Efficient Broadcast Disks Program Construction in Asymmetric Communication Environments

Eleftherios Tiakas, Stefanos Ougiaroglou, Petros Nicopolitidis
Department of Informatics, Aristotle University of Thessaloniki
Box 888, 54124, Thessaloniki, Greece

tiakas@csd.auth.gr, stefanos.ougiaroglou@gmail.com, petros@csd.auth.gr

ABSTRACT

A well-known technique for broadcast program construction is the Broadcast Disks. However, it has important disadvantages, as for example that the broadcast program construction procedure leaves some parts of the broadcast program empty. This paper proposes a new approach for the construction of the broadcast program. Specifically, it presents three new algorithms, which face the problems of the Broadcast Disk Technique. According to our approach, the broadcast program is constructed with the minimum possible length, respecting the selected disk relative frequencies and keeps the average delays of retrieving data-items low. The constructed broadcast programs have no empty parts and retain their desired properties in any combination of disk relative frequencies. Experimental results show that this approach is more efficient than Broadcast Disks in all cases.

Categories and Subject Descriptors

C.2.m [**Computer Systems Organization**]: Computer-Communication Networks – *Miscellaneous.*

General Terms

Algorithms

Keywords

Broadcast disks, asymmetric communication environments, data broadcasting.

1. INTRODUCTION AND BACKGROUND

In over a decade broadcasting data methods have been chosen to handle asymmetric environment communication systems. Some of today's applications that use broadcasting methods are: (i) tele-text and radio-text systems, (ii) weather, advanced traffic, stock prices and hospital information systems, (iii) public safety applications, etc. In all these applications, a server broadcasts data to clients. So, the broadcast channel becomes a "spinning disk" from which clients can retrieve data as it goes by. One of the most well known such methods is the Broadcast Disks Technique [3].

A major advantage of broadcasting methods is that the system performance does not depend on the number of connected users that are listening. Many research works focus on how to improve the efficiency of the broadcasting systems. Interesting research has been made in both sides of the problem: server side and client side, where special automation methods and caching strategies have been presented. In this paper, we will concentrate in the server side and specifically in structuring the broadcast program.

2. MOTIVATIONS

In this section, we present some interesting properties and bounds, and depict the drawbacks and restrictions of the broadcast disks method [3,8]. Then we will present the motivations for our approach. Table 1 depicts basic notations that will be used in all the following sections for reasons of simplicity. We also refer to the broadcast disks method as BD.

Table 1. Notations used in the following sections.

N	The selected number of different broadcast disks
M	The total number of data-items that will be broadcasted
f_i	The relative frequency of disk d_i, ($i = 1,..., N$)
LCM	The least common multiple of all frequencies f_i
nc_i	The number of chunks of disks d_i
nd_i	The number of data-items assigned to each disk d_i
PR	The length of a broadcast program period in slots
ES	The number of slots that remain empty in a period
del_i	The expected delays of data-items of disks d_i

In BD the identities: $nc_i = \frac{LCM}{f_i}$, $\sum_{i=1}^{N} nd_i = M$ also hold, and the following properties and bounds can be proved:

(1) $PR = LCM \sum_{i=1}^{N} \left\lceil \frac{nd_i}{nc_i} \right\rceil$

(2) $ES = PR - \sum_{i=1}^{N} f_i \cdot nd_i = LCM \sum_{i=1}^{N} \left\lceil \frac{nd_i}{nc_i} \right\rceil - \sum_{i=1}^{N} f_i \cdot nd_i$

(3) BD produces no empty slots $\Leftrightarrow nc_i / nd_i , \forall i = 1,..., N$

(4) $ES \leq N \cdot (LCM - 1)$

(5) $PR \geq N \cdot LCM$

<u>*Some problems using Broadcast Disks Method:*</u>
Property 5, tells us that the number of disks N and the LCM of all relative frequencies must be kept small, otherwise the program period will increase significantly making bigger average delays on all data-items. The number of disks is easy to be kept small, but the LCM factor can be kept small if a lot of divisibility conditions between relative frequencies hold.

