave down (IS_4, IS_7, etc.) and straight line (IS_{14}). If each interval IS_i only contains a few points, line segment S_i will be very similar and close to the data distribution curve in IS_i. We can then treat each line segment as one degree of the predictive polynomial and approximate the data distribution curve in Figure 3 by a 6-degree polynomial.

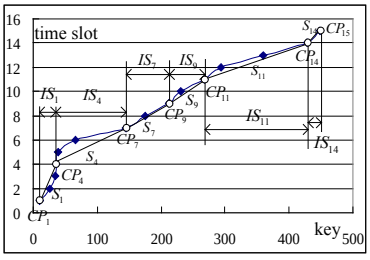


Figure 3. Data distribution curve and its approximation

4.2 Split Line Segments

The rule stated above for determining candidate points might not be accurate

enough to guarantee a reasonable small tuning time for some data points, if the data curve is partitioned based on these candidate points. Therefore, we need to refine some intervals IS_i (split line segments S_i to fit the curve more closely) and generate more candidate points. The condition to refine intervals is when the average tuning time (prediction error) for all data points is greater than 1. Assume there are n_i data points in IS_i, and the prediction error d_j for some point k_j can be computed by $| f(k_j) - S_i(k_j) |$ where f(x) is the polynomial for the data curve and S_i is the approximating line segment (tangent) in IS_i. Let e_i denote the estimated error of S_i, which is calculated by $e_i = \Sigma\ d_j$. Also let $E = \sum e_i$ denote the total estimated error for all of the n data points. Then the interval refinement condition is $E/n \geq 1$.

Once the interval refinement condition is met, we refine each of those intervals with largest estimated error and split its line segment into two new line segments. For an interval with a concave up data curve, we choose the point k that satisfies the condition $RS_{k-1} < SS_i < RS_k$ to split the line segment. For an interval with a concave down data curve, we choose the point k that satisfies the condition $RS_{k-1} > SS_i > RS_k$ to split the line segment. The split process is repeated until the refinement condition is not satisfied.

4.3 Select Boundary Points for Partition

As mentioned before, our broadcast mechanism uses two predictive polynomials to predict data locations, first by GLPF and then by LLPF. The total estimated prediction error must include the error from both GLPF and LLPF. The refinement process in last section basically minimizes the error, denoted by E, due to LLPF, i.e., the tuning time for finding desired data in that group. To select boundary points for partition, we also need to minimize the error, denoted by G, due to GLPF, i.e., the tuning time for finding which data group contains the desired data.

Figure 4(b) shows how mobile clients tune buckets and how we calculate the tuning time. There are 16 data buckets divided into 5 data groups. Buckets 1,5,7,11,14 (the shadow buckets) are the head buckets for these groups. If a mobile client wants to get data k, first it reads bucket 3 calculated by GLPF, and by the pointer in bucket 3 it reads bucket 5. If k is not in the data group (headed by bucket 5), then it reads the head bucket 7 of the next data group. Suppose that k is in this data group, it then reads bucket 8 calculated by LLPF there and performs a linear search from bucket 8 till it finds k. The total tuning time is 6, 3 due to finding the group (buckets 3, 5 and 7) and 3 due to search in the group (buckets 8, 9 and 10).

Figure 4(a) shows how to estimate *G*. Assume GLPF and LLPF are the degree-3 predictive polynomial. Data points a, b, c, d, e are the candidate points to delimit groups, which may correspond to the head buckets in Figure 4(b). Because the predicted positions by GLPF should appear in front of the head bucket of the data group containing the desired data, we may obtain those points A, B, C, and D in Figure 4(a), which represent the largest key and the earliest time-slot of each data group. Connecting these points sequentially, we obtain 3 Estimated Line segments AB, BC and CD for estimating the total tuning time. To estimate *G*, we calculate the number of candidate points between **ap** (actual position) and **pp** (predicted position) for every data point and sum them up.

