A Complementary Approach to Data Broadcasting in Mobile Information Systems [1]

Ye-In Chang, and Che-Nan Yang

Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
Republic of China
{E-mail: changyi@cse.nsysu.edu.tw}
{Tel: 886-7-5252000 (ext. 4334)}
{Fax: 886-7-5254301}

Abstract

Acharya et al. have proposed the use of a periodic dissemination architecture in the context of mobile systems, called *Broadcast Disks*. This strategy can construct a memory hierarchy where the highest level contains a few items and broadcasts them with high frequency while subsequent levels contain more and more items and broadcast them with less and less frequency. In this way, one can establish a trade-off between *access time* for high-priority data and that of the low-priority items, where *access time* means that the time elapsed from the moment a client submits a query to the receipt of data of his (her) interest on the broadcast channel. However, based on Acharya et al.'s algorithm, some broadcast slots may be unused, which results in the waste of bandwidth and the increase of access time. Therefore, in this paper, we propose an efficient broadcast program, the *complementary* approach, in which no empty slot is wasted. The basic idea of the complementary approach is to move some pages which are located near the end of a broadcast cycle to those empty slots which occur before those pages. Therefore, finally, the total number of slots in a broadcast cycle is equal to the one computed from Acharya et al.'s algorithm minus the number of empty slots. Obviously, our complementary approach generates a small number of slots in one broadcast cycle and shorter mean access time than Acharya et al.'s algorithm.

(**Key Words:** bandwidth, broadcast disks, broadcast schedule, mobile databases, mobile information systems.)

[1] This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-89-2218-E-110-004, and by National Sun Yat-Sen University.

1 Introduction

The emergence of powerful portable computers, along with advances in wireless communication technologies, has made mobile computing a reality [2, 9]. In the evolving field of mobile computing, there is a growing concern to provide mobile users with timely access to large amounts of information [19]. Examples of such services include weather, highway conditions, traffic directions, news and stock quotes.

Although a wireless network with mobile clients is essentially a distributed system, there are some characteristic features that make the system unique and a fertile area of research [2]. These features include *asymmetry in the communications*, *frequent disconnections*, *power limitations* and *screen size* [2]. Among them, asymmetry in the communications means that the bandwidth in the downstream direction (servers-to-clients) is much greater than that in the upstream direction. The communication asymmetry, along with the restriction in power that the mobile units have, have made the model of broadcasting data to the clients, an attractive proposition.

In traditional client-server information systems, clients initiate data transfers by sending requests to a server [1, 10]. Such systems are *pull-based*, which has the advantage of allowing clients to play a more active role in obtaining the data they need. However, pull-based systems are a poor match for asymmetric communications environments, as they require substantial upstream communications capabilities. Therefore, another *push-based* architecture that exploits the relative abundance of downstream communication capacity in asymmetric environments is proposed [13].

In a push-based information system, servers broadcast the desired data items in the broadcast channel continuously and repeatedly. The main advantage of broadcast delivery is its scalability: it is independent of the number of users the system is serving. In the simplest scenario, given an indication of the data items that are desired by each client listening to the broadcast, the server would simply take the union of the requests and broadcast the resulting set of data items cyclicly, as was done in Datacycle [5, 7]. Therefore, retrieving data pages from the broadcast channel can be viewed as sequentially accessing the broadcast data, where *access time* is the amount of time a client has to wait for an information item that it needs. It is important to minimize the *access time* so as to decrease

the idle time at the client [6]. Alternatively, the server can broadcast different items with differing frequency [1, 10]. Such a broadcast program can emphasize the most popular items and de-emphasize the less popular ones, which was proposed in Acharya et al.'s Broadcast Disks [1, 10].

There have been many strategies proposed for efficient broadcast delivery. Basically, those strategies can be classified into two types: *static* and *dynamic*. By "static broadcast", we mean that a broadcast where the schedule of programs is fixed and even though the contents of a program can change with time [1, 10]. In contrast, in "dynamic broadcast", both the schedule of programs and its contents can change and there exists limited support to handle user's requests [12, 14, 15, 19]. Also, there have been some researches on reducing access time [1, 10], nonuniform broadcast [1, 10, 16, 17], fault-tolerance [3, 4], and broadcast data on multiple wireless channels [18].

