Universität Konstanz

FlexSched: A Flexible Data Schedule Generator for Multi-Channel Broadcast Systems

André Seifert
Jen-Jou Hung

Konstanzer Schriften in Mathematik und Informatik
Nr. 211, Oktober 2005

ISSN 1430–3558

Konstanzer Online-Publikations-System (KOPS)
URL: http://www.ub.uni-konstanz.de/kops/volltexte/2006/2231/
URN: http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-22318

© Fachbereich Mathematik und Statistik
© Fachbereich Informatik und Informationswissenschaft
Universität Konstanz
Fach D 188, 78457 Konstanz, Germany
Email: preprints@informatik.uni–konstanz.de
WWW: http://www.informatik.uni–konstanz.de/Schriften/

FlexSched: A Flexible Data Schedule Generator for Multi-Channel Broadcast Systems

André Seifert[1] and Jen-Jou Hung[2]

[1] University of Konstanz
Database Research Group
P.O. BOX D188
78457 Konstanz, Germany
`andre.seifert@uni-konstanz.de`
[2] National Taiwan University of Science and Technology
Department of Information Management
No. 43, Section 4, Keelung Road, Taipei 10672, Taiwan R.O.C.
`jjhung@juno.cs.ntust.edu.tw`

Abstract. This paper is concerned with the issue of building efficient and flexible data broadcast programs for a multiple physical channel environment. The goal of our work has been twofold: (a) to minimize the cost of building the broadcast program and (b) to minimize the average access latency for data requests of mobile clients. To achieve both goals, a highly flexible scheduler, called FlexSched, is proposed, taking a divide-and-conquer approach to solve the data scheduling problem. To determine the optimal allocation of the data to the existing physical channels, FlexSched initially makes a single scan over the vector of access probabilities of the data items and assigns them to the broadcast channels based on the square-root formula of optimal bandwidth usage, i.e., data items are mapped to channels in proportion to the square-root of their access probabilities. For each physical channel a skewed broadcast schedule is then produced by first assigning each data item to its own data group and then iteratively merging neighboring data groups so that the overall average access time of the initial schedule is continously reduced. The merging procedure terminates if there are no more neighboring data groups in the program whose merging would further decrease the overall average access latency. An extensive performance study confirms the effectiveness and efficiency of FlexSched. It is shown that FlexSched significantly outperforms existing data scheduling schemes, while being able to generate the broadcast schedule with an average time complexity of $\mathcal{O}(N \log N)$.

1 Introduction

Wireless data broadcasting has been identified as a powerful and efficient way to deliver timely information to a large number of clients, anytime, anywhere, and anyhow. Wireless data broadcast service providers such as Ambient [1], Microsoft [21], or SkyTel [27] try to capitalize on the emerging business opportunities provided by the new communication technologies and continuously disseminate interesting private and public information such as appointments and meeting requests, stock quotes, flight schedules, traffic news and accident reports, etc. to mobile clients subscribed to their broadcast channels. While the systems' communication protocols and network infrastructures are likely to be dissimilar, their data delivery models are very much alike: a number of distributed resource-rich broadcast servers continuously and repeatedly disseminate data items from a selected set of producers over pre-defined frequency intervals, while mobile clients (e.g., wristwatches, key-chains, PDAs, etc.) passively listen to the broadcast channels until the desired data item comes up. Since available mobile client and network bandwidth resources are scarce and network service subscribers expect the best possible service (i.e., short data access times) at the lowest possible costs (i.e., low subscription fees and low energy expenditure), the broadcast server needs to be concerned with two critical issues when building a broadcast schedule, namely (a) *access efficiency* and (b) *power expenditure*.

The focus of this paper is primarily on the access efficiency issue, i.e., our goal is to minimize the average access latency of the broadcast schedule, which is the average number of time elapsed between the moment the client starts seeking the data item of interest and the moment it gets that item. Unfortunately, due to the serial nature of the broadcast medium, the average access time of a broadcast schedule is directly related to its size and it increases when its size becomes larger. To improve the access efficiency of the broadcast schedule, the following three design guidelines should be followed:

– If the broadcast server has access to multiple physical broadcast channels, the data items to be transfered should be fairly distributed among those channels according to their popularities and the channels' bandwidth capacities.
– If the distribution of the demand probabilities of the data items is non-uniform (which is typically the case in reality [5, 6, 11]), a skewed, rather than a flat broadcast should be generated in which data items appear proportional to the square-root of their popularities.
– The broadcast schedule itself should be constructed in such a way that the distance between consecutive instances of the same data item is constant.

Guided by the above design principles and motivated by weaker than expected performance results of the state-of-the-art scheduling algorithms for skewed data access patterns, we present a new flexible scheduling algorithm, called FlexSched. Similar to previous work [4, 9, 13, 17, 23, 34], FlexSched assumes the existence of multiple disjoint broadcast channels since such an architecture allows for better configurability, fault tolerance, and scalability [24, 33] and it also focuses on data indexing, rather than on the objective of effective and efficient data scheduling alone. It does so by addressing the problem of generating latency-minimal broadcast schedules under the constraint that consecutive instances of each data item are scheduled with a fixed period. Forcing the broadcast server to produce exclusively so-called *perfectly periodic schedules* enables mobile clients to significantly reduce their power expenditures by allowing them to compute the time their desired data items are broadcast next on the basis of "obsolete" index information. Perfect periodicity, however, is also a very desirable property from the server's point of view since it may allow the server to reuse indexes across broadcast cycles.

Contributions. In particular, the paper makes the following contributions:

– We address the issue of generating access latency-minimal broadcast schedules by relying on two orthogonal concepts: (a) skewed allocation of the data to the set of existing disjoint broadcast channels and (b) generation of skewed data schedules for each broadcast channel, i.e., more frequently accessed data items are likely be broadcast more often per broadcast cycle within the respective channel compared to less popular ones.

– We perform an extensive experimental evaluation of the FlexSched scheduling algorithm and compare the obtained results with existing state-of-the-art scheduling algorithms.
– We show that FlexSched either outperforms or performs at least as well as the Dichotomic algorithm [4] which is based on dynamic programming and finds optimal solutions for the multi-channel flat scheduling case.
– We indicate that FlexSched has a low average runtime complexity ($\mathcal{O}(N \log N)$) which is by a factor of $\mathcal{O}(M)$ lower than that of the Dichotomic algorithm, where N is the number of data items and M is the number of broadcast channels.
– We also provide evidence that FlexSched's efficiency can be even further improved by a factor of M, where M is again the number of broadcast channels, by parallelizing the construction process of the schedule on multi-processor machines. Other algorithms, such as the Dichotomic or VF$^{\mathrm{K}}$ schemes [4, 23], do not possess this potential.

Organization. The remainder of the paper is organized as follows: In the next section related work will be reviewed, followed by the definition of some notations and the mathematical formulation of the theoretical foundation behind FlexSched's structure in Section 3. Section 4 describes the FlexSched scheduling algorithm in detail and experimental results follow in Section 5. A summary of the paper is presented in Section 6.

2 Related Work

Research issues in the field of data broadcasting have received much attention lately and were first introduced in the field of teletext systems by Ammar and Wang [2, 3]. Existing solutions to the problem can roughly be categorized into those that attempt to minimize the *access latency* [2–4, 8, 9, 13, 17, 23, 31, 34] and others that try to minimize the *tuning time* [7, 12, 14, 15, 19, 26, 32], i.e., the amount of time a mobile client stays active to receive the data of interest, by interleaving index information with the data broadcast. While many studies, like the ones mentioned above including ours, address the issue of data scheduling and indexing separately, some others, however, examine the problem from a more integrated perspective [10, 18, 20, 24]. Since our primary objective is to minimize the expected access latency of the mobile clients, we confine ourselves here to summarizing the findings related to this topic.