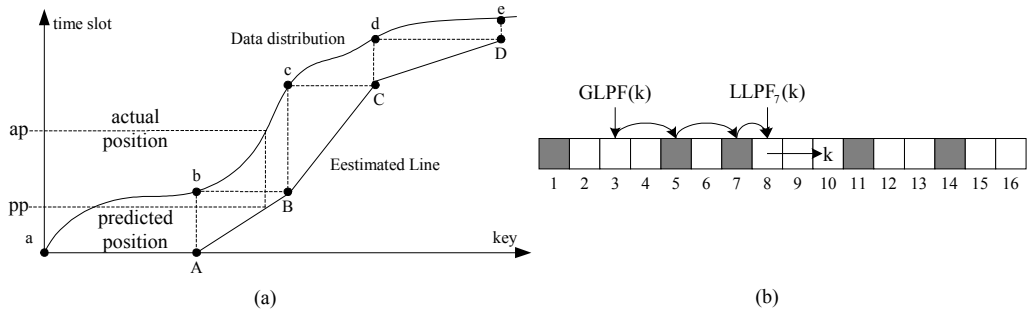


Figure 4. Estimate G

To minimize G+E, we merge two consecutive segments S_i and S_j into a new one if they have very small estimated error, that is, $|SS_i - SS_j|$ is the smallest for all the nearest $i, j \quad i \neq j$. In general, fewer data groups cost less tuning time for finding the wanted group but cost more tuning time for finding data in that group. So after reaching the condition ($\frac{E}{n} \leq 1$), we estimate G and compute G+E. Then we merge segments to maximize E (in order to minimize G), but $\frac{E}{n} \leq 2$ should be satisfied in the merge process. We keep on merging segments until the new (G+E) is larger than the old one. At this moment the remaining candidate points are the boundary points.

4.4 Construct Predictive Polynomial

In LPB-Model, the LP() function implements the linear programming solution technique. It takes as input a set of data points represented in the (k, t) coordinate system, and generates a q-degree predictive polynomial with prediction error minimized and predicted position in front of actual position. We may use this function to generate both global and local predictive polynomials (GLPF and LLPF). To construct a LLPF for each data group, we call LP() with all the data points in that group as input. On the other hand, to construct GLPF, we call LP() with all the boundary points as input. The detail algorithms are presented in [Wu02].

5. Broadcast Mechanism

This section describes our broadcast mechanism, including how to generate the broadcast structure and how mobile clients tune buckets to access data.

5.1 Broadcast Structure

Assume that there are n data items with keys $\{k_1, k_2, \cdots, k_n\}$ and each data item k_i is placed into a data bucket B_i respectively. These data items are partitioned into p

groups and each group contains m_i data items. Also assume that mobile clients have memory and computing ability.

The structure of data buckets is shown in Table 2. Buckets are classified into head buckets and non-head buckets. Each bucket has 2 parts, control part and data part. It stores the information necessary for predicting data locations and searching data.

Table 2. Bucket structure

Bucket type	Data/control part	Parameter	Description
Head bucket	Data part	$B_{i,1}.key$	Key of the data in this bucket
	Control part	$B_{i,1}.id$	Time slot over the channel
		$B_{i,1}.\max\_key$	The max key of the ith data group
		$B_{i,1}.ptr$	Pointer to the head bucket $B_{i+1,1}$
		$B_{i,1}.cycle$	Cycle length
		$B_{i,1}.GLPF$	Global predictive polynomial
		$B_{i,1}.LLPF$	Local predictive polynomial
Non-head bucket	Data part	$B_{i,q}.key$	Key of the data in this bucket
	Control part	$B_{i,q}.id$	Time slot in the channel
		$B_{i,q}.ptr$	Pointer to the head bucket $B_{i+1,1}$
		$B_{i,q}.\max\_key$	The max key of the ith data group

5.2 Generate Broadcast Cycle

The following Algorithm CBT constructs the broadcast structure based on the groups generated by the data partition scheme. CBT() takes as input the keys and the numbers of data in each group, and produces the content of every bucket over the broadcast channel. It places data as well as the parameters of GLPF and LLPF into buckets according to the bucket structure. The complexity of CBT() is in GLPF() and LLPF(), so its complexity is $O((2n)^{3.5}L^2)+\sum_{i=1}^{p}O((2m)^{3.5}L^2)$.