Among those strategies for efficient broadcast delivery, Acharya et al.'s Broadcast Disk strategy [1, 10] is one of well-known static algorithms. Based on Broadcast Disks, the server can construct a memory hierarchy in which the highest level contains a few items and broadcasts them with high frequency while subsequent levels contain more and more items and broadcast them with less and less frequency. However, based on Acharya et al.'s approach, some broadcast slots may be unused, which results in the waste of bandwidth and the increase of access time. Therefore, in this paper, we propose an efficient broadcast program, the *complementary* approach, in which no empty slots is wasted. The basic idea of the complementary approach is to move some pages which are located near the end of a broadcast cycle to those empty slots which occur before those pages. Therefore, finally, the total number of slots in a major cycle is equal to the one computed from Acharya et al.'s algorithm minus the number of empty slots. Obviously, our complementary approach generates a small number of slots in one broadcast cycle and shorter mean access time than Acharya et al.'s algorithm.

The rest of paper is organized as follows. In section 2, we give a brief description of Acharya et al.'s algorithm and show an example of the empty slot problem in Acharya et al.'s algorithm. In section 3, we present our complementary approach to solve the empty slot problem. In section 4, we study the performance of our complementary approach, and

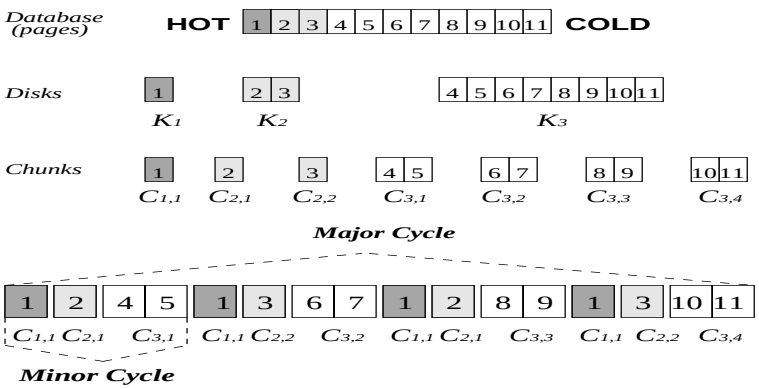


Figure 1: A broadcast program based on Acharya et al.'s algorithm

make a comparison with Acharya et al.'s algorithm. Finally, section 5 gives the conclusion.

2 Background

In Acharya et al.'s *Broadcast Disks* strategy [1, 10], the broadcast is created by assigning data items to different "disks" of various sizes and speeds, and then multiplexing the disks on the broadcast channel. Figure 1 shows an example of the broadcast program generation. Assume a list of pages that has been partitioned into three disks, in which pages in disk 1 are to be broadcast twice as frequently as pages in disk 2, and four times as frequently as pages in disk 3. Therefore, $R_1 = 4$, $R_2 = 2$, and $R_3 = 1$, where R_i is the relative broadcast frequency of disk i. Each disk i is split into NC_i chunks by first calculating L as the LCM (Least Common Multiple) of the relative frequencies and then being split into $NC_i = L/R_i$ chunks. That is, L is 4 (=LCM(4, 2, 1), so $NC_1 = 1$, $NC_2 = 2$, and $NC_3 = 4$. Finally, we create the broadcast program by interleaving the chunks of each disk in the following manner, where C_{ij} denotes the j'th chunk in disk i:

```
01 for i := 1 to L do
02    for j := 1 to S do begin
03        k := ((i - 1) mod NC_j) + 1;
04        Broadcast chunk C_{j,k}; end.
```

The resulting broadcast consists of 4 *minor cycles* (containing one chunk from each disk) which is the LCM of the relative frequencies, and has a period of 16 pages. This broadcast produces a three-level memory hierarchy in which disk one is the smallest and fastest level and disk three is the largest and slowest level. Thus, the multi-level broadcast corresponds to the traditional notion of a memory hierarchy.

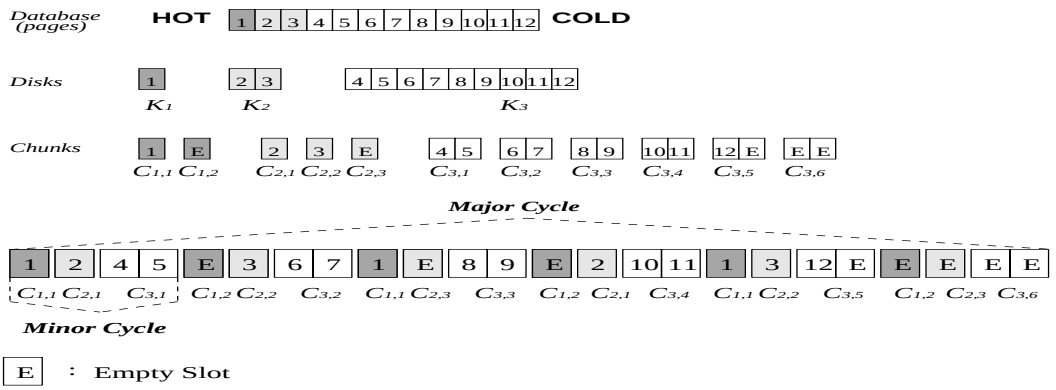


Figure 2: A broadcast program with 8 empty slots based on Acharya et al.'s algorithm

