An Improvement of the Complementary Approach to Data Broadcasting in Mobile Information Systems [1]

Ye-In Chang, and Shih-Ying Chiu

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*. However, based on Acharya et al.'s algorithm, some broadcast slots may be unused, which resulting in the waste of bandwidth and the increase of access time. Chang and Yang have presented a complementary approach to solve the empty slots problem. 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. However, based on the complementary approach, the distances between slots containing the same page may not be a constant, resulting in an increase of the mean access time. In this paper, we propose an efficient broadcast program, an improvement of the complementary approach, not only to mitigate the above phenomenon but also to solve the empty slots problem. The basic idea of the improvement of the complementary approach is try to move some "good" pages to those empty slots, as more as possible, where "good" pages are those pages which have relative request frequency = 1. From the simulation results, our improvement of the complementary approach generates a smaller number of slots in a broadcast cycle than Acharya's algorithm and needs shorter mean access time than Acharya's algorithm and the complementary approach.

(**Key Words:** bandwidth, broadcast disks, broadcast schedule, data broadcast, 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 National Sun Yat-Sen University.

1 Introduction

Wireless computing is more and more popular in recent years, because it can satisfy people's information needs at any time and any place. Wireless networks have five different characters from traditional fixed wired networks, including narrow bandwidth, an asymmetric communication environment, limited power, and portable units with very small screens [9]. These characteristics provide many new challenges for traditional techniques of a wired information system.

In wireless networks, servers can deliver data to clients in two modes: broadcasting mode and on-demand mode. Under the broadcasting mode, the server must construct a broadcast "program" to meet the needs of the client population [1]. The amount of time a client has to wait for an information item that it needs is called *access time*. A good broadcast program can minimize the mean access time. For example, data pages required by clients are A, B, and C. The required frequency of each data is 0.5, 0.25, and 0.25 respectively. The schedule in one broadcast cycle of the flat broadcast program is "ABC". In contrast, in the regular broadcast program, "$ABAC$", there is no variance in the inter-arrival time for each page. The performance characteristics of the regular program are the same as if page A was stored on a disk that is spinning twice as fast as the disk containing pages B and C. Thus we refer to the regular program as a Multi-disk broadcast, which was proposed in Acharya et al.'s Broadcast Disks [1].

There have been many strategies proposed for efficient broadcast delivery. In [3, 4], they presented real-time, fault-tolerant, secure broadcast organization technique. In [10, 11], they presented new algorithms to generate broadcast programs that facilitate range queries for multiple-disk broadcast programs. In [9, 12], they focused on the issue of scheduling the broadcast data on multiple wireless channels. In [5, 7, 9], they presented algorithms to consider the case that an MC may access more than one data page in one query.

Among those strategies for efficient broadcast delivery, Acharya et al.'s Broadcast Disks [2, 8] is one of well-known algorithms. Using Broadcast Disks 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. In this way, one can establish a trade-off between *access time* for

high-priority data and that of the low-priority items. 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. Chang and Yang have presented a complementary approach to solve the empty slot problem [6]. 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. However, based on the complementary approach, the distances between slots containing the same page may not be a constant, resulting in an increase of the mean access time.

Therefore, in this paper, we propose an efficient broadcast program, an improvement of the complementary approach, not only to mitigate the above phenomenon but also to solve the empty slots problem. If we want to move some pages on nonempty slots to these empty slots and expect a better performance for access time, we have to concern about two things: one is which pages on those nonempty slots are to be moved and the other is to which empty slots these pages on nonempty slots are to be moved. Basically, where there are N empty slots, there are $N!$ combinations to assign N pages into those N empty slots. However, even each of $N!$ combinations has been tried, the resulting performance is still not guaranteed to the globally optimal. Therefore, a heuristic approach to assign pages to those empty slots is a good choice between the trade-off of the mean access time and the computation time of the algorithm. In the improvement of the complementary approach, the basic idea is try to move some "good" pages to those empty slots, as more as possible, where "good" pages are those pages which have relative request frequency = 1. The reason for choosing such a good pages is that the locations of them in the broadcast cycle do not affect the mean access time. From our performance analysis and simulation, we show that our improvement of the complementary approach generates a smaller number of slots in one broadcast cycle than Acharya's algorithm and needs shorter mean access time than Acharya et al.'s algorithm and the complementary approach.

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

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

2 Background

2.1 Broadcast Disks

In Acharya et al.'s *Broadcast Disks* strategy [2, 8], 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 $R_1 = 3$, $R_2 = 2$, 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 6 (=LCM(3, 2, 1)), so $NC_1 = 2$, $NC_2 = 3$, and $NC_3 = 6$. 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 program consists of 6 *minor cycles* (containing one chunk from each disk) which is the LCM of the relative frequencies, and has a period of 30 slots with 7 empty slots.