Property 3 shows that the BD has a very uncomfortable condition for not producing empty slots. It is very difficult to choose relative frequencies because they must satisfy several divisibility conditions for keeping the *LCM* as low as possible and at the same time they must satisfy other divisibility conditions for keeping the number of items divisible with the number of chunks.

Property 4, defines an upper bound of total empty slots in a broadcast program period that again is depended by *N* and *LCM* parameters. Thus, all the above difficulties affect also the number of empty slots.

The most effective selection to solve the above problems is to choose for relative frequencies numbers that are small powers of the same small prime numbers but this is very restrictive.

Other Motivations:
System developers that use existing broadcasting methods cannot success in all conditions and restrictions that methods provide, so they cannot select wisely their parameters.

In most applications, the data-items are categorized and have extra priorities that are application dependent, so their allocation to the disks can be more complicated. For example, we cannot allocate into the same disk multimedia content and text information.

Even worst, if the broadcast systems have to redesign their programs due to large data-item movements (insertions, deletions, updates etc.) very often, the developers have to reconsider the previous subjects and this is not always possible, especially if the available redesign time is crucial for the system's services.

3. THE PROPOSED METHOD

We propose a method for broadcast program creation that improves the efficiency of the broadcast disks based methods and solves the above problems. The method's motives were:

1. Producing program periods with the shortest length possible.
2. Respecting the number of disks and all relative frequencies.
3. Keeping as low as possible the average delays of data-items.
4. Not using chunks or any other splitting policy to the disks.
5. Not having any empty slots into the program period.
6. Not having any restrictions in the selection of parameters.
7. Not having the mentioned problems.
8. Can be applied in all type of distributions of data-items.

3.1 Basic Definitions and Methodology

The proposed method uses multiple disks (d_i) of different sizes (nd_i) and speeds (f_i) as in the broadcast disks technique. The total number of the broadcasted data-items is: $\sum_{i=1}^{N} nd_i = M$. But now, there are not any restrictions. The system designer can select freely all parameters respecting only the conditions: $f_1 > ... > f_N = 1$ and $nd_1 < ... < nd_N$. With this freedom on parameters selection, we accomplished easy the motives (2), (6). In addition, we can easily calibrate any desired distribution of data-items, so motive (8) is also accomplished.

In order to have the minimum possible program length, every data-item that belongs to disk d_i must appear in the broadcast program period exactly f_i times, and there must not be any empty slots. This happens in our method, so we have always a broadcast program period with the shortest length: $PR = \sum_{i=1}^{N} f_i \cdot nd_i$. Thus, motives (1), (5) are accomplished. Our method, does not produce empty slots because does not make use of LCM, chunks or any other splitting policy on the disks. By avoiding their use, we avoid all mentioned divisibility and bounding problems of Section 2, so motives (4), (7) are also accomplished.

The most critical motive is (3), which is completely related with how we must put the data-items and their copies into the broadcast program period. In order to keep low the average delays, we must keep their inter-arrival times constant

Main Strategy: *The Broadcast Disks method always keeps constant the inter-arrival times by sacrificing the period length, which can be much larger than the minimum. But then, as the program period increases, all data-items will have longer average delays. Our strategy is exactly the opposite one: we keep constant the period length (to the minimum possible) and we try to define the inter-arrival times to have the lowest possible variance.*

The most effective way to put a data-item of disk d_i to the broadcast program period is to repeat the data-item f_i times in equally spaced distances, covering the period length. Then the data-item will have the ideal average delay. But, often the relative frequencies do not divide exactly the program period. Therefore, we repeat the data-item into the program every $del_i = \left\lfloor \frac{PR}{f_i} \right\rfloor$ slots, using the floor function. But, it is not easy. During the allocation procedure with these calculations, several data-items may be colliding to the same position, so we must define what to do with the **collisions** and how to minimize them. The following sections present 3 algorithms that construct broadcast programs and deal with the collisions. These algorithms produce the desired program periods which are stored into the array *x[1…PR]*.