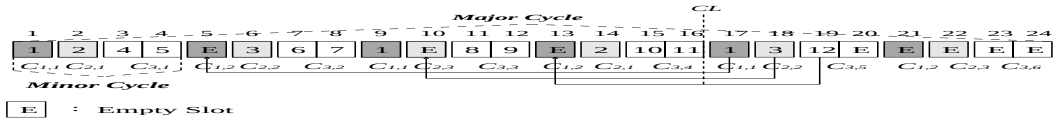
Figure 2 shows an example of a broadcast program generated by Acharya et al.'s algorithm, in which several empty slots can occur. Assume a list of pages that has been partitioned into three disks, and we have $R_1 = 3$, $R_2 = 2$, and $R_3 = 1$. These disks are split into chunks according to Acharya et al.'s algorithm. That is, we have L is 6, so $NC_1 = 2$, $NC_2 = 3$, and $NC_3 = 6$. The resulting broadcast program consists of 6 *minor cycles*, and has a period of 24 slots with 8 empty slots.

3 A Complementary Approach

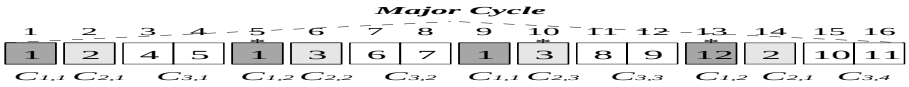
The basic idea of the complementary approach is to move some pages which are located near the end of a broadcast cycle to those empty slots which occur before those pages. Therefore, finally, the total number of slots in a broadcast cycle is equal to the one computed from Acharya et al.'s algorithm minus the number of empty slots.

3.1 Assumptions

This paper focuses on wireless broadcast environment. Some assumptions should be restricted in order to make our work feasible. These assumptions include: (1) The client population and their access patterns do not change. (2) Data is read-only. (3) There is no prefetching, and no cache scheme on the clients. (4) Clients make no use of their upstream communications capability. (5) When a client switches to the public channel, it can retrieve data pages immediately. (6) A query result contains only one page. (7) The server broadcasts pages over a single channel. (8) The broadcast infrastructure is reliable. (9)



(a)



(b)

Figure 3: A broadcast program based on the complementary approach: (a) deciding the cutline; (b) the result after the complementary approach.

The length of each page is fixed.

3.2 The Algorithm

Now, we present the proposed algorithm which partitions D pages into S broadcast disks such that no empty slot occurs. In the proposed algorithm, the following variables are used:

1. D: the number of pages;
2. P_i: the i'th page in a decreasing order of demand frequency, $1 \leq i \leq D$;
3. S: the number of disks;
4. R_i: the relative frequency of disk i, $1 \leq i \leq S$;
5. L: the least common multiple of R_i, $1 \leq i \leq S$, i.e., L = LCM (R_1, R_2, ..., R_S);
6. K_i: the number of pages in disk i, $1 \leq i \leq S$, and $\sum_{i=1}^{S} K_i = D$;
7. NC_i: the number of chunks in disk i, and $NC_i = L \ / \ R_i$, $1 \leq i \leq S$;
8. NS_i: the number of slots in a chunk of disk i, $1 \leq i \leq S$, i.e., $NS_i = \lceil \frac{K_i}{NC_i} \rceil = \lceil \frac{K_i}{L/R_i} \rceil = \lceil \frac{K_i \times R_i}{L} \rceil$;
9. C_{ij}: the j'th chunk in disk i, $1 \leq i \leq S$;
10. $Moved[]$: an array to store the pages after the cutline;
11. $Broadcast[]$: an array to store the data for broadcast;
12. O_{ijk}: the k'th slot of the j'th chunk in disk i, $1 \leq i \leq S$.

For the example shown in Figure 2, $S = 3$, $D = 12$ and we let $R_1 = 3$, $R_2 = 2$, and $R_3 = 1$. Therefore, we have L = LCM (3, 2, 1) = 6, $NC_1 = 2$, $NC_2 = 3$, $NC_3 = 6$, $NS_1 = 1$, $NS_2 = 1$, and $NS_3 = 2$.