2.2 The Complementary Approach

Chang and Yang have proposed a complementary approach to solve the empty slot problem occurring in Acharya et al.'s algorithm [6]. For the same example shown in Figure 1, in the complementary approach, it first computes the total number of slots in a major cycle, which is 30 in this example. Second, it computes the total number of empty slots in such a major cycle, which is 7 in this example. Therefore, the algorithm can determine a cutline, as shown in Figure 2-(b) which is 7 slots away from the end of the major cycle. Third, the algorithm finds those slots which are not empty after the cutline, which are slots 24 and

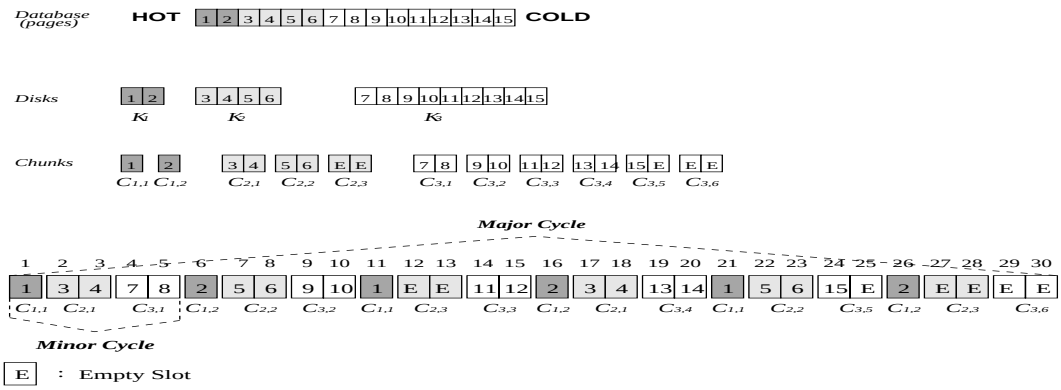
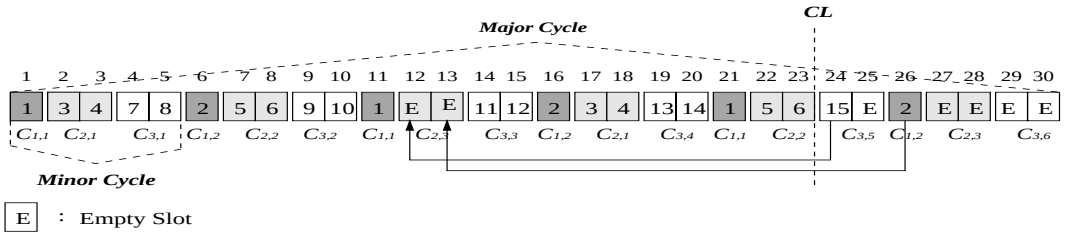


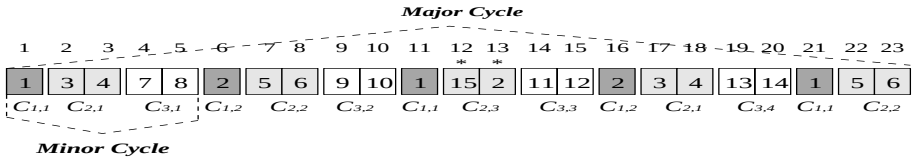
Figure 1: A broadcast program with 7 empty slots based on Acharya et al.'s algorithm

26 containing pages 15 and 2 in this example. Fourth, it finds the empty slot before the cutline, which are slots 12 and 13 in this example. Finally, the algorithm moves data pages from slots 24 and 26 to those empty slots 12 and 13, respectively. The final result is shown in Figure 2-(c), where, the "*" symbol denotes those moved pages.

Although the complementary approach solve the empty slot problem, it may result in the case in which the distances between slots containing the same page may not be a constant. For the example shown in Figure 3-(a), $S = 3$, $D = 19$ and we let $R_1 = 3$, $R_2 = 2$, and $R_3 = 1$. Therefore, $L =$ LCM (3, 2, 1) $= 6$, $NC_1 = 2$, $NC_2 = 3$, and $NC_3 = 6$. In this case, the total number of slots in a major cycle is 36 and the total number of empty slots in such a major cycle is 11. Therefore, we can determine a cutline, as shown in Figure 3-(b), which is 11 slots away from the end of the major cycle. The final result is shown in Figure 3-(c). By observing the final result of the complementary approach, shown in Figure 3-(c), we find that both slots 7 and 8 contain the same page, page 4. If some client wants to access page 4 but misses slots 7 and 8, then the client has to wait for the next cycle to achieve his goal; i.e. wait for up to 24 slots. From the example, we show that the complementary approach could induce long access time by the FIFO moving.



(a)



(b)

Figure 2: A broadcast program based on the complementary approach: (a) deciding the cutline; (b) the result after the complementary approach.