Algorithm CBT()

Input: data, and group information

Output: content of a broadcast cycle

{

//obtain the predictive polynomials

GLPF= GLPF($\{k_1, k_2, \cdots, k_n\}$ ， $\{m_1, m_2, \cdots, m_p\}$);//global predictive polynomial

LLPF[]=LLPF($\{k_1, k_2, \cdots, k_n\}$ ， $\{m_1, m_2, \cdots, m_p\}$);//local predictive polynomial

$m_0 = 0$;

for i=1 to p //p data groups

for q=1 to m_i //The i[th] part has m_i data items

if q == 1 //if head bucket

$B_{i,1}.key = k_{(\sum_{x=0}^{i-1} m_x)+1}$;

$B_{i,1}.id = (\sum_{x=0}^{i-1} m_x) + 1$;

$B_{i,1}.\max\_key = k_{(\sum_{x=0}^{i} m_x)}$;

$B_{i,1}.ptr = m_i$;

$B_{i,1}.cycle = n$;

$B_{i,1}.GLPF = \text{GLPF}$;

$B_{i,1}.LLPF = \text{LLPF[i]}$;

else //non-head bucket

$B_{i,q}.key = k_{(\sum_{x=0}^{i-1} m_x)+q}$;

$B_{i,q}.id = (\sum_{x=0}^{i-1} m_x) + q$;

$B_{i,q}.ptr = m_i - q + 1$;

$B_{i,q}.\max\_key = k_{(\sum_{x=0}^{i} m_x)}$;

endif

endfor

endfor

}

5.3 Tune Data Buckets

Assume that k_w is the key of the desired data, B_w is the bucket containing k_w and B_c is the current broadcast bucket over the broadcast channel. To search for k_w, a mobile client must first obtain the cycle length and GLPF information from B_c (if it did not access data before), and then it should compute $GLPF(k_w)$ and follow Algorithm TB below to obtain k_w. An example of tuning buckets was presented in

Figure 4(b) before.

Algorithm TB ()

Input: k_w

Output: B_w, the bucket containing k_w.

{

Step1. **if** k_w has been broadcast

Step2. Switch into doze mode and wait for the predicted bucket at time slot $= GLPF(k_w)$ in the next cycle;

Step3. **else if** the predicted bucket at $GLPF(k_w)$ has not been broadcast

Step4. Switch into doze mode and wait for the bucket at $GLPF(k_w)$;

endif

Step5. **if** k_w is not in current data group

Step6. Switch into doze mode and wait for the head bucket in next data group;

Step7. **else if** the current bucket is not the head bucket, **goto** Step9;

endif

Step8. Compute $LLPF(k_w)$;

Switch into doze mode, and wait for the bucket at time slot = $LLPF(k_w)$;

Step9. Read buckets sequentially till k_w is found;

}

6. Experiment

We implement a simulator and conduct several experiments to observe the performance of our broadcast scheme. The following schemes are also simulated and compared: Hashing B (HB) [IVB94c], Flexible Indexing (FI) [IVB94c], Simple Scheme, Variation Scheme, Partitioned Scheme [Wa01] and our scheme.

6.1 Simulation Environment

The computer platform is PC with 1 GHz CPU, 256 MB memory and WIN2000 OS. We use Borland C++ Builder 5.0 to develop the simulator. The broadcast database contains 2 types of data distribution: normal distribution of key values with standard deviation 2000 and mean value 6000 generated by EXCEL 2000, and random distribution with key values from 1 to 10000 by the random number generator in Borland C++ library.

6.2 Simulation Results

This section shows the experimental results. Figure 5 and Figure 6 show the tuning time of our scheme for different polynomial degrees and various numbers of data with key values having normal and random distributions respectively. In the following figures, k denotes the degree of polynomial, and the access or tuning time is measured in terms of number of buckets. It is observed that tuning time increases as the number of data increases, but it decreases as the higher degree of polynomial is used. Besides, the tuning time under normal distribution is higher than that under random distribution when k=2, but the tuning time under both distributions is about the same when k is larger than 2. The reason is that, the curve of normal distribution is like the curve of a degree-3 polynomial and using a degree-2 polynomial is not accurate enough to approximate its distribution curve.