3.2 The "Next Position" Algorithm

The first proposed algorithm, which we call "*Next Position Algorithm*", has the following simple strategy: "*It tries to allocate every data-item and its copies into the ideal spaces and if there is a collision it just moves the allocation pointer to the next available position*". Figure 1 depicts its operation.

```
Algorithm NextPosition()
pos = 0
for i = 1 to N
    for j = 1 to nd_i
        { pos++
          for k = 1 to f_i
              { fpos = pos + (k-1) * del_i
                while x[fpos] ≠ ∅ do fpos++
                x[fpos] = data-item j of disk i  }  }
```

Figure 1. First proposed algorithm for program creation

Now, let us see a construction example using the "*Next Position Algorithm*": Suppose that we have N = 5 Disks with relative frequencies f_i = {5, 4, 3, 2, 1} and data-items nd_i = {1, 3, 5, 12, 25}. Then, with the above calculations we will have a program length PR = 81 and expected delays del_i = {16, 20, 27, 40, 81}.

Figure 2 depicts some instances of the construction procedure using the Next Position algorithm. The notation *Di,j* refers to the *j*-th data-item of disk d_i. The gray slots express the single collisions that happen. The dark gray slots express double and triple collisions (collisions that happen into the same slot 2 or 3 times). There are totally 47 collisions in allocation procedure.

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2											D1,1				
	D2,1	D2,2	D2,3								D3,1	D1,1	D3,2							
	D2,1	D2,2	D2,3					D1,1										D3,1	D3,2	
	D2,1	D2,2	D2,3	D1,1																

(a) Allocation of data-items of disks d_1, d_2, d_3 till a collision happens

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2	D3,3	D3,4	D3,5								D1,1				
	D2,1	D2,2	D2,3								D3,1	D1,1	D3,2	D3,3	D3,4	D3,5				
	D2,1	D2,2	D2,3					D1,1										D3,1	D3,2	
D3,3	D2,1	D2,2	D2,3	D1,1	D3,4	D3,5														

(b) Continuing allocation of data-items of disk d_3 where new collisions happens

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2	D3,3	D3,4	D3,5	D4,1	D4,2	D4,3	D4,4	D4,5	D4,6	D4,7	D1,1	D4,8	D4,9	D4,10	
D4,11	D2,1	D2,2	D2,3	D4,12							D3,1	D1,1	D3,2	D3,3	D3,4	D3,5				
	D2,1	D2,2	D2,3					D1,1	D4,1	D4,2	D4,3	D4,4	D4,5	D4,6	D4,7	D4,8	D4,9	D3,1	D3,2	
D3,3	D2,1	D2,2	D2,3	D1,1	D3,4	D3,5	D4,10	D4,11	D4,12											

(c) Allocation of data-items of disk d_4 with their copies

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2	D3,3	D3,4	D3,5	D4,1	D4,2	D4,3	D4,4	D4,5	D4,6	D4,7	D1,1	D4,8	D4,9	D4,10	
D4,11	D2,1	D2,2	D2,3	D4,12	D5,1	D5,2	D5,3	D5,4	D5,5	D5,6	D3,1	D1,1	D3,2	D3,3	D3,4	D3,5	D5,7	D5,8	D5,9	
D5,10	D2,1	D2,2	D2,3	D5,11	D5,12	D5,13	D5,14	D1,1	D4,1	D4,2	D4,3	D4,4	D4,5	D4,6	D4,7	D4,8	D4,9	D3,1	D3,2	
D3,3	D2,1	D2,2	D2,3	D1,1	D3,4	D3,5	D4,10	D4,11	D4,12	D5,15	D5,16	D5,17	D5,18	D5,19	D5,20	D5,21	D5,22	D5,23	D5,24	D5,25