In the complementary approach, we first compute the total number of slots in a major cycle, which is 24 in this example. Second, we compute the total number of empty slots in

such a major cycle, which is 8. Therefore, we can determine a cutline, as shown in Figure 3-(a), which is 8 slots away from the end of the major cycle. Third, we find those slots which are not empty after the cutline, which are slots 17, 18 and 19. Forth, we find the empty slots before the cutline, which are slots 5, 10 and 13. Finally, we move data page from slots 17, 18 and 19 to those empty slots 5, 10 and 13, respectively. The final result is shown in Figure 3-(b), where, the "*" symbol denotes those moved pages.

The main step in this strategy is how to detect whether a slot O_{ijl} is empty or not by checking the values of i, j, k only. Basically, we must consider two cases: (1) a fully wasted chunk, and (2) a partial wasted chunk. We now consider these two cases in details as follows:

1. For a fully wasted chunk in disk i:
   - Let $FW_i = NC_i - \lceil \frac{K_i}{NS_i} \rceil$, which is the number of chunks with fully wasted slots.
   - If $FW_i \geq 1$, then the fully wasted chunk j in disk i will occur in the range C_{ij} as follows: $NC_i - FW_i + 1 \leq j \leq NC_i$.
   - For such a fully wasted chunk C_{ij}, the number of wasted slots in C_{ij} is equal to NS_i. Moreover, the fully wasted chunk C_{ij} will occur in the $(j \times u)$'th minor cycle, $1 \leq u \leq R_i$, and O_{ijk} is an empty slot, $1 \leq k \leq NS_i$.
2. For a partially wasted chunk in disk i:
   - If there exists a chunk which contains w empty slots, $1 \leq w < NS_i$, then the following condition is satisfied: $PW_i = \lceil \frac{K_i}{NS_i} \rceil - \lfloor \frac{K_i}{NS_i} \rfloor = 1$.
   - If $PW_i = 1$, then the partially wasted chunk j in disk i will occur in C_{ij}, where $j = NC_i - FW_i$.
   - For such a partially wasted chunk C_{ij}, the number of wasted slots in C_{ij} is equal to $(NS_i \times NC_i - K_i - FW_i \times NS_i)$. Moreover, the partially wasted chunk C_{ij} will occur in the $(j \times v)$'th minor cycle, $1 \leq v \leq R_i$, and O_{ijk} is an empty slot, $NS_i - (NS_i \times NC_i - K_i - FW_i \times NS_i) + 1 \leq k \leq NS_i$.

The complete algorithm is described as follows:

1. Calculate the total slots (denoted as TS) and total wasted slots (denoted as TWS) in one major broadcast cycle generated by Acharya et al.'s broadcast disks program.
2. Find out the cutline (denoted as CL) which equals to (TS - TWS).
3. Find out the nonempty slots after the cutline and record them in the array $Moved$.
4. Let's use a sequence number (SN) to denote the sequence of those slots in a major cycle as $1, 2, ..., TS$. Find out the corresponding O_{ijk} of a SN before the cutline and record the corresponding O_{ijk} in an array $Broadcast$. If the corresponding O_{ijk} is empty then replace it with the data recorded in $Moved$.
5. Broadcast the contents of the $Broadcast$ array in sequence.
   for $a := 1$ to CL do Broadcast $Broadcast[a]$.

In step 1, the total number of slots (TS) is equal to

$$TS = L \times \sum_{i=1}^{S} NS_i = L \times \sum_{i=1}^{S} \lceil \frac{K_i \times R_i}{L} \rceil.$$

```
Input: A schedule created from Acharya et al.'s Algorithm, which contains a
       sequence of pages p_i in one major cycle, 1 ≤ i ≤ TS.
Output: One major cycle without empty slots and the length = TS − TWS.
1.          Procedure Complementary (TS, TWS: integer);
2.          begin
                (* phase 1: find nonempty slots after the cutline *)
3.              for p := (TS - TWS + 1) to TS
4.              begin
5.                  Call FindChunk(p, i, j, k);
                    (* find the corresponding O_ijk for slot p *)
6.                  if (Not FindEmpty(i, j, k)) then        (* a nonempty slot *)
7.                  begin
8.                      count := count + 1;
9.                      Moved[count] := O_ijk;
10.                 end;
11.             end;
                (* phase 2: find empty slots before the cutline *)
12.             q:=0;
13.             for p := 1 to (TS - TWS)
14.                 while (q < count) do
15.                 begin
16.                     Call FindChunk(p, i, j, k);
17.                     if (FindEmpty(i, j, k)) then        (* an empty slot *)
18.                     begin
19.                         q := q+1;
20.                         Broadcast[p] := Moved[q];
21.                     end
22.                     else
23.                         Broadcast[p] := O_ijk;
24.                 end;
25.         end;
```