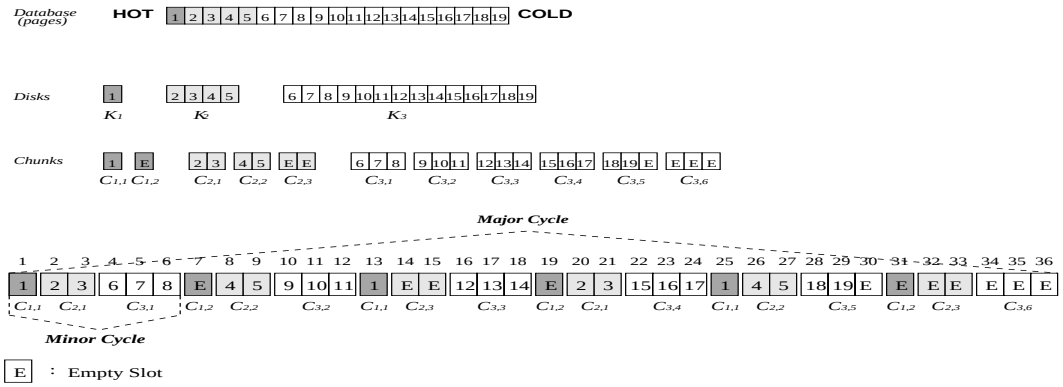
3 The Improvement of the Complementary Approach

3.1 Assumptions

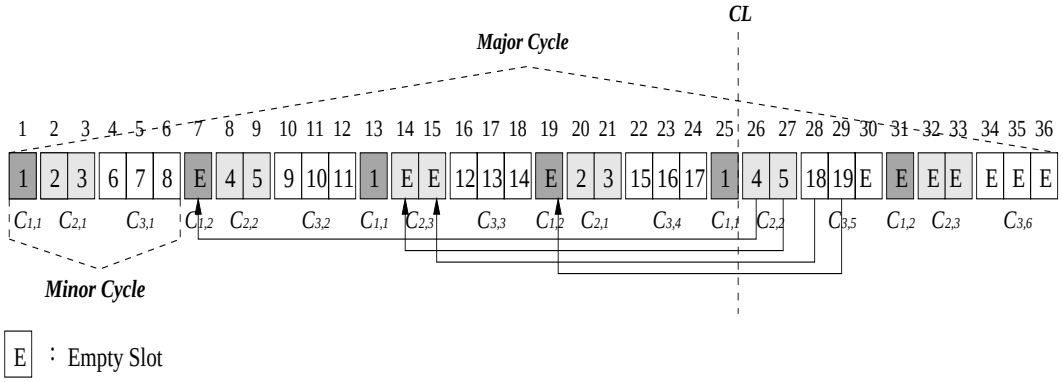
This paper focus on wireless broadcast environment. Some assumptions should be restricted in order to make our work feasible [5]. These assumptions include: (1) The client population and their access patterns do not change. (2) Data is read-only. (3) Clients retrieve data items from the broadcast on demand. (4) Clients are simple and without a great amount of memory. (5) Clients make no use of their upstream communications capability. (6) When a client switches to the public channel, it can retrieve data pages immediately. (7) A query result contains only one page. (8) The server broadcasts pages over a single channel. (9) The broadcast infrastructure is reliable. (10) The length of each page is fixed. (11) The relative frequency of the last disk is one.

3.2 The algorithm

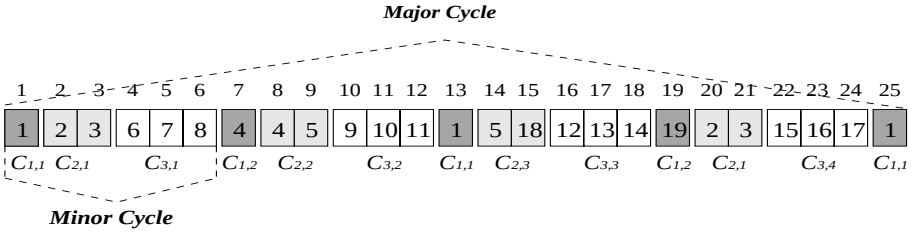
Now, we present the proposed algorithm which partitions D pages into S broadcast disks such that no empty slots occurs. In the proposed algorithm, the following variables are used:



(a)



(b)



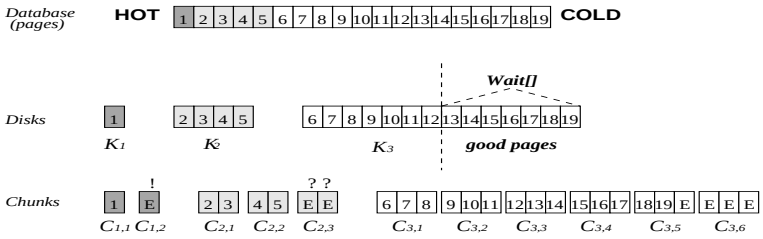
(c)