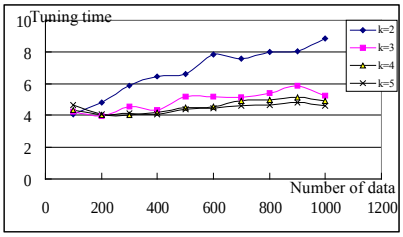


Figure 5. Tuning time for various numbers of data with keys having normal distribution

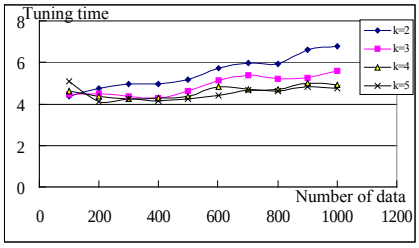


Figure 6. Tuning time for various numbers of data with keys having random distribution

Figure 7 shows the access time of various schemes. LP denotes the schemes based on linear programming, including Simple scheme (S), Variation scheme (V),

Partitioned scheme (P) and our scheme (O). FI denotes Flexible Indexing, and HB denotes Hashing B, respectively. We partition FI and HB into $\sqrt{n}$ data segments in the simulation because their access time is almost the best in all the ways of partitioning data segments. In Figure 7, LP has the best access time (about half of the number of data buckets) because this type of schemes does not cause directory miss. FI also has good access time, but HB has the worst access time.

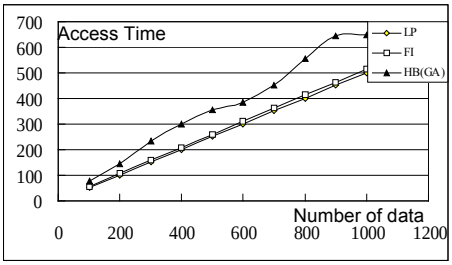


Figure 7. Access time for various schemes

Figure 8 and Figure 9 show the tuning time of various schemes for different number of data with keys having normal distribution and random distribution respectively. For schemes based on linear programming we let k=3 in our simulation because it is suitable for normal distribution and has reasonable good performance.

There are two implementations of Hashing B, HB(BA) and HB(GA), depending on the number of segments *p* into which the hash function divides data. HB(BA) has *p* near the number of data, which has very good tuning time but very bad access time. HB(GA) has *p* near $\sqrt{n}$, which has the average tuning time and the best access time.

It is observed from both Figure 8 and Figure 9 that, our scheme has the second best tuning time among all these schemes. Though HB(BA) has the best tuning time, it has a huge amount of access time (about twice the number of data buckets). HB(GA) has the worst tuning time but an average access time. Simple scheme has the worst tuning time among all the linear programming-based schemes, Variation scheme is better than Simple scheme, but worse than Partitioned scheme. FI also has not bad

tuning time.

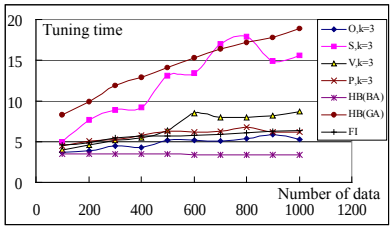


Figure 8. Tuning time of various schemes for different numbers of data with keys having normal distribution

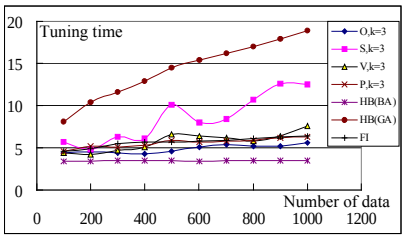


Figure 9. Tuning time of various schemes for different numbers of data with keys having random distribution

Figure 10 and Figure 11 compare the number of polynomials and space for polynomial coefficients among the linear programming-based schemes. In both figures, LP denotes the three schemes (S, V, P) presented in [Wa01]. Figure 10 shows the number of polynomials required by different schemes. Because all LP schemes require every bucket contains a polynomial, the number of polynomials and the number of data are the same. In our scheme, the polynomial is just stored in the head bucket of each data group.