(d) Final constructed broadcast program period

Figure 2. Construction using "Next Position" Algorithm

3.3 The "Padding" Algorithm

The second algorithm, called "*Padding Algorithm*", has the same strategy plus: "*When allocates the data-items of the same disk and their copies, remembers the number of collisions that locally happened (pad variable) and uses it to the next allocations*". This expansion avoids the repeated collisions that may happen to the same position. Figure 3 depicts its operation.

```
Algorithm Padding()
$pos = 0$
for $i = 1$ to $N$
  { $pad = 0$
    for $j = 1$ to $nd_i$
      { $pos$++
        for $k = 1$ to $f_i$
          { $fpos = pos + (k-1) * del_i + pad$
            while $x[fpos] \neq \emptyset$ do
                { $fpos$++
                  if $fpos > PR$ then $fpos = fpos - PR$
                  $pad$++ }
            $x[fpos]$ = data-item $j$ of disk $i$ } } }
```

Figure 3. Second proposed algorithm for program creation

Figure 4 depicts some instances of the construction procedure using the Padding algorithm on the previous example. There are totally 21 collisions in allocation procedure.

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2											D1,1				
	D2,1	D2,2	D2,3								D3,1	D1,1	D3,2							
	D2,1	D2,2	D2,3					D1,1										D3,1		
D3,2	D2,1	D2,2	D2,3	D1,1																

(a) Allocation of data-items of disk d_3 till a collision happens

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2		D3,3					D3,4	D3,5			D1,1				
	D2,1	D2,2	D2,3								D3,1	D1,1	D3,2	D3,3					D3,4	
D3,5	D2,1	D2,2	D2,3					D1,1										D3,1		
D3,2	D2,1	D2,2	D2,3	D1,1	D3,4	D3,5														

(b) Continuing allocation of data-items of disk d_3 where fewer collisions happens

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2		D3,3		D4,1	D4,2	D4,3	D3,4	D3,5	D4,4	D4,5	D1,1	D4,6	D4,7		
D4,8	D2,1	D2,2	D2,3						D4,9	D4,10	D3,1	D1,1	D3,2	D3,3	D4,11	D4,12			D3,4	
D3,5	D2,1	D2,2	D2,3					D1,1	D4,1	D4,2	D4,3			D4,4	D4,5		D4,6	D3,1	D4,7	
D3,2	D2,1	D2,2	D2,3	D1,1	D3,3	D3,4	D3,5	D4,8	D4,9	D4,10					D4,11	D4,12				

(c) Allocation of data-items of disk d_4 with their copies

D1,1	D2,1	D2,2	D2,3	D3,1	D3,2	D5,23	D3,3	D5,24	D4,1	D4,2	D4,3	D3,4	D3,5	D4,4	D4,5	D1,1	D4,6	D4,7	D5,25	
D4,8	D2,1	D2,2	D2,3	D5,1	D5,2	D5,3	D5,4	D5,5	D4,9	D4,10	D3,1	D1,1	D3,2	D3,3	D4,11	D4,12	D5,6	D5,7	D3,4	
D3,5	D2,1	D2,2	D2,3	D5,8	D5,9	D5,10	D5,11	D1,1	D4,1	D4,2	D4,3	D5,12	D5,13	D4,4	D4,5	D5,14	D4,6	D3,1	D4,7	
D3,2	D2,1	D2,2	D2,3	D1,1	D3,3	D3,4	D3,5	D4,8	D4,9	D4,10	D5,15	D5,16	D5,17	D5,18	D4,11	D4,12	D5,19	D5,20	D5,21	D5,22

(d) Final constructed broadcast program period

Figure 4. Construction using "Padding" Algorithm

3.4 The "Minimize Collisions" Algorithm

The third algorithm, called "*Minimize Collisions Algorithm*", has a different strategy: "*Before every allocation of a data-item and its copies to the program period, a function scans all possible positions of the whole structure (data-item and its copies in equally spaced distances del$_i$) and counts the total collisions happened. Then, it returns the position with the minimum number of collisions (w variable). This position is used to the allocation procedure*". This strategy minimizes the collision number and in most experimental cases this number becomes 0 making the program's efficiency ideal. Figure 5 depicts its operation.

```
Algorithm MinimizeCollisions()
for $i = 1$ to $N$
    for $j = 1$ to $nd_i$
      { $w$ = ReturnPositionWithMinimumCollisions($i$)
        for $k = 1$ to $f_i$
          { $fpos = w + (k-1) * del_i$
            while $x[fpos] \neq \emptyset$ do $fpos$++
            $x[fpos]$ = data-item $j$ of disk $i$ } }
Function ReturnPositionWithMinimumCollisions(disk $i$)
$mincol = \propto$ , $minw = 1$
for $w = 1$ to $(del_i + 1)$
  { $col = 0$ , $overflow$ = false
    for $k = 1$ to $f_i$
      { $fpos = w + (k-1) * del_i$
        if $fpos > PR$ then $overflow$ = true
        while $x[fpos] \neq \emptyset$ and $overflow$ = false do
            { $col$++ , $fpos$++
              if $fpos > PR$ then $overflow$ = true } }
    if $overflow$ = false and $mincol > col$ then
      { $mincol = col$ , $minw = w$ } }
return($minw$)
```