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

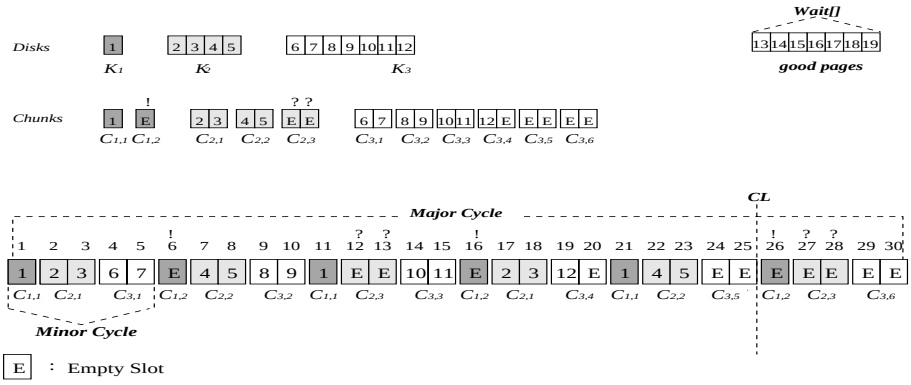
1. D : the number of pages;
2. P_i : the ith page in a decreasing order of demand frequency, $1 \leq i \leq D$;
3. S : the number of disks;
4. R_i : the relative frequency of disk i, $1 \leq i \leq S$; note that R_S must be one;
5. L : the least common multiple of R_i, $1 \leq i \leq S$, i.e., L = LCM (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 jth chunk in disk i, $1 \leq i \leq S$;
10. O_{ijk} : the kth slot of the jth chunk in disk i, $1 \leq i \leq S$.
11. HES (Hot Empty Slots) : the total number of empty slots with $R_i > 1$, i.e., $HES = \sum_{i=1}^{S-1}(NC_p \times NS_p - K_p) \times R_p$.
12. NTS (Nonempty Total Slots) : the total slots in a major cycle without empty slots, i.e., $NTS = \sum_{i=1}^{S} K_i \times R_i$.
13. $Wait$: an array which stores the pages with frequency = 1 and are ready to wait (or support) for complementing empty slots.
14. $Moved$: an array which stores the pages that occur after the cutline.
15. $Disk[k][d]$: the dth page in disk k.

Note that variables HES, NTS, $Wait$ and $Moved$ are new added, as compared to the complementary approach [6].

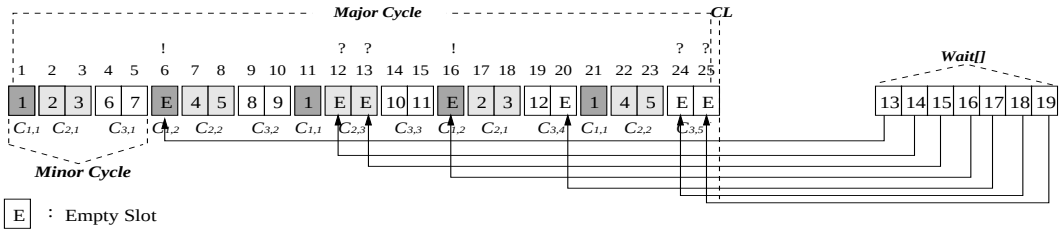
For the example shown in Figure 4-(a), we have $S = 3$ and $D = 19$. First, we order the pages from the hottest one to the coldest one. Second, we partition these 19 pages into 3 disks and let $K_1 = 1$, $K_2 = 4$, and $K_3 = 14$. Third, we let $R_1 = 3$, $R_2 = 2$, and $R_3 = 1$ and compute L = LCM (3, 2, 1) = 6. Fourth, we compute $NTS = \sum_{i=1}^{S} K_i \times R_i = 25$. Fifth, we compute $NC_1 = 2$, $NC_2 = 3$, $NC_3 = 6$, $NS_1 = 1$, $NS_2 = 2$, and $NS_3 = 6$. We can imagine the situation which partitions the pages on each disk into chunks, as shown in Figure 4-(a). For disk 1, $K_1 = 1$, $NC_1 = 2$, and $NS_1 = 1$, so there will be $(2 \times 1 - 1) = 1$ empty slot which is denoted as "!" and the empty slot occurs three times in a major cycle.



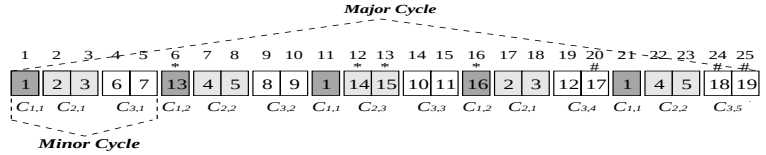
(a)



(b)



(c)



(d)

Figure 4: A broadcast program based on the improvement of the complementary approach: (a) ready to computing HES (b) ready to reserve "good" nonempty pages (c) deciding the cutline; (d) the result after the improvement of the complementary approach.

For disk 2, $K_2 = 4$, $NC_2 = 3$, and $NS_2 = 2$, so there will be $(3 \times 2 - 4) = 2$ empty slots which are denoted as "?" and these two empty slots occur two times in a major cycle. For disk 3, $K_3 = 14$, $NC_3 = 6$, and $NS_3 = 3$, so there will be $(6 \times 3 - 14) = 4$ empty slots and these four empty slots occur once in a major cycle. Since we want to keep the relative distance of the same page with relative request frequency > 1 as a constant as possible, we have to use those "good" pages to complement the empty slots occurring in the hot disks as possible as we can, where a hot disk i means $R_i > 1$. That's why we need to compute the number of empty slots in the hot disks in a major cycle, which is denoted as HES, and reserve HES good pages as possible as we can. In this example, $HES = 1 \times 3 + 2 \times 2 = 7$. Since the number of "good" pages in this example is 14, it is enough to reserve $HES = 7$ pages in the $Wait$ array.