The problem of finding latency-minimal broadcast schedules has been extensively discussed by both the networking and the database communities, but from slightly different perspectives. While the network community focuses on finding infinite broadcast schedules for single and multiple broadcast channels [2, 3, 8, 31], the database community seeks for perfectly periodic broadcast schedules for single and multiple broadcast channels which can be efficiently indexed and additionally allow clients to know a priori when exactly the data of interest is scheduled [4, 9, 13, 17, 23, 34]. Nearly all proposals from the network community as well as our proposal adopt the square-root rule [2] when addressing the issue of designing optimal cyclic broadcast schedules. In reality, however, it may not be possible to satisfy all the conditions imposed by the square-root rule at the same time (see [30] for more details). To overcome this difficulty, the researchers from the network community follow the approach of relaxing on the periodicity property of the schedule while maintaining the *frequency property*, i.e., the broadcast frequency of a data item is chosen to be proportional to the square-root of its demand probability.

The database community as well as ourselves, on the other hand, explores the other alternative, namely insisting on maintaining the *periodicity property* while giving up on the general satisfaction of the frequency property. To produce latency-minimal broadcast schedules without violating the periodicity constraint in a multi-channel broadcast environment, the generation of *flat schedules* has been proposed [4, 9, 13, 17, 23, 34]. Here, the periodicity constraint is trivially satisfied, but the allocation of the data items to the broadcast channels becomes a challenge. This problem is solved by allocating the data items to the available channels using a *skewed distribution* [4, 9, 13, 17, 23, 34] with the Dichotomic algorithm [4] being the best algorithm proposed so far. It finds optimal solutions for the multi-channel uniform length case in $\mathcal{O}(N\,M \log N)$ time. The problem with flat schedules in a multi-channel broadcast environment, however, is that all data items assigned to the same physical broadcast channel are disseminated with the same frequency. Obviously, if the demand probability of the data items within the same physical channel is different, the above approaches will fail to generate optimal periodic schedules. FlexSched rectifies this problem by using both *skewed data allocation* and *skewed data scheduling* to produce perfectly periodic schedules.

3 Preliminaries

3.1 Notations and Assumptions

Let a set $D = \{d_1, d_2, \ldots, d_n\}$ of N data items represent (a pre-selected subset of) a database to be broadcast over a set $C = \{c_1, c_2, \ldots, c_m\}$ of M disjoint physical broadcast channels. Each data item d_i has a demand probability p_i (with $0 < p_i \leqslant 1$) to be accessed by clients and has a length l_i which is assumed to be the unit length, i.e., $l_i = 1$. Clearly, $\sum_{i=1}^{N} p_i = 1$. Note that the demand probability is obtained as an average over all clients served by the server and there are several feedback-based methods to estimate it [22, 25, 28, 35]. Each physical broadcast channel c_j has a bandwidth capacity bc_j and the total bandwidth of all channels is given by $BC = \sum_{j=1}^{M} bc_j$. Obviously, to exploit the available bandwidth capacity of each physical channel, the set of data items has to be partitioned (somehow) into a set $G = \{g_1, g_2, \ldots, g_m\}$ of M data groups and each data group g_j is then assigned to its associated broadcast channel c_j. The cardinality of data group g_j, i.e., the number of data items collected in group g_j, is denoted by $|g_j|$. Since the data items in data group g_j are cyclically broadcast according to a *skewed schedule*, the set of data items assigned to group g_j is further segmented into a set $G_j = \{g_{j,1}, g_{j,2}, \ldots, g_{j,k}\}$ of K data groups. Each data group

$g_{j,k}$ is assigned an integer-valued broadcast frequency $bf_{j,k}^{int}$ which represents the number of instances of data items belonging to group $g_{j,k}$ being disseminated in a minor broadcast cycle (MIBC). The broadcast schedule of each physical channel c_j consists of a major broadcast cycle (MBC) which, in turn, is divided into a sequence of λ_j equal-sized MIBCs (with $\lambda_j \geqslant 1$) of size $S_j = \sum_{k=1}^{|G_j|} bf_{j,k}^{int}$, where $|G_j|$ denotes the cardinality of G_j. Each MIBC is further subdivided into a sequence of data buckets $B = \langle B_1, B_2, \ldots, B_n \rangle$ which are containers for the scheduled data items and there is no index data interleaved with the data items. Notice that a data bucket is the smallest logical access unit of the broadcast schedule and its size is a multiple of the size of a network packet. The cardinality of a data group $g_{j,k}$ is denoted by $|g_{j,k}|$, while the total number of MIBCs in an MBC is given by:

$$\lambda_j = \max_{\forall g_{j,k} \in G_j} \frac{|g_{j,k}|}{bf_{j,k}^{int}} \tag{1}$$

For reasons of simplicity, we assume that all mobile clients have complete a priori knowledge of the broadcast schedule of each physical channel c_j. We also assume that clients may be able to listen to multiple physical channels simultaneously and may also perform instantaneous hopping from one physical channel to another.

3.2 Theoretical Foundation

To achieve optimal access latencies in a multi-channel broadcast environment, the following four scheduling conditions need to be satisfied:

- To obtain an optimal solution for partitioning a set D of data items into M data groups, the items should be sorted descendingly according to their access probabilities and should be assigned to the M groups in consecutive order [34], i.e., for all data items $d_i \in g_k$ and all $d_j \in g_l$ either the condition $p_i \leqslant p_j$ or $p_i \geqslant p_j$ holds.
- The partitioning of N data items into a set $G = \{g_1, g_2, \ldots, g_m\}$ of M groups leads to a segmentation with the following generic layout:

$$\underbrace{d_1, \ldots, d_{b_1}}_{g_1}, \underbrace{d_{b_1+1}, \ldots, d_{b_2}}_{g_2}, \ldots, \underbrace{d_{b_{m-2}+1}, \ldots, d_n}_{g_m},$$

  where b_m denotes the index of the last data item that belongs to group g_m. According to the square-root rule for optimal bandwidth allocation [2], a partitioning of N data items into M groups is optimal if the inequality

$$\frac{bc_j \cdot \sum_{i=1}^{N} \sqrt{p_i}}{BC} \leqslant \sum_{i=d_{b_{j-1}+1}}^{d_{b_j}} \sqrt{p_i} \tag{2}$$

  holds for all groups $g_j \in G$ along with their associated broadcast channels and the removal of any last data item d_{b_j} from its corresponding group g_j would violate the above inequality. That is, the procedure of assigning unallocated, decreasingly ordered data items to a group $g_j \in G$ is to be finished whenever inequality (2) is fulfilled for the first time.

- To obtain an optimal schedule that provides minimal data access latency for any physical broadcast channel c_j, each data item d_i should be allocated a fraction f_i of c_j's broadcast bandwidth bc_j that is given by:

$$f_i = \frac{\sqrt{p_i}}{\sum_{k=d_{b_{j-1}+1}}^{b_j} \sqrt{p_k}}, \tag{3}$$

  i.e., broadcast bandwidth should be allocated to data items in proportion to the square-root of their access probabilities.
- Last not least, consecutive instances of the same data item d_i disseminated in broadcast channel c_j should be spaced equally apart [16, 29] with spacing s_i being equal to $1/f_i$.

While the first two conditions can be easily fulfilled by any broadcast scheduler, the latter two requirements may not always be realizable in practice [30]. E.g., this is the case when the square-root rule requires instances of distinct data items to be scheduled at the same time. As scheduling collisions are not permitted in real schedules, they disallow direct enforcement of the equal-spacing assumption of the square-root rule, thus resulting in producing only near-optimal periodic schedules. Apart from that, however, the above specified scheduling conditions present a valuable set of theoretical design guidelines for evolving at least close-to-optimal multi-channel broadcast schedules. We conclude this preliminary section by summarizing the notations introduced above and those to follow in Table 1.