Figure 4: The *Complementary* procedure

The total number of wasted slots (TWS) in one major cycle is

$$\begin{aligned} TWS &= \sum_{i=1}^{S}((NS_i \times NC_i - K_i) \times R_i) \\ &= \sum_{i=1}^{S}((NS_i \times \tfrac{L}{R_i} - K_i) \times R_i) \\ &= \sum_{i=1}^{S}(NS_i \times L - K_i \times R_i). \end{aligned}$$

In step 2, we let $CL = TS$ - TWS. For steps 3 and 4, we call Procedure *Complementary* as shown in Figure 4 which calls Procedure $FindChunk$ and Function $FindEmpty$ as shown in Figure 5 and Figure 6, respectively. The purpose of procedure $FindChunk$ is to find the corresponding O_{ijk} for a given sequence number. The purpose of the boolean function $FindEmpty$ is to check whether the input O_{ijk} is an empty slot or not, which is decided based on the observation as described before; that is, the decision is based on the values of FW_i, PW_i, j and k.

Note that, since the performance of how to decide the best match (which provides the optimal access time) between those moved pages after the cutline and those empty slots

Input: SN is the sequence number of O_{ijk}.
Output: i, j, k.

```
1.      Procedure FindChunk(SN: integer; var i, j, k: integer);
2.      begin
3.          i := 0;
4.          y := 0;
5.          for x := 1 to S do
6.              y := y + NS_x;      (* y is the length of a minor cycle, y = sum_{i=1}^{S} NS_i *)
7.          a := SN div y;
8.          b := SN mod y;                  (* Step A *)
            (* a = (j - 1) + (c div y) + (NC_i × z), c = sum_{x=1}^{i-1} NS_x + k *)
9.          if (b = 0) then                 (* Step B *)
            (* c = y, i = S, k = NS_i *)
10.         begin
11.             a := a - 1;
12.             b := y;
13.         end;
14.         while (b > 0) do        (* b = sum_{x=1}^{i-1} NS_x + k *)
15.         begin
16.             i := i + 1;
17.             k := b;
18.             b := b - NS_i;
19.         end;
              (* if (b ≤ 0) then SN ∈ disk i *)
              (* a means the number of minor cycles which occurs before O_ijk *)
20.         j := (a + 1) mod NC_i;          (* Step C *)
21.         if (j = 0) then j := NC_i       (* Step D *)
22.     end;
```

Figure 5: The *Findchunk* procedure

Input: i, j, k (* O_{ijk} is the chunk corresponding to the sequence number SN *)
Output: return $True$ when O_{ijk} is an empty slot.

```
1.      Function FindEmpty(i, j, k: integer): Boolean;
2.      begin
3.          FindEmpty := False;
4.          FW_i := NC_i - ⌈K_i / NS_i⌉;
5.          PW_i := ⌈K_i / NS_i⌉ - ⌊K_i / NS_i⌋;
6.          if (FW_i >= 1) then             (* a fully wasted chunk *)
7.          begin
8.              if (((NC_i - FW_i + 1) ≤ j) and (j ≤ NC_i)) then
9.                  FindEmpty := True;      (* O_ijk is an empty slot *)
10.         end;
11.         if (PW_i = 1) then          (* a partially wasted chunk *)
12.         begin
13.             if (j = (NC_i - FW_i)) then
14.             begin
15.                 num := (NS_i × NC_i - K_i - FW_i × NS_i);
16.                 if (((NS_i - num + 1) ≤ k) and (k ≤ NS_i)) then
17.                     FindEmpty := True;      (* O_ijk is an empty slot *)
18.             end;
19.         end;
20.     end;
```

Figure 6: The *FindEmpty* function

Table 1: Parameters used in the simulation

S	the number of disks
D	the number of distinct pages to be broadcast
K_i	the number of pages in disk i
R_i	the relative frequency of disk i
Δ	the broadcast shape parameter
θ	the $Zipf$ factor for partition size
γ	the $Zipf$ factor for frequency of access

before the cutline is $O(N!)$, where N is the number of those empty slots before the cutline, we apply the simplest way to deal with this match problem. In step 5, we construct the final broadcast program. To achieve this, we need a mapping procedure which maps a sequence number SN into the corresponding O_{ijk}; that is, we call procedure $FindChunk$. Basically, the sequence number for O_{ijk} is

$$(y \times (j-1) + \sum_{x=1}^{i-1} NS_x + k) + (TS/R_i) \times z,$$

where y is the total number of slots in a minor cycle, i.e., $y = \sum_{i=1}^{S} NS_i, 0 \leq z \leq R_i - 1$.