Now, after moving out 7 pages, there are still remaining 7 pages ($K_3 = 14 - 7 = 7$), pages 6 to 12 in disk 3, as shown in Figure 4-(b). Then, we have to recompute $NS_3 = \lceil K_3/NC_3 \rceil = \lceil 7/6 \rceil = 2$. Sixth, we partition pages on each disk into chunks and the result is shown in Figure 4-(b). Seventh, let NTS be the cutline. Eighth, we find that there are only empty slots after the cutline, so no page has to be recorded in the $Moved$ array. In other words, the complement candidates will only come from the $Wait$ array. In Figure 4-(c), we find that 7 slots before the cutline are empty. Finally, we move 7 data pages in the $Wait$ array to those empty slots. The final result is shown in Figure 4-(d), where, the "$*$" or "#" symbols denote those moved pages. The "$*$" symbol denotes the empty slot which do not belong to disk 3. The "#" symbol denotes the empty slot which belong to disk 3.

Figure 4 is one of the best case since (1) only empty slots occur after the cutline, and (2) there are enough good pages to be reserved. In this case, for all the pages with relative request frequency > 1, the relative distance of the same page is guaranteed to be a constant except the distance between the last occurrence and the first occurrence of the same page. However, if (1) there are pages with frequencies > 1 occurring after the cutline or (2) there are not enough good pages, the decision about to which empty slots a page should be moved such that it can provide good performance is not an easy problem. For the first concern, we will put all the pages after the cutline into an array $Moved$. For the second

concern, we then have to choose some pages from those pages which have frequency > 1 after the cutline and are recorded in the *Moved* array. Therefore, in our approach, we will prepare two arrays: *Wait* and *Moved*. However, since we want to keep the relative distance of the same page with relative request frequency > 1 as a constant as possible, we apply the following policies to fill the empty slots: (1) For a hot empty slot, we prefer the pages stored in the *Wait* array to the *Moved* array, where a hot slot means that its relative frequency > 1. (2) For a cold empty slot, we prefer the pages stored in the *Moved* array to the *Wait* array, where a cold slot means that its relative frequency $= 1$.

The complete algorithm is described as follows:

1. Order the pages from the hottest one (most popular) to the coldest one.
2. Partition the list of pages into multiple disks ($= S$ disks), where each range contains pages with similar access probabilities.
3. Choose the relative frequency R_i of broadcast for each disk i ($i = \{0...S\}$), and calculate L as the LCM of the relative frequencies.
4. Calculate the total slots in a major cycle without empty slots and denote it as NTS.
5. Call procedure *Partition* to decide the values of NC_i, NS_i and HES first. Next, we move at most HES pages from disk S to the *Wait* array. Then, update K_S and NS_S if necessary.
6. Partition each disk i into NC_i chunks, and the number slots of NC_i is NS_i.
7. Let NTS be the cutline.
8. Call procedure *FindNE* to check whether there are nonempty slots after the cutline. If there are nonempty slots after the cutline, then record them in an array *Moved*.
9. Let's use a sequence number to denote the sequence of those slots in a major cycle as 1, 2,..., TS. Create a broadcast program consisting of those slots with sequence number 1 to NTS.
   for a:=1 to NTS do
   begin
     if the corresponding O_{ijk} is not empty then broadcast corresponding O_{ijk}
     else broadcast a data page from the *Wait* array or from the *Moved* array
     under some rules
   end.

In the above algorithm, what we do from Step 1 to Step 3 is to decide the values of K_i, R_i and L. In Step 4, we have to know how to compute the total number of slots in a major cycle without empty slots (NTS), $NTS = \sum_{i=1}^{S} K_i \times R_i$.

```
01          Procedure Partition;
02          begin
03            HES := 0;
04            for p := 1 to S do        (* Phase 1: compute NC_i and NS_i. *)
05            begin  NC_p := L/R_p;  NS_p := ⌈K_p/NC_p⌉;  end;
06            for p := 1 to S − 1 do        (* Phase 2: compute HES. *)
07              HES := HES + (NC_p × NS_p − K_p) × R_p;
               (* Phase 3: reserve at most HES pages in the Wait array,
                  and recalculate K_S and NS_S. *)
08             if HES ≥ 0 then     (* empty slots with R_i ≠ 1 occur *)
09             beign
10               Waitcount := 0;
11               if K_S ≥ HES then         (* have enough pages with R_i = 1*)
12                 Waitcount := HES
12               else        (* don't have enough pages with R_i = 1*)
12                 Waitcount := K_S;
13               K_S = K_S − Waitcount;    (* recalculate K_S and NS_S *)
14               NS_S = ⌈K_S/NC_S⌉; w := 1; d := K_S + 1;
                  (* start to move Waitcount pages to the Wait array *)
15               for p := 1 to Waitcount
16               begin  Wait[w] := Disk[S][d];  w := w + 1;  d := d + 1;  end;
17             end;
18           end;
```

Figure 5: The *Partition* procedure: Step 5

In Step 5, we call procedure *Partition* as shown in Figire 5. In procedure *Partition*, it contains three phases: the first phase is to decide the values of NC_i, NS_i and HES, the second phase is to calculate HES and the third phase is to move at most HES pages from disk S to the $Wait$ array and update K_S and NS_S. For the first phase, $NC_i = L/R_i$ and $NS_i = \lceil K_i/NS_i \rceil$. For the second phase, HES represents the number of hot empty slots which will occur in a major cycle, so we just need to accumulate the number of the empty slots except those from the last disk as follows: $HES = \sum_{i=1}^{S-1}(NC_i \times NS_i - K_i) \times R_i$. For the third phase, we try to reserve HES good pages in the $Wait$ array. There are two cases to be considered: (1) $K_S \geq HES$, and (2) $K_S < HES$. (Note that Case 1 denotes that we have enough good pages.) Here, we use $Waitcount$ to record the size of the $Wait$ array.