Figure 5. Third proposed algorithm for program creation

Figure 6 depicts instances of the construction procedure using the Minimize Collisions algorithm on the previous example. All collisions now are avoided.

D1,1	D2,1	D2,2	D2,3	D3,1		D3,2										D1,1				
	D2,1	D2,2	D2,3								D3,1	D1,1	D3,2							
	D2,1	D2,2	D2,3					D1,1										D3,1		
D3,2	D2,1	D2,2	D2,3	D1,1																

(a) Allocation of data-items of disk d_3 avoiding the collision

D1,1	D2,1	D2,2	D2,3	D3,1		D3,2					D3,3	D3,4	D3,5			D1,1				
	D2,1	D2,2	D2,3								D3,1	D1,1	D3,2					D3,3	D3,4	
D3,5	D2,1	D2,2	D2,3					D1,1										D3,1		
D3,2	D2,1	D2,2	D2,3	D1,1	D3,3	D3,4	D3,5													

(b) Continuing allocation of data-items of disk d_3 avoiding all collisions

D1,1	D2,1	D2,2	D2,3	D3,1	D4,1	D3,2	D4,2		D4,3	D4,4	D3,3	D3,4	D3,5	D4,5	D4,6	D1,1	D4,7		D4,8	
	D2,1	D2,2	D2,3					D4,9	D4,10	D4,11	D3,1	D1,1	D3,2	D4,12				D3,3	D3,4	
D3,5	D2,1	D2,2	D2,3		D4,1		D4,2	D1,1	D4,3	D4,4				D4,5	D4,6		D4,7	D3,1	D4,8	
D3,2	D2,1	D2,2	D2,3	D1,1	D3,3	D3,4	D3,5	D4,9	D4,10	D4,11				D4,12						

(c) Allocation of data-items of disk d_4 with their copies

D1,1	D2,1	D2,2	D2,3	D3,1	D4,1	D3,2	D4,2	D5,1	D4,3	D4,4	D3,3	D3,4	D3,5	D4,5	D4,6	D1,1	D4,7	D5,2	D4,8	
D5,3	D2,1	D2,2	D2,3	D5,4	D5,5	D5,6	D5,7	D4,9	D4,10	D4,11	D3,1	D1,1	D3,2	D4,12	D5,8	D5,9	D5,10	D3,3	D3,4	
D3,5	D2,1	D2,2	D2,3	D5,11	D4,1	D5,12	D4,2	D1,1	D4,3	D4,4	D5,13	D5,14	D5,15	D4,5	D4,6	D5,16	D4,7	D3,1	D4,8	
D3,2	D2,1	D2,2	D2,3	D1,1	D3,3	D3,4	D3,5	D4,9	D4,10	D4,11	D5,17	D5,18	D5,19	D4,12	D5,20	D5,21	D5,22	D5,23	D5,24	D5,25

(d) Final constructed broadcast program period

Figure 6. Construction using "Minimize Collisions" Algorithm

3.5 Properties of the Proposed Algorithms

The following properties and bounds of the proposed method and algorithms, can be proved:

(1) $del_i > nd_i, \forall i = 1,...,N$ **(2)** $del_i > nd_{i-1}, \forall i = 2,...,N$

(3) $del_i \geq del_{i-1}, \forall i = 2,...,N$ **(4)** $del_N = PR$

(5) Total collisions number (col) satisfies the inequalities:

(a) Next-Position:
$$col < \sum_{i=2}^{N-1}\left(\left(f_i - 1\right)\cdot nd_i \cdot \sum_{j=1}^{i-1} nd_j\right)$$

(b) Padding:
$$col < \sum_{i=2}^{N-1}\left(\left(f_i - 1\right)\cdot nd_i \cdot \left(\sum_{j=1}^{i-1} nd_j - 1\right)\right)$$

(c) Minimize Collisions: $col < \sum_{i=2}^{N-1}\left(f_i \cdot \sum_{j=1}^{i-1} nd_j \right)$

From these bounds we derive that Padding is more efficient from Next-Position and Minimize Collisions is the most efficient. Practically, in most experiment cases the total number of collisions is much less than these bounds (actually 0). They can be reached only when the relative frequencies (f_i) are too close and the same holds for the number of data-items (nd_i).

It is also important to note that using any of the 3 proposed algorithms, an allocation of all data-items and their copies always can be found. Therefore, the algorithms always find a solution.

Finally, it can be proved that their worst complexities are: (a) Next-Position: $O\left(\frac{1}{2}\cdot PR^2\right)$, (b) Padding: $O\left(PR^2\right)$, (c) Minimize Collisions: $O\left(\frac{M}{4}\cdot PR^3\right)$. We can conclude that the most efficient algorithm we use, the most computations are required.

4. PERFORMANCE EVALUATION

In this section we present experimental results about the efficiency and the performance of the proposed algorithms in comparison to the BD. To have a fair comparison, we built a simulation environment similar to the model suggested by the authors of the BD in [8]. We extended this model to support our method. Then, we make several experiments using parameter values as in real-world cases. Experiments made using both auto-created data-items that follow specific demand probability distributions and data-items that imported to the simulator from disk files.

During experiments, we vary the parameters values. In order to have a fair comparison of the proposed algorithms with the BD, we selected similar parameter values that used in papers of broadcasting methods. The average response times (average delays) measured in broadcast units (time steps), which define the final performance. The average values were taken after broadcasting 100 times the program period and till client requests were satisfied. We chose the distribution of data-item demand probabilities as *Zipf*. This distribution has a basic parameter θ, which takes values into [0,1]. For θ=0 reduces to a uniform distribution of demand for all data-items. For large values of θ, produces increasingly skewed demand patterns. Also, we calculate the Disk relative frequencies as in *Delta method* [3, 8-10], where d_i {i=1,…,N} is computed relative to the frequency of the slowest disk by the formula: $\frac{rel\_freq(d_i)}{rel\_freq(d_N)} = (N-i)*\Delta+1$. The frequency of the slowest disk is selected $rel\_freq(d_N)$=1.

Varying Theta (θ) Parameter

In this experiment group we chose the following parameter values: M=5000, N=5, Δ=2, θ=0.25-0.95.

The results gave a total number of collisions 0 in all cases except at θ=0.25, where "Next-Position" reports 8090 collisions, "Padding" reports 42 and "Minimize Collisions" reports 0. That happens because when θ=0.25 or below, the nd_i values are close enough to each other, increasing the number of possible collisions. Figure 7 presents the average response times of all methods. The response times in BD are always longer when compared to proposed algorithms. In addition, proposed algorithms have almost the same average response times and that is why their lines in the graph cannot be distinguished. Similar results are reported if we choose different Δ, or different N.

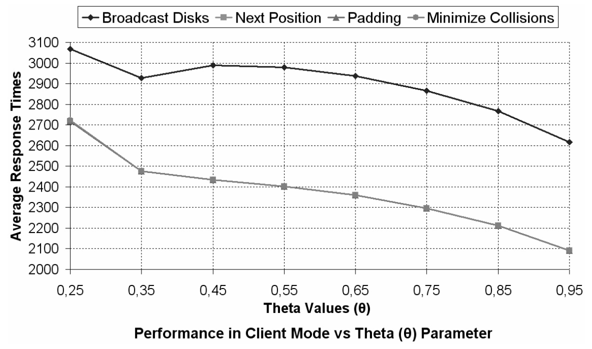


Figure 7. Experimental results varying theta (θ)

Varying Delta (Δ) Parameter

In this experiment group we chose M=5000, N=3, θ=0.95, Δ=1-7.