4 Performance Evaluation

In this section, we study the performance of our complementary approach and make a comparison with Acharya et al.'s algorithm. Our experiments were performed on a Pentium III 500 MHz, 128 MB of main memory, running Windows 98.

4.1 The Performance Model

The parameters used in the model are shown in Table 1. When we simulate the process of Acharya et al.'s algorithm, we need to decide the values of R_i's, which can be dependent on Δ. Using Δ, the frequency of broadcast R_i of each disk i, can be computed relative to R_S, the broadcast frequency of the slowest disk (disk S) as follows [1, 10, 17]: $\frac{R_i}{R_S} = (S-i)\Delta+1$, and $R_S = 1$, $1 \leq i \leq S$. For example, for a 3-disk broadcast, when $\Delta = 3$, the relative frequencies are 7, 4, and 1 for disks 1, 2, and 3, respectively.

Moreover, when we simulate the process of Acharya et al.'s algorithm, we need to decide the values of K_i's, which can be decided based on the $Zipf$ distribution [1, 10, 16, 17].

The *Zipf* distribution is typically used to model nonuniform access patterns [1, 10]. The *Zipf* distribution for a specific M can be expressed as $p_i = \frac{(1/i)^\theta}{\sum_{j=1}^{M}(1/j)^\theta}$, $1 \leq i \leq M$, where θ is a parameter named access skew coefficient or *Zipf* factor and $M \in N$. That is, it produces access patterns that become increasingly skewed as θ increases – the probability of accessing any page numbered i is proportional to $(1/i)^\theta$. For example, when $M = 3$, $\theta = 1$, we have $p_1 = \frac{6}{11}$, $p_2 = \frac{3}{11}$, and $p_3 = \frac{2}{11}$. Therefore, K_i in Acharya et al.'s algorithm can be decided based on the *Zipf-like* distribution as follows [1, 10, 17]: $K_i = D \times \frac{(\frac{1}{S-i+1})^\theta}{\sum_{j=1}^{S}(1/j)^\theta}$. Here, K_1 has the fewest pages, K_2 has the next fewest pages, and K_S has the most number of pages.

When we consider the demand frequency of data access for page i (denoted as DFP_i), we also apply the *Zipf* distribution with a *Zipf* factor γ. Here, we partition the pages into regions (= number of disks) of K_i pages each, where $1 \leq i \leq S$, and we assume that the probability of accessing any page within a region is uniform; that is, the *Zipf* distribution is applied to these regions [1, 10]. Therefore, we model the demand frequency of access of the ith disk (DFD_i) using the *Zipf* distribution as follows: $DFD_i = \frac{(1/i)^\gamma}{\sum_{j=1}^{S}(1/j)^\gamma}$, where γ is the *Zipf* factor of the *Zipf* distribution. In this case, the first disk (K_1), which has the least number of records, is the most frequently accessed, the second disk (K_2) is next, and so on. Since each page w in disk i has the same demand frequency DFP_w, we have $DFP_w = DFP_i$, $i \leq i \leq S$.

Two performance measures are considered in this comparison:

1. The total number of slots in one broadcast cycle.

2. The mean access time (or the expected time delay) denoted as $AccessT$, which equals to multiply the probability of access for each page i (DFP_i) with the expected delay for that page (EDP_i) and sum the results. That is, $AccessT = \sum_{i=1}^{D} EDP_i \times DFP_i$.

4.2 Performance Analysis

Let SP_i denote the distance (i.e., the number of slots) between the same page i in disk k occurring in a major cycle, where $SP_i = TS/R_k$. For the mean access time (EDP_i) for page i in disk k in Acharya et al.'s algorithm, it can be computed as follows: $EDP_i = (1/SP_i) \times ((SP_i - 0.5) + (SP_i - 1 - 0.5) + ... + (SP_i - (SP_i - 1) - 0.5)) = SP_i/2$

For the mean access time in the complementary approach, it can be computed as follows.

When $R_k = 1$, $EDP_i = (TS - TWS)\ /\ 2 = CL\ /\ 2$.

When $R_k \neq 1$, there are two cases to be considered:

(1) $\forall a \in D$, $a \notin$ Moved, i.e., page a does not need to be moved.