Lines 14 and 15 are the process for Case 1, and lines 16 and 17 are the process for Case 2. In Case 2, we will move all the pages from disk S to the $Wait$ array such that $Waitcount$ equals to K_S later. Lines 18 and 19 are the process to recalculate the value of K_S and NS_S. From lines 22 to 27, we start to move $Waitcount$ pages from disk S to the

Input: A schedule created from improved Acharya et al.'s Algorithm, which contains
a sequence of pages p_i in one major cycle, $1 \leq i \leq TS$.
Output: An array called Moved records the nonempty slot after the cutline and the size of the array, called count.

```
01          Procedure FindNE (NTS: integer);
02          begin
03              TS := 0;
04              for p := 1 to S do
05                  TS := TS + ⌈(K_i × R_i)/L⌉;
06              TS := L × TS;
07              for p := (NTS + 1) to TS do
08              begin
09                  Call FindChunk(p, i, j, k);
                    (* find the corresponding O_ijk for slot p *)
10                  if (Not FindEmpty(i, j, k)) then        (* a nonempty slot *)
11                  begin
12                      Movedecount := Movedcount + 1;
13                      Moved[Movedcount] := O_ijk;
14                  end;
15              end;
16          end.
```

Figure 6: The *FindNE* procedure: find nonempty slots after the cutline (Step 8)

Wait array. Here, we use $Disk[S][d]$ to represent the dth page in disk S. After getting the values of NC_i and NS_i from Step 5, in Step 6, we can partition the pages in each disk i into NC_i chunks, and the number slots of NC_i is NS_i. In Step 7, we have to decide where the cutline is. We let NTS to be the cutline.

In Step 8, to check whether there are nonempty slots after the cutline, we call procedure *FindNE* as shown in Figure 6, which calls Procedure *FindChunk* and Function *FindEmpty* as shown in Figure 7 and Figure 8, respectively. These procedures have been presented in details in Chang and Yang's complementary approach [6]. The purpose of procedure *FindChunk* is to map a sequence number SN into the corresponding O_{ijk}, which denotes the kth slot of the jth chunk in disk i. The purpose of the boolean function *FindEmpty* is to check whether the input O_{ijk} is an empty slot or not.

The purpose of procedure *FindNE* is to find nonempty slots after the cutline, then record them in the *Moved* array to wait for complementing empty slots. In the procedure, we have to calculate the total number slots in a broadcast cycle, denoted as TS, first. $TS = L \times \sum_{i=1}^{S} \lceil \frac{K_i \times R_i}{L} \rceil = L \times \sum_{i=1}^{S} NS_i$. Since we let NTS be the cutline, we have to

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

```
01          Procedure FindChunk(SN: integer; var i, j, k: integer);
02          begin
03              i := 0;     y := 0;
04              for x := 1 to S do
05                  y := y + NS_x; (* y is the length of a minor cycle, y = sum_{i=1}^{S} NS_i *)
06              a := SN div y;     b := SN mod y;                (* Step A *)
                (* a = (j - 1) + (c div y) + (NC_i × z), c = sum_{x=1}^{i-1} NS_x + k *)
08              if (b = 0) then                 (* Step B *)
                (* c = y, i = S, k = NS_i *)
09              begin   a := a - 1;     b := y;     end;
10              while (b > 0) do       (* b = sum_{x=1}^{i-1} NS_x + k *)
11              begin     i := i + 1;       k := b;     b := b - NS_i;      end;
                  (* if (b ≤ 0) then SN ∈ disk i *)
                  (* a means the number of minor cycles which occurs before O_ijk *)
12              j := (a + 1) mod NC_i;          (* Step C *)
13              if (j = 0) then j := NC_i       (* Step D *)
14          end;
```

Figure 7: The *FindChunk* procedure

check whether the pages from slot $(NTS + 1)$ to slot TS are empty or not. Then, we will move those nonempty pages after the cutline to the $Moved$ array. In the case that the size of the $Wait$ array is less than HES, we are sure that there must be pages after the cutline.