Symbol	Description
AT	access time averaged over all data items being broadcast
AT^{opt}	lower bound on the achievable overall average access time
B	sequence of data buckets
B_i	i-th data bucket in B
BC	total bandwidth capacity of all physical broadcast channels in C
$b_{m-1}+1$	lowest index value (infimum) among the data items belonging to data group g_m
b_m	largest index value (supremum) among the data items belonging to data group g_m
bc_j	bandwidth capacity of the j-th physical broadcast channel
$bf_{j,k}^{int}$	integer-valued broadcast frequency of the k-th data group of G_j
$bf_{j,k}^{opt}$	optimal relative broadcast frequency of the k-th data group of G_j
bf_i^{opt}	optimal relative broadcast frequency of the i-th data item/group
C	set of physical broadcast channels
c_j	j-th physical broadcast channel

continued on next page

Symbol	Description
D	set of data items (database)
d_i	i-th data item of database D
δ_i	difference between bf_i^{opt} and bf_i^{int} of data group g_i
f_i	fraction of the network bandwidth of a physical broadcast channel to be allocated to data item d_i
G	set of data groups
G_j	set of data groups associated with broadcast channel c_j
$\|G_j\|$	cardinality of G_j
$\|g_j\|$	cardinality of the j-th data group of G
$\|g_{j,k}\|$	cardinality of the k-th data group of G_j
g_j	j-th data group of G
$g_{j,k}$	k-th data group of G_j
K	number of data groups in G_j
L	number of data groups in R_j
λ_j	number of MIBCs contained in an MBC of physical broadcast channel c_j
l_i	length of the i-th data item
M	number of groups and channels in G and C, respectively
N	number of data items in D
p_i	demand probability of the i-th data item
R_j	set of groups associated with broadcast channel c_j
S_j	MIBC size of broadcast channel c_j in terms of data items
s_i	average spacing between two consecutive instances of the i-th data item
$s_{i,j}$	spacing between the j-th and $j+1$-th instance of data item d_i
w_i	average wait time encountered by a client to receive the i-th data item
Z_j	sum of the product of the demand probability p_i and the average spacing s_i of each data item belonging to group G_j

Table 1. Summary of notations.

4 The FlexSched Scheduling Algorithm

4.1 Analysis of Average Access Time

The FlexSched scheduling algorithm is designed to enable mobile clients to retrieve the disseminated data efficiently, i.e., with low access latency. The access latency of a requested data item d_i is equal to the time elapsed between the moment the client starts seeking for a data item in the broadcast stream until the moment it gets that item. The overall average access time, denoted AT, is defined as the average waiting time encountered by a mobile client averaged over all data items. More precisely,

$$AT = \sum_{i=1}^{N} p_i \cdot w_i, \tag{4}$$

where N denotes the number of distinct data items which are disseminated, p_i represents the probability of a data item d_i being requested by a mobile client, and w_i denotes the average wait time encountered by a mobile client needing to inspect data item d_i. Recall that the average access time can be minimized when all instances of a data item are equally spaced apart, i.e., the periodicity property is not violated. The spacing between two instances of data item d_i is defined as the time elapsed between the transmission of the first and second instance of data item d_i. Let $s_{i,j}$ denote the spacing between the j-th instance of data item d_i and the next instance of the same data item with $1 \leqslant j \leqslant bf_{j,k}^{int}$. Notice that after the $bf_{j,k}^{int}$-th instance of data item d_i is disseminated, the next instance of d_i broadcast is again the first instance of d_i in its next broadcast cycle. Since we do not allow different spacings between consecutive instances of the same data item, the average spacing s_i denotes the spacing for data item d_i. That is, $s_{i,j} = s_i$ for each j with $1 \leqslant j \leqslant bf_{j,k}^{int}$. If we assume that a client is equally likely to request data item d_i at any instance of time, then its average access latency $w_i = s_i/2$. Therefore, the overall average access time is given by:

$$AT = \frac{1}{2}\sum_{i=1}^{N} p_i \cdot s_i. \tag{5}$$

4.2 The Algorithm in Steps

To present the FlexSched scheduling scheme, we take a top-down approach and start by depicting an abstract overview of our procedure by which the broadcast program is constructed. Figure 1 gives a low-level abstraction of the FlexSched algorithm and all of its individual steps will be discussed hereafter.

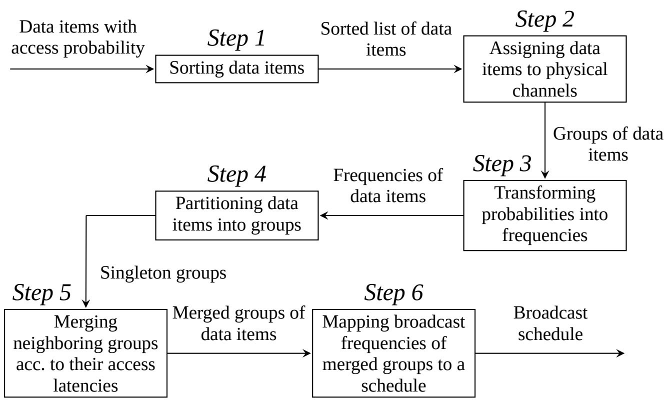
Data items with access probability
Step 1
Sorting data items
Sorted list of data items
Step 2
Assigning data items to physical channels
Groups of data items
Step 3
Transforming probabilities into frequencies
Frequencies of data items
Step 4
Partitioning data items into groups
Singleton groups
Step 5
Merging neighboring groups acc. to their access latencies
Merged groups of data items
Step 6
Mapping broadcast frequencies of merged groups to a schedule
Broadcast schedule

Fig. 1. Building the broadcast schedule.

Steps 1 and 2. FlexSched initially uses an $\mathcal{O}(N \log N)$ sorting algorithm such as Quicksort in order to put the set $D = \{d_1, d_2, \ldots, d_n\}$ of N data items in a descending order of their demand probabilities (Step 1). The resulting sorted sequence $S = \langle d_1, d_2, \ldots, d_n \rangle$ of N data items is then partitioned into an ordered set $G = \{g_1, g_2, \ldots, g_m\}$ of M groups, whereby each group g_j with $1 \leqslant j \leqslant m$ is associated with a physical broadcast channel c_j with bandwidth capacity bc_j, by applying Algorithm 1. The output of Algorithm 1 is a segmentation

$$\underbrace{d_1, \ldots, d_{b_1}}_{g_1}, \underbrace{d_{b_1+1}, \ldots, d_{b_2}}_{g_2}, \ldots, \underbrace{d_{b_{m-1}+1}, \ldots, d_n}_{g_m}$$

with $p_{b_i} \geqslant p_{b_j}$ whenever $b_i < b_j$ (Step 2).

Algorithm 1: Procedure to assign data items to physical broadcast channels

```
1  $BC \longleftarrow \sum_{j=1}^{M} bc_j$; // determine overall available bandwidth capacity
2  $j \longleftarrow 1$; // initialize data group index to 1
3  $bc_j^{alloc} \longleftarrow 0$; // set the amount of allocated bandwidth of broadcast channel $c_j$ to 0
4  foreach data item $d_i$ in sequence $S$ in descending order do
5      if $bc_j^{alloc} < \frac{bc_j}{BC}$ then
6          $bc_j^{alloc} \longleftarrow bc_j^{alloc} + \frac{\sqrt{p_i}}{\sum_{j=1}^{N} \sqrt{p_j}}$; // allocate channel bandwidth for data item $d_i$
7          assign $d_i$ to data group $g_j$
8      else
9          $j \longleftarrow j + 1$; // increment data group index by 1
10         $bc_j^{alloc} \longleftarrow 0$ // reinitialize bandwidth allocation variable with 0
```