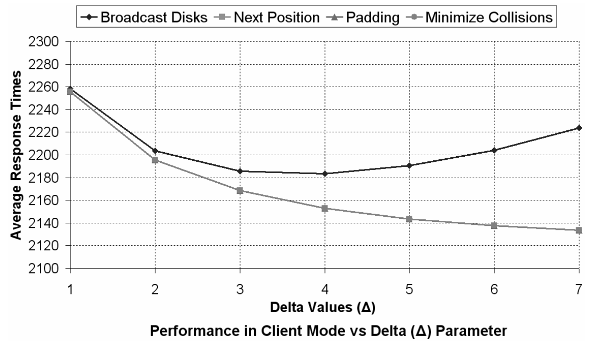


Figure 8. Experimental results varying delta (Δ)

In this experiment we observed that the proposed algorithms, as expected, did not give any collisions. This happened because we had few disks and high theta value, so there was enough space in allocations. Figure 8 presents the final average response times. The response times in BD are always longer when compared to our algorithms. An interesting observation is that BD starts to increase its response times after Δ=4, but the proposed algorithms keep them low. This proves that our method is efficient in any relative frequency values. Similar results will be reported if we choose different θ, or different N.

Varying the number of Disks (N) parameter

In this experiment group we chose: M=5000,Δ=1, θ=0.95, N=1-9.

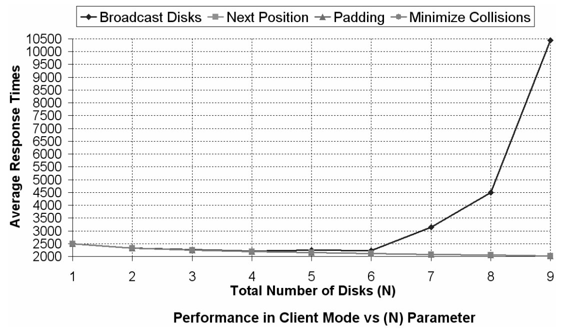
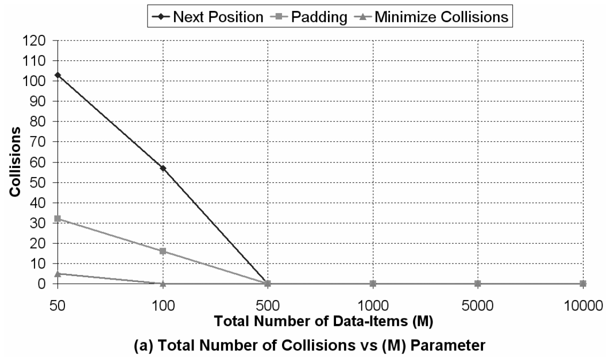


Figure 9. Experimental results varying number of disks (N)

The proposed algorithms, as we expected, did not give collisions again. Figure 9 presents the final average response times using all methods. The response times in BD are longer for $N>4$ than the response times of the proposed algorithms and are increasing significantly for $N>6$. In opposition, the proposed algorithms keep the response time low. This proves that our method is efficient in any number of disks. Similar results reported when we chose different θ, or different Δ.

Varying the number of Data-Items (M) parameter
Here we chose N=5, Δ=2, θ=0.55, M=50-10000.

Figure 10a shows an interesting phenomenon: When we have a small database, the collisions are increased. In addition, we observe an evaluation of the collisions bounds order of the proposed algorithms (see properties of sub-section 3.5). Figure 10b presents the final average response times. The response times in BD are always longer than the proposed algorithms.