(2) $\forall a \in D$, $a \in$ Moved, i.e., page a must be moved.

Let $newSN_i^j$ denote the new sequence number of the j'th occurrence of page i in disk k after executing procedure *Complementary*, $1 \leq j \leq R_i$.
For case (1), $EDP_i = (1/CL) \times ((R_k - 1) \times SP_i^2/2 + (SP_i - TWS)^2/2))$.
For case (2), $EDP_i = (1/CL) \times (\sum_{j=1}^{R_k-1}(newSN_i^{j+1} - newSN_i^j)^2/2 + (CL - newSN_i^{R_k} + newSN_i^1)^2/2)$.

4.3 Simulation Results

In this simulation, we let $\theta = 0.8$, $\gamma = 0.9$. We consider 12 test samples which include the combinations of $S = 2$, 3, and 4 and $\Delta = 2$, 3, 4 and 5, respectively, for a fixed D that is a random value between 4000 and 5000. For each test sample, we compute the average result for 1000 values of D.

The total number of slots in the complementary approach is equal to (the total number of slots - the wasted slots) in Acharya et al.'s algorithm. Therefore, obviously, the total number of slots in the complementary approach is less than that in Acharya et al.'s algorithm.

When $\Delta = 2$, 3, 4 and 5, the detailed simulation results about the total number of slots in one broadcast cycle in the complementary approach and Acharya et al.'s algorithm for 1000 executions are shown in Tables 2, 3, 4 and 5, respectively. Obviously, our complementary approach always generates a smaller number of slots than Acharya et al.'s algorithm. As Δ is increased, the total number slots is increased in both the complementary approach and Acharya et al.'s algorithm. As S is increased, the total number of slots and the percentage of the total number of wasted slots are also increased in both the complementary approach and Acharya et al.'s algorithm.

A comparison of the mean access time (in terms of the time units) in the Acharya et al.'s algorithm and the complementary approach for 1000 executions is shown in Table 6. From

Table 2: $\Delta = 2$, $R_i = (S - i) \times 2 + 1$, $1 \leq i \leq S$

S	$Complementary$	BD	$TWS\ of\ BD$	$Maximum\ TWS$
2	7792	7793	1.0 (0.012%)	2 (0.026%)
3	10866	10884	18.4 (0.169%)	36 (0.331%)
4	13803	15540	201.4 (1.438%)	346 (2.471%)

$Complementary$: the total number of slots in the complementary approach.
BD: the total number of slots in Acharya et al.'s broadcast disk approach.
$TWS\ of\ BD$: the total number of wasted slots in Acharya et al.'s broadcast disk approach.
$Maximum\ TWS$: the maximum number of wasted slots in Acharya et al.'s broadcast disk approach.

Table 3: $\Delta = 3$, $R_i = (S - i) \times 3 + 1$, $1 \leq i \leq S$

S	$Complementary$	BD	$TWS\ of\ BD$	$Maximum\ TWS$
2	9436	9437	1.5 (0.016%)	3 (0.032%)
3	14046	14082	36.2 (0.257%)	70 (0.497%)
4	18451	18721	269.6 (1.440%)	511 (2.730%)

Table 4: $\Delta = 4$, $R_i = (S - i) \times 4 + 1$, $1 \leq i \leq S$

S	$Complementary$	BD	$TWS\ of\ BD$	$Maximum\ TWS$
2	11079	11081	2.0 (0.018%)	4 (0.036%)
3	17226	17286	60.0 (0.346%)	115 (0.665%)
4	23100	24335	1235.6 (5.077%)	1943 (7.984%)

Table 5: $\Delta = 5$, $R_i = (S - i) \times 5 + 1$, $1 \leq i \leq S$

S	$Complementary$	BD	$TWS\ of\ BD$	$Maximum\ TWS$
2	12722	12725	2.5 (0.032%)	5 (0.039%)
3	20406	20496	89.6 (1.382%)	179 (0.873%)
4	27748	28755	1006.6 (3.501%)	1597 (5.554%)

Table 6: A comparison of the mean access time

	Complementary				*BD*			
S	$\Delta = 2$	$\Delta = 3$	$\Delta = 4$	$\Delta = 5$	$\Delta = 2$	$\Delta = 3$	$\Delta = 4$	$\Delta = 5$
2	1172	1326	1491	1662	1172	1326	1491	1662
3	798	941	1089	1241	800	943	1093	1245
4	613	738	874	1000	619	746	909	1027

this result, we show that the mean access time in our complementary approach is always smaller than or equal to that in Acharya et al.'s algorithm. As S is increased, the access time is decreased in both the complementary approach and Acharya et al.'s algorithm. As Δ is increased, the access time is increased in both the complementary approach and Acharya et al.'s algorithm.