Steps 3 and 4. According to the square-root rule, the minimum overall data access latency is achieved when the spacing for each data item d_i is inversely proportional to $\sqrt{p_i}$. This implies that, for optimal performance, a data item $d_i \in g_j$ should be disseminated with a relative broadcast frequency bf_i^{opt} being proportional to $\sqrt{p_i}/\sqrt{p_{b_{j-1}+1}}$, where $p_{b_{j-1}+1}$ is the demand probability of data item $d_{b_{j-1}+1}$ that has the highest demand probability among the data items of group g_j. This observation is exploited by FlexSched and in Step 3 of the algorithm, it calculates for each group $g_j \in G$ and each data item $d_i \in g_j$ with $b_{j-1}+1 < i < b_j$ the item's relative optimal broadcast frequency bf_i^{opt} w.r.t. the broadcast frequency of data item $d_{b_{j-1}+1}$. That is, each data item $d_i \in g_j$ is assigned an optimal relative broadcast frequency bf_i^{opt} with $0 < bf_i^{opt} \leqslant 1$. Thereafter, FlexSched once again partitions the data items $d_{b_{j-1}+1}, d_{b_{j-1}+2}, \ldots, d_{b_j}$ collected in each group $g_j \in G$ into an ordered set $G_j = \{g_{j,1}, g_{j,2}, \ldots, g_{j,k}\}$ of K singleton groups, where K denotes the cardinality of G_j (Step 4). That is, each data item $d_i \in g_j$ is assigned to its own group $g_{j,k}$ with $1 \leqslant k \leqslant |g_j|$ and the group's initial broadcast frequency $bf_{j,k}^{int}$ is set to 1. Additionally, FlexSched keeps track of $g_{j,k}$'s optimal relative broadcast frequency $bf_{j,k}^{opt}$ and its demand probability $p_{j,k}$ which are equal to d_i's optimal relative broadcast frequency bf_i^{opt} and demand probability p_i, respectively. We follow the approach of initially assigning to each singleton group $g_{j,k} \in G_j$

an integer-valued broadcast frequency of 1 (even if that value might be orders of magnitude higher than its optimal one) since it allows for instant generation of a perfectly periodic broadcast schedule. More importantly, however, by using such an opportunistic assignment approach, FlexSched is able to produce an initial “seed” schedule upon which further refinements can be made.

Step 5. Obviously, the initial seed schedule turns automatically into the final schedule when the demand probabilities of the data items are uniformly distributed over D. For *skewed demand probabilities*, however, the initial seed schedule performs poorly and therefore, needs to be improved. The reason the initial schedule produces less than optimal performance results under skewed access conditions is related to the fact that there is typically a large gap between a group’s optimal broadcast frequency and its nearest integer-valued broadcast frequency. That is, the probability of data group $g_{j,k} \in G_j$ having an associated integer-valued broadcast frequency $bf_{j,k}^{int}$ being equal to its optimal broadcast frequency $bf_{j,k}^{opt}$ is usually small. This observation motivates us to use a *group merging approach* to solve the data scheduling problem since it helps reducing the discrepancy between a group’s optimal relative broadcast frequency and its nearest integer-valued one in a simple and direct way. A straightforward approach would be to merge a pair of adjacent data groups whenever the difference between their optimal relative broadcast frequencies and their nearest integer-valued broadcast frequencies decreases by merging them. However, using such a heuristic may lead to suboptimal merging decisions as the following example shows.

Example 1. Let δ_i denote the absolute difference between the optimal relative broadcast frequency bf_i^{opt} and the nearest integer-valued broadcast frequency bf_i^{int} of data group g_i and let $bf_{i,j}^{opt}$ denote the sum of the optimal relative broadcast frequencies bf_i^{opt} and bf_j^{opt} of data groups g_i and g_j, respectively. Further, the nearest integer-valued broadcast frequency of $bf_{i,j}^{opt}$ is represented by $bf_{i,j}^{int}$ and the absolute difference between $bf_{i,j}^{opt}$ and $bf_{i,j}^{int}$ is denoted by $\delta_{i,j}$. Now suppose the broadcast program consists of a set $D = \{d_1, d_2, d_3, \text{ and } d_4\}$ of 4 data items with the associated demand probabilities of $p_1 = 0.4$, $p_2 = 0.3$, $p_3 = 0.2$, and $p_4 = 0.1$, respectively. Given those demand probabilities, the optimal relative broadcast frequencies of the data items are $bf_1^{opt} = 1.0$, $bf_2^{opt} = 0.87$, $bf_3^{opt} = 0.71$, and, $bf_4^{opt} = 0.5$, respectively, and their nearest integer-valued broadcast frequencies are obviously 1. At the beginning of the merging procedure, the broadcast schedule would consist of a set $G = \{g_1, g_2, g_3, \text{ and } g_4\}$ of 4 data groups with the data items 1–4 associated to them. According to the merge criteria specified above, the algorithm would first merge groups g_3 and g_4 into data group $g_{3,4}$ since $\delta_{3,4} < \delta_3 + \delta_4$ $(0.21 < 0.29 + 0.50)$. This merge step would reduce the average access time of the initial schedule from 2 to 1.95 data items. In the second iteration of the merge loop, groups g_2 and $g_{3,4}$ would then be grouped together to form group $g_{2,3,4}$ because $\delta_{2,3,4} < \delta_2 + \delta_{3,4}$ $(0.08 < 0.13 + 0.21)$. It is easy to see that the last merge step violates the equal-spacing assumption since $|g_{2,3,4}|$ is 3 and $bf_{2,3,4}^{int}$ is 2. As a side effect, the average access time of the schedule increases to 2.1 data items, which is about 10% higher than before.

Based on the insights gained from Example 1, FlexSched abstains from using a broadcast frequency-based approach for making merging decisions. It rather uses an *access* *latency-based approach* to decide whether to merge a pair of neighboring data groups or not. More specifically, FlexSched verifies for each pair $(g_{j,i}, g_{j,j})$ of neighboring data groups of G_j whether merging the pair would improve the average access latency of the broadcast schedule. If so, the pair qualifies for merging; otherwise not. In contrast to the heuristic method given above, FlexSched always decides on the basis of *two pairs* of neighboring groups (with one group being element of both pairs) whether to merge one of them or better to leave them as they are. That is, whenever FlexSched evaluates the impact of merging neighboring groups $g_{j,i}$ and $g_{j,j}$ on the overall system performance, it also analyzes the effect of merging adjacent groups $g_{j,j}$ and $g_{j,k}$. The intuition behind such a strategy is to prevent the scheduling algorithm from merging the pair $(g_{j,i}, g_{j,j})$ even though the performance benefit of putting the pair $(g_{j,j}, g_{j,k})$ together would be even greater.

Algorithm 2: Merging procedure of the FlexSched scheduling algorithm

```
1  $Z_j \longleftarrow 0$; $S_j \longleftarrow 0$; // initialize globally scoped variables $Z_j$ and $S_j$ with 0
2  foreach singleton group $g_{j,k}$ of set $G_j$ do
3      $bf_{j,k}^{opt} \longleftarrow \frac{\sqrt{p_{j,k}}}{\sqrt{p_{j,b_{j-1}+1}}}$; // calculate opt. relative broadcast frequency of group $g_{j,k}$
4      $bf_{j,k}^{int} \longleftarrow 1$; // set integer-valued broadcast frequency of group $g_{j,k}$ to 1
5      $s_{j,k} \longleftarrow$ spacing($|g_{j,k}|, bf_{j,k}^{int}$); /* assign the inter-arrival time between two instances
           of the same data item of group $g_{j,k}$ to $s_{j,k}$                                    */
6      $Z_j \longleftarrow Z_j + s_{j,k} \cdot p_{j,k}$; // add the product of $s_{j,k}$ and $p_{j,k}$ to $Z_j$
7      $S_j \longleftarrow S_j + 1$ // increment MIBC size by 1
8  repeat
9      for $k \longleftarrow 4; k \leqslant |G_j|; k \longleftarrow k+2$ do
10         $bf_{k-2,k-1}^{int} \longleftarrow$ round($bf_{k-2}^{opt} + bf_{k-1}^{opt}$);
11         $bf_{k-1,k}^{int} \longleftarrow$ round($bf_{k-1}^{opt} + bf_{k}^{opt}$);
12         $\Delta_{k-2,k-1} \longleftarrow$ deltaAT($bf_{k-2}^{int}, bf_{k-1}^{int}, bf_{k-2,k-1}^{int}, |g_{j,k-2}|, |g_{j,k-1}|, p_{k-2}$,
           $p_{k-1}$);
13         $\Delta_{k-1,k} \longleftarrow$ deltaAT($bf_{k-1}^{int}, bf_{k}^{int}, bf_{k-1,k}^{int}, |g_{j,k-1}|, |g_{j,k}|, p_{k-1}, p_k$);
14         if evalMergePair($\Delta_{k-2,k-1}, \Delta_{k-1,k}, |g_{j,k-2}| + |g_{j,k-1}|, bf_{k-2,k-1}^{int}$) then
15             $p_{j,k-2} \longleftarrow p_{j,k-2} + p_{j,k-1}$; // calculate access probability of group $g_{j,k-2}$
16             $bf_{k-2}^{opt} \longleftarrow bf_{k-2}^{opt} + bf_{k-1}^{opt}$; /* determine opt. relative broadcast frequency
                   of group $g_{j,k-2}$                                                   */
17             $Z_j \longleftarrow Z_{k-2,k-1}$; // update $Z_j$ after merging groups $g_{j,k-2}$ and $g_{j,k-1}$
18             $S_j \longleftarrow S_{k-2,k-1}$ // update $S_j$ after merging groups $g_{j,k-2}$ and $g_{j,k-1}$
19         if evalMergePair($\Delta_{k-1,k}, \Delta_{k-2,k-1}, |g_{j,k-1}| + |g_{j,k}|, bf_{k-1,k}^{int}$) then
20             $p_{j,k-1} \longleftarrow p_{j,k-1} + p_{j,k}$; // calculate access probability of group $g_{j,k-1}$
21             $bf_{k-1}^{opt} \longleftarrow bf_{k-1}^{opt} + bf_{k}^{opt}$; /* determine opt. relative broadcast frequency of
                   group $g_{j,k-1}$                                                      */
22             $Z_j \longleftarrow Z_{k-1,k}$; // update $Z_j$ after merging groups $g_{j,k-1}$ and $g_{j,k}$
23             $S_j \longleftarrow S_{k-1,k}$; // update $S_j$ after merging groups $g_{j,k-1}$ and $g_{j,k}$
24             $k \longleftarrow k + 1$ // increment data group index by 1
25 until (no pair of adjacent data groups has been merged during the last loop iteration)
```