In Step 9, we call the *Broadcast* procedure as shown in Figure 9, which calls Procedure $FindChunk$ and Function $FindEmpty$ as shown in Figure 7 and Figure 8, respectively, to construct the final broadcast program. In the *Broadcast* procedure, if O_{ijk} is not empty, we broadcast the corresponding page, which O_{ijk} denotes the kth slot of the jth chunk in disk i. On the other hand, if O_{ijk} is empty, we have to choose where the complement page comes from and broadcast the slot using the complement page. The complement page can come from two places: one is the $Wait$ array and the other is the $Moved$ array. Since we want to keep the relative distance of the same page with relative request frequency > 1 as a constant as possible, we prefer to moving good pages which have been reserved in the $Wait$ array to those hot empty slots. Therefore, we have to check whether the frequency of the empty slot is one or not. There are two cases to be considered: (1) The frequency of the empty slot O_{ijk} is not one, i.e. $R_i \neq 1$ and $i \neq S$. (2) The frequency of the empty slot O_{ijk} is one, i.e., $R_i = R_S = 1$. For Case 1, we will check the $Wait$ array first. If there are

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.

```
01        Function FindEmpty(i, j, k: integer): Boolean;
02        begin
03          FindEmpty := False;
04          FW_i := NC_i - ⌈K_i / NS_i⌉;
05          PW_i := ⌈K_i / NS_i⌉ - ⌊K_i / NS_i⌋;
06          if (FW_i >= 1) then            (* a fully wasted chunk *)
07          begin
08            if (((NC_i - FW_i + 1) ≤ j) and (j ≤ NC_i)) then
09              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 8: The *FindEmpty* function

pages in the $Wait$ array, we will move one page to the empty slot (Case 1-(a)). However, if there is no page in the $Wait$ array, we will move one page from the $Moved$ array to the empty slot (Case 1-(b)). The reason is that we want to move good pages to those hot empty slots.

For Case 2, we will check the $Moved$ array first. If there are pages in the $Moved$ array, we will move one page from the $Moved$ array to the empty slot (Case 2-(a)). However, if there is no page in the $Moved$ array, we will move one page from the $Wait$ array to the empty slot (Case 2-(b)). The reason is that good pages are reserved to be the complement candidates for those hot empty slots.

In the *Broadcast* procedure, we use $Waitcount$ and $Movedcount$ to represent the size of $Wait$ and $Moved$ array, respectively. In the meanwhile, we use $Windex$ and $Mindex$ to record the location after which the pages in the array have not been used. If $Windex$ equals to $Waitcount$, it means that all the pages in the $Wait$ array have been chosen to be candidates. Similarly, $Mindex$ equals to $Movedcount$, it means that all the pages in the

Input: A schedule created from improved Acharya et al.'s Algorithm, which contains
a sequence of pages p_i in one major cycle, $1 \leq i \leq TS$.
Output: One major cycle without empty slots and the length = NTS.