5 Conclusion

Broadcast data delivery is rapidly becoming the method of choice for disseminating information to a massive user population in many new application areas where client-to-server communication is limited. The main advantage of broadcast delivery is its scalability: it is independent of the number of users the system is serving. Based on Acharya et al.'s approach, some broadcast slots may be unused, which results in the waste of bandwidth and the increase of access time. In this paper, we have presented a complementary approach to solve the empty slot problem. From our performance analysis and simulation, we have shown that our complementary approach generates a small number of slots in one broadcast cycle and shorter mean access time than Acharya et al.'s algorithm. In environment where different clients may listen to different number of broadcast channels, the schedules on different broadcast channel should be coordinated so as to minimize the access time for most clients [18]. Therefore, how to design efficient broadcast programs for the case of broadcasting over multiple channels is one of the possible future research directions.

References

[1] S. Acharya, M. Franklin, S. Zdonik, and R. Alonso, "Broadcast Disks : Data Management for Asymmetric Communications Environments," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 199-210, San Jose, May 1995.

[2] D. Barbara, "Mobile Computing and Database-A Survey," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 11, No. 1, pp. 108-117, 1999.

[3] S. Baruah and A. Bestavros, "Pinwheel Scheduling for Fault-Tolerant Broadcast Disks in Real-Time Database Systems," *Proc. 13th Int'l Conf. Data Eng.*, pp. 543-551, London, March 1997.

[4] A. Bestavros, "AIDA-Based Real-Time Fault-Tolerant Broadcast Disks," *Proc. Real-Time Technology and Applications Symp.*, pp. 49-58, Boston, May 1996.

[5] T. Bowen, et al. "The Datacycle Architecture", *CACM*, Vol. 35, No. 12, pp. 71-81, Dec. 1992.

[6] S. Hameed, N. Vaidya, "Efficient Algorithm for Scheduling Data Broadcast," *Wireless Networks*, Vol. 5, No. 3, pp. 183-193, 1999.

[7] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pp. 97-103, San Francisco, May, 1987.

[8] T.Imielinski and B.R. Badrinath, "Querying in Highly Mobile and Distributed Environments," *Proc. 18th Int'l Conf. on Very large Data Bases*, pp. 41-52 , Vancouver, B.C., Canada, Aug. 1992.

[9] T.Imielinski and B. Badrinath, "Mobile Wireless Computing : Challenges in Data Management," *CACM*, Vol. 37, No. 10, pp. 18-28 , Oct. 1994.

[10] T. Imielinski and H.Korth, "Mobile Computing," *Kluwer Academic Publishers*, Chapter 12, pp. 331-361, 1996.

[11] T.Imielinski and S. Viswanathan, and B.R. Badrinath, "Data on Air : Organization and Access," *IEEE Trans. Knowledge and Data Eng.*, Vol. 9, No. 3, pp. 353-371, May/June, 1997.

[12] R. Kaza, "Adaption and Mobility in Wireless Information Systems," *IEEE Personal Comm.*, 1st Quarter, pp. 6-17 , 1994.

[13] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen, "The Information Bus – Architecture for Extensible Distributed Systems," *Proc. 14th SOSP Conf.*, pp. 58-68, Asheville, N.C., Dec. 1993.

[14] C. J. Su, L. Tassiulas, and V. Tsotras, "Broadcast Scheduling for Information Distribution," *Wireless Networks*, Vol. 5, No. 2, pp. 137-147, 1999

[15] K. L. Tan, J. X. Yu, "A Dynamic Schedule for the Infinite Air-Cache," *Data and Knowledge Eng.*, Vol. 24, No. 1, pp. 97-112, 1997.

[16] K. L. Tan, L. X. Yu, "Generating Broadcast Programs that Support Range Queries," *IEEE Trans. Knowledge and Data Eng.*, Vol. 10, No. 4, pp. 668-672, July/August, 1998.

[17] K. L. Tan, J. X. Yu, and P. K. Enk, "Supporting Range Queries in a Wireless Environment with Nonuniform broadcast," *Data and Knowledge Eng.*, Vol. 29, No. 2, pp. 201-221, 1999.

[18] N. H. Vaidya, S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," *Wireless Networks*, Vol. 5, No. 3, pp. 171-182, 1999.

[19] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, K. Ramamritham, "Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments," *Proc. 1997 Real-Time Technology and Applications Symp.*, pp. 38-48, June 1997.