In what follows, we will describe our merging algorithm in more detail and we refer the reader to Algorithms 2 and 3 for the complete pseudocode of the merge procedure. The merging algorithm is divided into two main steps: (a) initiation step and (b) merge step. In the *initiation step* (lines 1–7), the algorithm initializes the global variables S_j and Z_j whose product, if divided by 2, corresponds to the average access latency of the broadcast schedule. At this initial stage of the algorithm, FlexSched also determines for each group $g_{j,k} \in G_j$ its optimal relative broadcast frequency $bf^{opt}_{j,k}$ and it sets the respective group's integer-valued broadcast frequency $bf^{int}_{j,k}$ to 1.

Algorithm 3: Auxiliary functions of the merging procedure

```
/* function which determines the inter-arrival time between two consecutive instances of
   the same data item of a group $g_{j,k}$ */
function spacing($|g_{j,k}|, bf^{int}_{j,k}$)
begin
    return $\frac{|g_{j,k}|}{bf^{int}_{j,k}}$
end
/* function which calculates the difference in the average access latency of a broadcast
   schedule before and after merging data groups $g_{j,k-1}$ and $g_{j,k}$ */
function deltaAT($bf^{int}_{k-1}, bf^{int}_{k}, bf^{int}_{k-1,k}, |g_{j,k-1}|, |g_{j,k}|, p_{k-1}, p_k$)
begin
    $s_{k-1} \longleftarrow$ spacing($|g_{j,k-1}|, bf^{int}_{k-1}$);
    $s_k \longleftarrow$ spacing($|g_{j,k}|, bf^{int}_{k}$);
    $s_{k-1,k} \longleftarrow$ spacing($(|g_{j,k-1}| + |g_{j,k}|), bf^{int}_{k-1,k}$);
    $Z_{k-1,k} \longleftarrow Z_j - (s_{k-1} \cdot p_{k-1} + s_k \cdot p_k) + s_{k-1,k} \cdot (p_{k-1} + p_k)$;
    $S_{k-1,k} \longleftarrow S_j - (bf^{int}_{k-1} + bf^{int}_{k}) + bf^{int}_{k-1,k}$;
    $AT \longleftarrow (S_j \cdot Z_j)/2$;
    $AT_{k-1,k} \longleftarrow (S_{k-1,k} \cdot Z_{k-1,k})/2$;
    return $AT - AT_{k-1,k}$
end
// function to evaluate and, if judged beneficial, to merge data groups $g_{j,i}$ and $g_{j,j}$
function evalMergePair($\Delta_{i,j}, \Delta_{j,k}, |g_{j,i+j}|, bf^{int}_{j,i+j}$)
begin
    if $\Delta_{i,j} > 0$ and $\Delta_{i,j} \geqslant \Delta_{j,k}$ and $(|g_{j,i+j}| \mod bf^{int}_{j,i+j}) = 0$ then
        merge group $g_{j,i}$ with $g_{j,j}$ by adding each item $d_i \in g_{j,j}$ to $g_{j,i}$;
        delete group $g_{j,j}$;
        return true // merging of groups $g_{j,i}$ and $g_{j,j}$ was successful
    else
        return false // merging of groups $g_{j,i}$ and $g_{j,j}$ was not successful
end
```

Hereafter, the *merge step* is performed (lines 8–25). The merge procedure is implemented as a conditional loop which is active as long as at least one pair of adjacent data groups is merged during an entire merge loop iteration. To decide on merging data groups, FlexSched considers triples $(g_{j,k-2}, g_{j,k-1}, g_{j,k})$ of successive data groups.

Clearly, each such triple contains two pairs of adjacent data groups $(g_{j,k-2}, g_{j,k-1})$ and $(g_{j,k-1}, g_{j,k})$, and, therefore, each such pair is regarded as a merging candidate by FlexSched. For each of the two merging candidate pairs $(g_{j,k-2}, g_{j,k-1})$ and $(g_{j,k-1}, g_{j,k})$, FlexSched then verifies whether merging them has a favorable, unfavorable, or even no effect at all on the overall system performance.

To do that, FlexSched performs a *what-if analysis*. In lines 10–24 of the algorithm, the what-if scenario of merging groups $(g_{j,k-2}, g_{j,k-1})$ and $(g_{j,k-1},\ g_{j,k})$, respectively, is explored and its effect on various performance-critical scheduling parameters is quantified. More specifically, for each pair of adjacent data groups $(g_{j,k-2}, g_{j,k-1})$ and $(g_{j,k-1},\ g_{j,k})$, FlexSched first computes their optimal relative broadcast frequencies $bf^{int}_{k-2,k-1}$ and $bf^{int}_{k-1,k}$ (lines 10–11), and determines then the difference between the overall average access time of the schedule before and after merging groups $(g_{j,k-2}, g_{j,k-1})$ and $(g_{j,k-1},\ g_{j,k})$, respectively (lines 12–13). The latter operation is carried out by the function *deltaAT()* being presented in Algorithm 3. Based on these accumulated data, FlexSched then decides by using the auxiliary function *evalMergePair()* whether to merge data groups $(g_{j,k-2},\ g_{j,k-1})$ and $(g_{j,k-1},\ g_{j,k})$, respectively, or better keep them all split apart. FlexSched will decide to merge the pair $(g_{j,k-2},\ g_{j,k-1})$ (and $(g_{j,k-1},\ g_{j,k})$, respectively), if all of the following conditions hold (see also line 18 of Algorithm 3):

– the overall average access time of the broadcast schedule decreases by merging data groups $g_{j,k-2}$ and $g_{j,k-1}$ ($g_{j,k-1}$ and $g_{j,k}$),
– the reduction in average access latency is larger for merging groups $g_{j,k-2}$ and $g_{j,k-1}$ ($g_{j,k-1}$ and $g_{j,k}$), than for groups $g_{j,k-1}$ and $g_{j,k}$ ($g_{j,k-2}$ and $g_{j,k-1}$), and
– the equal-spacing assumption will not be violated by merging $g_{j,k-2}$ and $g_{j,k-1}$ ($g_{j,k-1}$ and $g_{j,k}$).