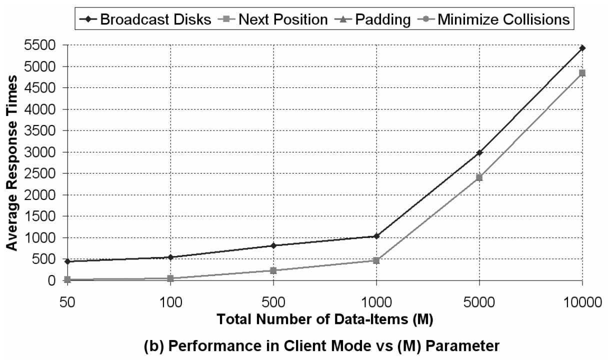


Figure 10. Experimental results varying data-items (*M*)

It is important to note that in experiment groups with noise existence (10% to 75% deviations in client demand probabilities), proposed algorithms reported similar results. Thus, our method is efficient enough in noise existence.

5. CONCLUSIONS

In this paper we presented the problems and drawbacks of the well-known technique of Broadcast Disks. We gave motivations for better broadcasting programs and how important is to construct efficient program periods. To successfully support the demanding nowadays-broadcasting applications, we developed a new method followed by 3 different and simple algorithms, which construct improved and efficient broadcast programs. We presented analytical construction examples in each algorithm and their properties and bounds. We tested the proposed algorithms in simulation environments similar to the suggested by the authors of BD, which we extended to support and test our method. Simulation results show that the proposed algorithms have significant performance gains over the broadcast disks method in all parameters set-ups and situations. In all experiments our method results shorter program period, lower average delays, and is not affected from the use of prime numbers, numbers with no common divisors or any other for frequencies and data-item allocations. The big advantage of the proposed algorithms is that they can easily be applied in automation systems in broadcast servers, they can be easily integrated in hardware components and they can be executed very fast, supporting automatic creation of program periods in the servers, even if the set of data-items is modified rapidly and continuously on time, or even if there are large amounts of insertions, deletions and updates on the data-items set. Also, our method can be successfully applied in multi-broadcasting networks and sensor systems and as a future work we will modify the proposed algorithms to test them in such environments.

REFERENCES

[1] N.H.Vaidya, S.Hameed, Scheduling Data Broadcast In Asymmetric Communication Environments, *ACM/Baltzer Wireless Networks*, vol. 5, pp. 171-182, August 1999.

[2] N.H.Vaidya, S.Hameed, Data Broadcast in Asymmetric Wireless Environments, *in Proceedings of WOSBIS*, November, 1996.

[3] S.Acharya, M.Franklin, S.Zdonik, Dissemination-based Data Delivery Using Broadcast Disks, *IEEE Personal Communications*, vol. 2, no. 6, pp. 50-60, December 1995.

[4] P.Nicopolitidis, G.I.Papadimitriou and A.S.Pomportsis, Using Learning Automata for Adaptive Push-Based Data Broadcasting in Asymmetric Wireless Environments, IEEE Transactions on Vehicular Technology, vol.51, no.6, November 2002, pp. 1652-1660.

[5] P.Nicopolitidis, G.I.Papadimitriou and A.S.Pomportsis, Exploiting Locality of Demand to Improve the Performance of Wireless Data Broadcasting, *IEEE Transactions on Vehicular Technology*, May 2006 issue.

[6] P.Nicopolitidis, G.I.Papadimitriou, M.S.Obaidat and A.S.Pomportsis, Performance Optimization of an Adaptive Wireless Push System in Environments with Locality of Demand, *Computer Communications*, Elsevier, 2005.

[7] T. Bowen, et al, The Datacycle Architecture, *Communications of the ACM*, vol.35, no. 12, pp. 71-81, December 1992.

[8] S.Acharya, R. Alonso, M.Franklin, S.Zdonik, Broadcast Disks: Data Management for Asymmetric Communication Environments, *Technical Report CS-94-43*, 1994.

[9] S.Acharya, M.Franklin, S.Zdonik, Disseminating Updates on Broadcast Disks*, in Proceedings of 2nd VLDB Conference*, September, 1996, pp. 354-365.

[10] S.Acharya, M.Franklin, S.Zdonik, Balancing Push and Pull for Data Broadcast, *in Proceedings of ACM SIGMOD*, May, 1997, pp. 183-194.