```
01          Procedure Broadcast (NTS, Waitcount, Movedcount: integer);
02          begin
03              Windex := Mindex := 0;
04              Mindex := 0;
04              for p := 1 to NTS do
05              begin
06                  Call FindChunk(p, i, j, k);
07                  if FindEmpty(i, j, k) then        (* an empty slot *)
08                  begin
09                      if i ≠ S then
10                      begin       (*Case 1: O_ijk=empty and R_i > 1*)
11                          if Windex < Waitcount then
12                          begin         (*Case 1-a: fill with O_ijk ≠empty and R_i = R_S = 1*)
13                              Windex := Windex + 1;
14                              Broadcast Wait[Windex];
15                          end
16                          else if Mindex < Movedcount then
17                          begin         (*Case 1-b: fill with O_ijk ≠ empty after cutline*)
18                              Mindex := Mindex + 1;
19                              Broadcast Moved[Mindex];
20                          end;
21                      end     (*end of Case 1*)
22                      else
23                      begin       (*Case 2: O_ijk=empty and R_i = 1*)
24                          if Mindex < Movedcount then
25                          begin         (*Case 2-a: fill with O_ijk ≠empty after cutline *)
26                              Mindex := Mindex + 1;
27                              Broadcast Moved[Mindex];
28                          end
29                          else if Windex < Waitcount then
30                          begin         (*Case 2-b: fill with O_ijk ≠ empty and R_i = R_S = 1*)
31                              Windex := Windex + 1;
32                              Broadcast Wait[Windex];
33                          end;    (*end of case 2*)
34                      end;
35                  end    (*end of if FindEmpty (i,j.k)*)
36                  else Broadcast O_ijk;     (*non-empty slot*)
37              end; (*end of for p := 1 to NTS*)
38          end.
```

Figure 9: The *Broadcast* procedure (Step 9)

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

Moved array have been chosen to be candidates.

4 Performance Study

4.1 The Simulation 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 [2, 8, 11]: $\frac{R_i}{R_S} = (S-i)\Delta+1$, 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 [2, 8, 10, 11]. The *Zipf* distribution is typically used to model nonuniform access patterns [2, 8]. The *Zipf* distribution 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 $M \in N$. 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 [2, 8, 11]: $K_i = D \times \frac{(\frac{1}{S-i+1})^\theta}{\sum_{j=1}^{S}(1/j)^\theta}$. 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 [2, 8]. 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 $DFP_w = DFP_i$, $1 \leq i \leq S$.

Two performance measures are considered in this comparison: (1) The total number of slots in one broadcast cycle. (2) The mean access time (denote 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, where EDP_i denotes the average expected delay time for page i in disk k with the relative frequency = R_k. 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 complementary approach, 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 jth 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), $EDP_i = (1/CL) \times (\sum_{j=1}^{R_k - 1}(newSN_i^{j+1} - newSN_i^j)^2/2 + (CL - newSN_i^{R_k} + newSN_i^1)^2/2)$.

For the improvement of the complementary approach, when $R_k = 1$, we have $EDP_i = NTS\ /\ 2$. When $R_k \neq 1$, similar to the complementary approach, there are two cases to be considered. Let $orgSN_i^j$ denote the original sequence number of the jth occurrence of page i in disk k and let $newSN_i^j$ denote the new sequence number of the jth occurrence of page

Table 2: The parameters and their default settings

$Parameter$	$Default\ value$
S	3, 4
D	4000 .. 5000
Δ	4, 5
θ	0.8
γ	0.9

i in disk k after executing procedure $Broadcast$, $1 \leq j \leq R_i$. For Case (1), we have $EDP_i = (1/NTS) \times ((R_k - 1) \times (orgSN_i^2 - orgSN_i^1)^2/2 + (NTS - orgSN_i^{R_k} + orgSN_i^1)^2/2))$. For Case (2), we have $EDP_i = (1/NTS) \times (\sum_{j=1}^{R_k - 1}(newSN_i^{j+1} - newSN_i^j)^2/2 + (NTS - newSN_i^{R_k} + newSN_i^1)^2/2)$.

4.3 Simulation Results

In this simulation, we let $\theta = 0.8$, $\gamma = 0.9$. We consider 4 test samples which include the combinations of $S = 3$ and 4 and $\Delta = 4$ 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 parameters and their default settings are shown in Table 2.

The total number of slots in the improvement of the complementary approach is equal to (the total number of slots - the wasted slots) in Acharya's algorithm, and is equal to that in the complementary approach. Therefore, obviously, the improvement of the complementary approach generates a smaller number of slots in a broadcast cycle than Acharya's algorithm, as shown in Tables 3 and 4, respectively. As Δ (or S) is increased, the total number of slots is increased in both the improvement of the complementary approach and Acharya et al.'s algorithm.

A comparison of the mean access time (in terms of the time units) is shown in Table 5. From this result, we show that the mean access time in our improvement of the complementary approach is always smaller than that in Acharya et al.'s algorithm and the complementary approach. As S is increased, the access time is decreased in all three algorithms. As Δ is increased, the access time is increased in all three algorithms.

Table 3: $\Delta = 4$, $R_i = (S - i) \times 4 + 1$, $1 \leq i \leq S$

S	IC	BD	$TWS\ of\ BD$	$Maximum\ TWS$
3	17197	17257	59 (0.34%)	115 (0.67%)
4	23098	24338	1239 (5.09%)	1943 (7.98%)

IC: the total number of slots in the improvement of 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 4: $\Delta = 5$, $R_i = (S - i) \times 5 + 1$, $1 \leq i \leq S$

S	IC	BD	$TWS\ of\ BD$	$Maximum\ TWS$
3	20372	20463	90 (0.44%)	180 (0.88%)
4	27746	28751	1004 (3.49%)	1597 (5.55%)

5 Conclusion

The main advantage of broadcast delivery is its scalability: it is independent of the number of users the system is serving. In this paper, we have proposed an efficient broadcast program, the improvement of the complementary approach, which solves the empty slots problem and provides a good performance. From our performance analysis and simulation, we have shown that our improvement of the complementary approach generates a smaller number of slots in one broadcast cycle than Acharaya et al.'s algorithm. Moreover, our improvement of the complementary approach requires shorter mean access time than Acharya et al.'s algorithm and the complementary approach. 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. of ACM SIGMOD Intl. Conf. on Management of Data*, pp. 199-210, San Jose, May 1995.

Table 5: A comparison of the mean access time

	IC		BD		C(Complementary)	
S	$\Delta = 4$	$\Delta = 5$	$\Delta = 4$	$\Delta = 5$	$\Delta = 4$	$\Delta = 5$
3	1086.887	1237.533	1090.710	1243.114	1087.295	1238.849
4	864.575	993.043	908.617	1207.192	873.928	999.642

[2] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a Broadcast Disk, " *Proc. of the 12th IEEE Intl. Conf. on Data Eng.*, pp. 276-285, New Orleans, 1996.

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

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

[5] C. C. Chen, "Compression-based Broadcast Data for Reducing Access Time in Wireless Environment," *Proc. of 1999 National Computer Symposium*, Vol. 3, pp. 539-546, 1999.

[6] Y. I. Chang and C. N. Yang, "A Complementary Approach to Data Broadcasting in Mobile Information Systems," *Data and Knowledge Engineering*, Vol. 40, No. 2,pp. 181-194, Feb. 2002.

[7] Y. D. Chung and M. H. Kim "QEM: A Scheduling Method for Wireless Broadcast Data," *Proc. of the 6th Intl. Conf. on Database Systems for Advanced Applications*, pp. 135-142, Hsihchu, Taiwan, April 1999.

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

[9] C. H. Ke, "Broadcast Scheduling for Multiple Channels in Wireless Information Systems," *Proc. of 1999 National Computer Symposium*, Vol. 3, pp. 525-532, 1999.

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

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

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