Step 6. Once FlexSched finds out that no more adjacent data groups of G_j are eligible for merging, it proceeds by constructing the broadcast schedule for physical broadcast channel c_j. For this purpose, FlexSched applies Algorithm 4 which first logically divides the broadcast program into λ_j MIBCs and then transmits data items one at a time from the data groups belonging to R_j in a round-robin fashion. Due to the iterative nature of the algorithm, it produces an infinite periodic schedule unless either the broadcast contents or the demand probability distribution of the data items changes. Notice that we use the notation R_j instead of G_j in Algorithm 4 to emphasize the fact that the set $R_j = \{g_{j,1}, g_{j,2} \ldots, g_{j,l}\}$ of L data groups contains only a subset of those K data groups originally contained in G_j prior to performing the merge procedure, i.e., $L \leqslant K$.

4.3 Properties of the Algorithm

Before we evaluate the performance of FlexSched in comparison to other state-of-the-art scheduling algorithms, we briefly summarize the main properties of the algorithm:

– A crucial property of the FlexSched algorithm is that it uses a partitioning approach to divide the scheduling problem into a number of sub-problems which can

Algorithm 4: Procedure to construct the broadcast schedule

```
1  $\lambda_j \longleftarrow \max_{\forall g_{j,k} \in R_j} \frac{|g_{j,k}|}{bf^{int}_{j,k}}$; // calculate how many MIBCs are contained in an MBC
2  repeat
3      for $i \longleftarrow 0$ to $\lambda_j - 1$ do
           // construct the broadcast schedule of the $i+1$-th MIBC
4          foreach data group $g_{j,k}$ in set $R_j$ do
               // select which items of group $g_{j,k}$ are to be broadcast in the $i+1$-th MIBC
5              for $j \longleftarrow 1; j \leqslant bf^{int}_{j,k}; j \longleftarrow j+1$ do
6                  $k \longleftarrow (i \cdot bf^{int}_{j,k}) \mod |g_{j,k}|$;
7                  if $k + j > |g_{j,k}|$ then
8                      $k \longleftarrow (k+j) - |g_{j,k}|$
9                  else
10                     $k \longleftarrow k + j$
11                 broadcast $k$-th data item of data group $g_{j,k}$
12 until (broadcast contents or demand probability distribution has changed)
```

be solved independently from each other. Remember in Step 2 of the FlexSched algorithm the set $D = \{d_1, d_2, \ldots, d_n\}$ of N data items is partitioned into M data groups with each data group g_j being associated to a physical broadcast channels c_j. At this early stage of the algorithm, the scheduling problem is divided into M sub-problems which are mutually independent. As a result, the subsequent steps of the FlexSched algorithm (i.e., Steps 3–6) can be easily *parallelized* on multi-processor systems with both shared and distributed memory.

– Another beneficial property of FlexSched is that it produces perfectly periodic schedules, i.e., it obeys the *equal-spacing assumption*. As mentioned above, periodicity is not only a very desirable property from the access latency perspective, but also from the point of view of indexing the broadcast program to reduce the power expenditure of mobile devices. The issue of building energy-efficient indexes and using them is remarkably simplified if instances of data items are spaced equally since each data item appears with a well-known frequency in the program and, more importantly, exactly at the same relative time within a time period.
– As for the computational complexity of FlexSched, we can state that Algorithms 1 and 4 run in time bounded by $\mathcal{O}(N)$, where N is the number of data items to be broadcast. The running time of the merging procedure (Algorithms 2 and 3) strongly depends on the number of data groups being merged during a single iteration of the algorithm's loop. The worst case behavior for FlexSched occurs when only one pair of adjacent data groups is merged per iteration. The running time is then expressed by the recurrence relation $T(N) = \max_{1 \leqslant q \leqslant N-1}(T(q)) + \mathcal{O}(N)$, which obviously has solution $T(N) = \mathcal{O}(N^2)$. In the best case, each data group is merged with any of its neighboring groups during an iteration. The recurrence is then given by $T(N) = \max_{1 \leqslant q \leqslant N/2}(T(q)) + \mathcal{O}(N)$, resulting in the best case running time of $T(N) = \mathcal{O}(N)$. For the average case, suppose that half of the data

groups are merged within a single iteration of the merge loop. The recurrence relation is then defined by $T(N) = \max_{1 \leqslant q \leqslant 3N/4}(T(q)) + \mathcal{O}(N)$, which yields an average running time of $T(N) = \mathcal{O}(N \log N)$.

5 Performance Evaluation

In this section, we present simulation results for FlexSched and a number of other scheduling algorithms which will be briefly described below. If not otherwise stated, in each simulation the number of data items disseminated by the broadcast server is assumed to be 100,000 and the number of available physical broadcast channels is set to 4. Similar to other researchers [2, 3, 8, 17], we assume that the demand probability follows a Zipf distribution [36] with the parameter θ set to 0.8, meaning that approximately 75% of all requests apply to 25% of the database which approximately equals the data reference behavior created by Web and database applications [5, 11]. Note that in Zipf distribution, the probability of accessing the i-th most popular object is:

$$p_i = \frac{(\frac{1}{i})^\theta}{\sum_{j=1}^{N}(\frac{1}{j})^\theta}, \quad 1 \leqslant i \leqslant N, \tag{6}$$

where N is the number of data items to be transfered by the broadcast server and θ is the skewness parameter.

To measure the access and scheduling efficiency of the scheduling algorithms to be evaluated, we use the following two performance metrics: (a) the overall average access latency experienced by mobile clients for responding to data requests and (b) the number of splitting/merging alternatives being examined by the respective scheduling algorithm in order to find the most appropriate partition/segmentation that incurs the lowest costs. While the access latency is a good indicator for the access efficiency of the broadcast schedule, the number of different splitting/merging alternatives being tested is used to quantify the scheduling efficiency of the algorithms.

To evaluate the performance of FlexSched in comparison to other scheduling algorithms, we additionally implemented the following three multi-channel scheduling schemes: (a) <u>Casc</u>adedWebcasting (Casc) [17], (b) <u>V</u>ariant <u>F</u>anout with Constraint <u>K</u> (VF$^{\mathrm{K}}$) [23], and (c) Dichotomic [4]. Although the studied algorithms try to solve the data scheduling problem in different ways, they all use either a top-down or bottom-up approach to allocate the data items to the available broadcast channels and to subsequently generate the broadcast schedule.

The *Casc* scheme is a bottom-up scheme and uses the simplest approach among the three scheduling algorithms to build the broadcast schedule. In the initiation phase of the scheme, the set $D = \{d_1, d_2, \ldots, d_n\}$ of N data items is partitioned into a set $P = \{p_1, p_2, \ldots, p_v\}$ of V data groups with cardinalities $2^0, 2^1, \ldots,$ and 2^{v-1}, respectively. Notice that the cardinality of the v-th, i.e., last data group may be higher than 2^{v-1}. This will be the case if the number N of data items is not equal to $2^v - 1$ and it falls into the interval $2^v - 1 < N < 2^{v+1} - 1$. Then, Casc appends the remaining $r = N - (2^v - 1)$ data items to the v-th data group. Hereafter, the V seed partitions are greedily concatenated in $V - M$ merging steps, where M again represents the number of broadcast channels. At the i-th merging step, the algorithm examines $V - i - 1$ merging alternatives and selects the one incurring the lowest cost increase. It is easy to see that the runtime complexity of Casc is upper-bounded by $\mathcal{O}(N\,(\log^2(N) - M^2))$ with $\mathcal{O}(N)$ being the dominating cost factor.

Similar to Casc, *VF*$^{\mathrm{K}}$ is a heuristic algorithm which adopts a top-down approach to minimize the expected access latency of the data items in the broadcast program. VF$^{\mathrm{K}}$ transforms the problem of allocating the set $D = \{d_1, d_2, \ldots, d_n\}$ of N data items to the available M broadcast channels into one of constructing a channel allocation tree with variant-fanout. The algorithm starts by attaching all data items to the root node of the allocation tree, i.e., all data items are initially assigned to the first broadcast channel. Then iteratively, VF$^{\mathrm{K}}$ chooses the broadcast channel which incurs the highest average access latency and partitions its contents into two groups. This is done by using a partitioning procedure which selects among the set of possible partitions the one which maximizes the reduction gain achieved by attaching the newly created group to a new child node of the allocation tree, i.e., a new unallocated broadcast channel. The partition procedure repeats until all available broadcast channels are allocated, i.e., the height of the allocation tree has increased from 1 to M, where M is the number of available broadcast channels. The runtime cost of choosing the partition with the maximal cost reduction is $M \cdot \mathcal{O}(M \log M)$ and that of partitioning is $M \cdot \mathcal{O}(N)$. Consequently, the runtime complexity of VF$^{\mathrm{K}}$ is $M \cdot (\mathcal{O}(M \log M) + \mathcal{O}(N))$, where $M \cdot \mathcal{O}(N)$ is the dominating cost factor.