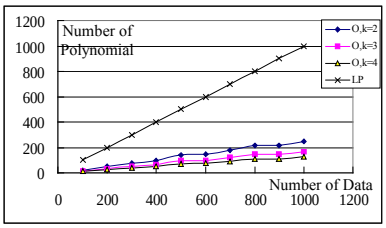


Figure 10. Number of polynomials required for various schemes

Figure 11 shows the space required for polynomial coefficients. In LP, it needs a

lot of space for coefficients. In our scheme, it needs much less space for coefficients.

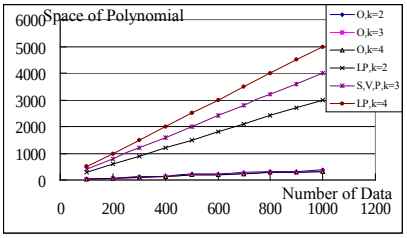


Figure 11. Space of polynomial coefficients required for various schemes

7. Conclusion

In general, many broadcast schemes cannot avoid the directory miss problem and this will make the access time increase greatly. Our broadcast scheme adopts the linear programming technique, and by the constraints imposed in linear programming, it can avoid the directory miss problem and achieve very good access time. In addition, our data partition scheme also shows that dividing data into groups based on the properties of polynomial and the data distribution curve can not only generate good tuning time, but also reduce a lot of space (bandwidth) for polynomial coefficients in the broadcast cycle.

References

[AAF95] S. Acharya, R. Alonso, M.Franklin, and S. Zdonik, “Broadcast Disks-Data Management for Asymmetric Communications Environment,” *Proceedings of ACM SIGMOD Conference*, 1995, pp.199-210.

[CK99] Y. D. Chung, and M. H. Kim, “QEM: A Scheduling Method for Wireless Broadcast Data,” *Proceedings of 6th International Conference on Database Systems for Advanced Application*, April 19-21, 1999, pp. 135-142.

[Go95] V. Gondhalekar, *Scheduling Periodic Wireless Data Broadcast*, M.S.Thesis, The University of Texas at Austin, U.S.A., 1995.

[IVB94a] T. Imielinski, S. Viswanathan and B.R.Badrinath, “Energy Efficient Indexing on Air,” *Proceedings of ACM SIGMOD Conference*, 1994, pp.25-36.

[IVB94b] T. Imielinski, S. Viswanathan and B.R.Badrinath, *Data on Air: Organization and Access*, Technical Report, Department of Computer Science, Rutgers University, U.S.A., 1994.

[IVB94c] T. Imielinski, S. Viswanathan and B.R.Badrinath, “Power Efficient Filtering of Data on Air,” *Proceedings of the International Conference on Extending Database Technology*, 1994, pp. 245-258.

[JV98] S. Jiang and N. Vaidya, *Scheduling Algorithms for a Data Broadcast System: Minimizing Variance of the Response Time*, Technical Report 98-005, Department of Computer Science, Texas A&M University, U.S.A., 1998.

[Ka84] N. Karmarkar, “A New Polynomial-time Algorithm For Linear Programming,” *Combinatorica*, Vol. 4, 1984, pp. 373-395.

[SCB92] S. Sheng, A. Chandrasekaran, and R.W. Broderson, “A Portable Multimedia Terminal for Personal Communications,” *IEEE Communications Magazine*, December 1992, pp. 64-75.

[Wa01] D.W. Wang, *A Broadcast Mechanism Based on Linear Programming*, M.S.Thesis, Institute of Computer Science, National Chung-Hsing University, 2001.

[Wu02] C.L. Wu, *Use Segmentation to Enhance the Wireless Broadcast Mechanism Based on Linear Programming*, M.S.Thesis, Institute of Computer Science, National Chung-Hsing University, 2002.