The *Dichotomic* scheme is a dynamic programming algorithm which is based on the DP algorithm [34], reducing its time complexity from $\mathcal{O}(N^2\,M)$ to $\mathcal{O}(N\,M\,\log N)$. To find an optimal solution for the problem of partitioning N data items into M groups such that the overall access latency of the schedule is minimized, the Dichotomic algorithm uses two $N \times M$ matrices A and B, where entry $A_{n,m}$ (with $n \leqslant N$ and $m \leqslant M$) of the first matrix contains the scheduling costs $AT^{opt}_{n,m}$ of the optimal solution of grouping data items $d_1, d_2, \ldots, d_n$ into m channels and the entry $B_{n,m}$ contains the final border of the partitioning solution corresponding to entry $A_{n,m}$. Note that the final border of a segmentation

$$\underbrace{d_1, \ldots, d_{b_1}}_{g_1}, \underbrace{d_{b_1+1}, \ldots, d_{b_2}}_{g_2}, \ldots, \underbrace{d_{b_{m-1}+1}, \ldots, d_N}_{g_m}$$

of set $D = \{d_1, d_2, \ldots, d_n\}$ of N data items into M groups is the index of the last item that belongs to the last but one group (g_{m-1}), i.e., b_{m-1}. Entries of both matrices are filled up row by row and the entries of each row are being determined and inserted in stages. At the i-th stage, 2^{i-1} entries of a row are calculated. Thus, all entries of a row are filled in $\lceil \log N \rceil$ stages. The total number of calculations involved in filling up the entries of a particular stage are bounded by $\mathcal{O}(N)$. Since the algorithm is executed in $\lceil \log N \rceil$ stages and both matrices consists of M rows, the runtime complexity is obviously given by $\mathcal{O}(N\,M\,\log N)$.

To be able to assess the quality of the solutions obtained by FlexSched and the other three scheduling algorithms, we compare them to the optimal overall average access time, denoted by AT^{opt}, which is obtained by the following formula:

$$AT^{opt} = \frac{1}{2} \cdot \frac{\left(\sum_{i=1}^{N} \sqrt{p_i}\right)^2}{\sum_{j=1}^{M} \frac{bc_j}{\min_{k=\{1,2,\ldots,M\}} bc_k}}. \tag{7}$$

Notice that AT^{opt} is derived by assuming that the equal-spacing assumption holds. Remember that this assumption cannot always be realized, so AT^{opt} represents a lower bound on the achievable overall average access time. Notice also that the value of AT^{opt} in the equation above is normalized relative to the physical broadcast channel with the lowest bandwidth capacity to take account of possible variations in the capacity of the channels.

5.1 Experiment 1. Effect of the Database Size

In the first experiment, we studied the effect of changing the size of the database on the average access latency of the broadcast program. Thereby, the number of data items to be broadcast by the server was varied from 1,000 to 200,000. Figure 2 shows the results of this experiment. By inspecting the results, we observe that the average access latency of FlexSched is always very close to the optimal value. The performance gap between AT^{opt} and FlexSched is not more than 3% for any database size examined. In addition, the average access latency of FlexSched is, on average, 9% and 11% lower than that of Dichotomic and VF$^{\mathrm{K}}$, respectively. The reason for FlexSched's outperformance w.r.t. the other scheduling algorithms is the fact that it uses skewed scheduling per broadcast channel, whereas Dichotomic, VF$^{\mathrm{K}}$, and Casc rely on flat scheduling. Remember that in flat scheduling the data items are transmitted in a round-robin fashion with each data item being allotted one time slot per MBC. Such scheduling behavior obviously leads to a waste of channel bandwidth when the distribution of the demand probabilities of the data items is non-uniform.

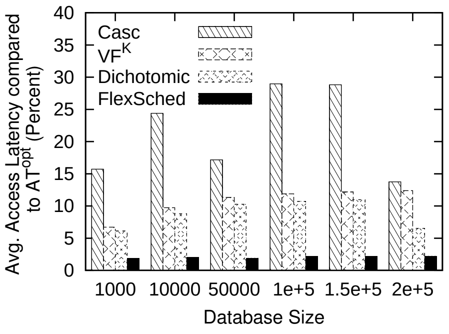

Fig. 2. Increase in average access latency compared to AT^{opt} for various sizes of the database.

5.2 Experiment 2. Effect of the Degree of Data Access Skewness

In order to study the effect of skewness in the distribution of the demand probabilities of the data items on the performance characteristics of the scheduling algorithms, we varied the value of the Zipfian parameter θ in the range from 0.1 to 0.9 with increments of 0.2. Figure 3 shows the results of this experiment. It can be seen that when θ is small (i.e., $\theta \leqslant 0.3$), the performance of all four scheduling algorithms, except for Casc, is fairly close to the optimal value AT^{opt}. With increasing skewness in the distribution of the demand probabilities (i.e., $\theta > 0.3$), however, the performance of all four scheduling algorithms starts to deteriorate relative to AT^{opt}. At the same time, a performance gap between FlexSched and the other broadcast schedulers begins to emerge which further widens as θ increases. For example, for $\theta = 0.5$, the performance penalty of Dichotomic and VF^{K} schemes relative to FlexSched is less than 1%, whereas for $\theta = 0.9$, the relative performance gap widens to around 12% and 14%, respectively. The reason for the increasingly poorer performance of the other scheduling algorithms with increasing θ values is again related to the adoption of different scheduling approaches (flat vs. skewed).

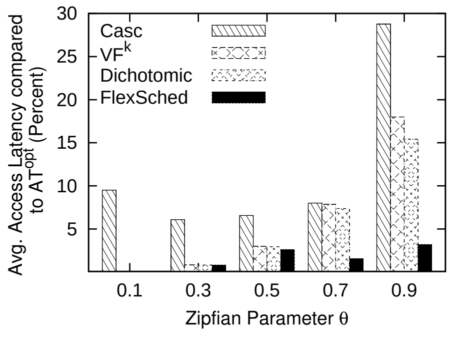

Fig. 3. Increase in average access latency compared to AT^{opt} for various values of the Zipfian parameter θ.

5.3 Experiment 3. Effect of the Number of Physical Broadcast Channels

Next, we studied the effect of the number of available broadcast channels on the scheduling performance of the various broadcast schedulers. To this end, we varied the number of physical broadcast channels from 1 to 8. As can be seen from Figure 4, the number of available broadcast channels has a significant impact on both the absolute and relative performance behavior of the scheduling algorithms. As the number of broadcast channels increases, the average access latencies of the broadcast system decreases since more and more data items can be transmitted per time unit. From Figure 4 one also observes that the performance difference between FlexSched and the other three scheduling algorithms decreases with increasing numbers of broadcast channels. For example, if only a single broadcast channel is available, the relative performance penalty (i.e., the increase in average access latency) of using Dichotomic and VF^{K} instead of FlexSched is about 55%; however, if the data items are spread over eight broadcast channels, the relative performance advantage of FlexSched compared to Dichotomic and VF^{K} decreases to values of 4% and 10%, respectively. The narrowing gap in the relative performance of the scheduling algorithms can be explained by the fact that the advantage of skewed over flat scheduling gradually decreases as the number of available broadcast channels grows.

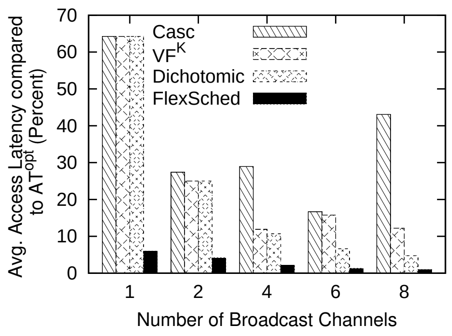

Fig. 4. Increase in average access latency compared to AT^{opt} for various numbers of available broadcast channels.

5.4 Experiment 4. Runtime Overhead

In the last experiment, we investigated the scheduling costs incurred by the different broadcasting algorithms. To ensure a fair comparison and to abstract from implementation details specific to particular algorithms, we use the metric “number of splitting/merging alternatives examined” as the measure for quantifying the actual runtime costs of the algorithms. Figure 5(a) shows the runtime costs of the studied algorithms for various sizes of the database. As expected from the time complexity analysis of the algorithms, Casc has the lowest CPU overhead among the four data schedulers. Casc is followed by FlexSched whose runtime costs are, on average, 25% and 96% lower than those of the VF^{K} and Dichotomic schemes, respectively.

Finally, Figure 5(b) illustrates the change in runtime costs when the number of available broadcast channels is altered. In the single-channel case, the computational overhead of Casc, VF^{K}, and Dichotomic is zero as data items do not need to be spread among multiple broadcast channels. This situation instantly changes, however, as more broadcast channels become available. Under those circumstances, the computational overhead of the other algorithms, with the exception of Casc, tends to be higher than that of FlexSched. In addition to that, we observe that the runtime overhead of Dichotomic and VF^{K} increases as the number of available broadcast channels increases, while the opposite is true for FlexSched and Casc. This is because the runtime costs of Dichotomic and VF^{K} are driven by both the database size and the number of broadcast channels, while the computational overhead of Casc and FlexSched is tied primarily to the database size.

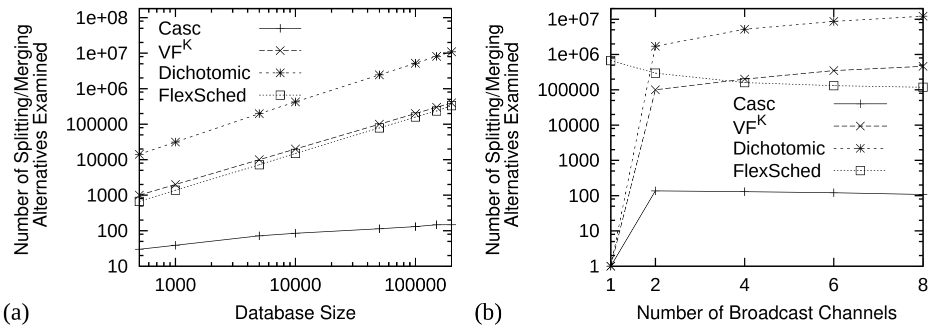

Fig. 5. Number of splitting/merging alternatives examined as a function of the (a) database size and (b) number of available broadcast channels.

6 Summary

In this paper, we studied the problem of constructing latency-minimal perfectly periodic broadcast schedules for multi-channel broadcast environments when the data items are equal-sized and their access probabilities are a priori known by the broadcast server. We presented a new scheduling algorithm, called FlexSched, which is built upon the concepts of skewed data allocation and skewed data scheduling to minimize the average access latency of the corresponding broadcast program. In order to evaluate the cost efficiency of FlexSched, we analyzed its runtime complexity and compared it against the state-of-the-art scheduling algorithms. We also analyzed its scheduling performance in a comparative study and conducted a sensitivity analysis on several simulation parameters, including the database size, the number of available broadcast channels, and the skewness in the distribution of the demand probabilities of the data items. Our performance evaluation has proven FlexSched to outperform the existing scheduling algorithms under a wide range of system settings and workload conditions and to provide performance results very close to the optimum. Due to is low average runtime complexity and good performance characteristics, FlexSched turns out to be a very well suited approach to solving the data allocation and scheduling problem in both single and multi-channel broadcast systems.

References

1. Ambient Devices Inc. Ambient Information Network and Device Design, 2005, `http://www.ambientdevices.com`.
2. M. H. Ammar and J. Wong. The Design of Teletext Broadcast Cycles. *Performance Evaluation* 5(4):235–242, 1985.
3. M. H. Ammar and J. Wong. On the Optimality of Cyclic Transmission in Teletext Systems. *IEEE Transactions on Communications* 35(1):68–73, 1987.
4. E. Ardizzoni, A. A. Bertossi, M. C. Pinotti, S. Ramaprasad, R. Rizzi, and M. V. S. Shashanka. Optimal Skewed Data Allocation on Multiple Channels with Flat Broadcast per Channel. *IEEE Transactions on Computers* 54(5):558–572, 2005.

5. P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in Web Client Access Patterns: Characteristics and Caching Implications. *World Wide Web* 2(1-2):15–28, 1999.
6. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. *Infocom 1999*, pp. 126–134, 1999.
7. M. S. Chen, K. L. Wu, and P. S. Yu. Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing. *IEEE TKDE* 15(1):161–173, 2003.
8. S. Hameed and N. H. Vaidya. Log-Time Algorithms for Scheduling Single and Multiple Channel Data Broadcast. *MobiCom 1997*, pp. 90–99, 1997.
9. C.-H. Hsu, G. Lee, and A. L. P. Chen. A Near Optimal Algorithm for Generating Broadcast Programs on Multiple Channels. *CIKM 2001*, pp. 303–309, 2001.
10. C.-H. Hsu, G. Lee, and A. L. P. Chen. Index and Data Allocation on Multiple Broadcast Channels Considering Data Access Frequencies. *MDM 2002*, pp. 87–93, 2002.
11. W. W. Hsu, A. J. Smith, and H. C. Young. Characteristics of Production Database Workloads and the TPC Benchmarks. *IBM Systems Journal* 40(3):781–802, 2001.
12. Q. L. Hu, W. C. Lee, and D. L. Lee. A Hybrid Index Technique for Power Efficient Data Broadcast. *DPDB* 9(2):151–177, 2001.
13. J.-H. Hwang, S. H. Cho, and C.-S. Hwang. Optimized Scheduling on Broadcast Disks. *MDM 2001*, pp. 91–104, 2001.
14. T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power Efficient Filtering of Data an Air. *EDBT 1994*, pp. 245–258, 1994.
15. T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering* 9(3):353–372, 1997.
16. R. Jain and J. Werth. Airdisks and airRAID: Modeling and Scheduling Periodic Wireless Data Broadcast. *SIGARCH Comput. Archit. News* 23(4):23–28, 1995.
17. D. Katsaros and Y. Manolopoulos. Broadcast Program Generation for Webcasting. *Data & Knowledge Engineering* 49(1):1–21, 2004.
18. S. Khanna and S. Zhou. On Indexed Data Broadcast. *STOC 1998*, pp. 463–472, 1998.
19. W. C. Lee and D. L. Lee. Using Signature Techniques for Information Filtering in Wireless and Mobile Environments. *DPDB* 4(3):205–227, 1996.
20. S.-C. Lo and A. L. P. Chen. Optimal Index and Data Allocation in Multiple Broadcast Channels. *ICDE 2000*, pp. 293–302, 2000.
21. Microsoft Corporation. DirectBand Network. Microsoft Smart Personal Objects Technology (SPOT), 2005, `http://www.microsoft.com/resources/spot`.
22. 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* 51(6):1652–1660, 2002.
23. W.-C. Peng and M.-S. Chen. Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment. *Wireless Networks* 9(2):117–129, 2003.
24. K. Prabhakara, K. A. Hua, and J. H. Oh. Multi-Level Multi-Channel Air Cache Designs for Broadcasting in a Mobile Environment. *ICDE 2000*, pp. 167–176, 2000.
25. T. Sakata and J. X. Yu. Statistical Estimation of Access Frequencies: Problems, Solutions and Consistencies. *ACM Wireless Networks* 9(6):647–657, 2003.
26. N. Shivakumar and S. Venkatasubramanian. Energy-Efficient Indexing for Information Dissemination in Wireless Systems. *MONET* 1(4):433–446, 1996.
27. SkyTel Corporation. Timex Internet Messenger, 2005, `http://mobile.timex.com/indexENTER.html`.
28. K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive Data Broadcast in Hybrid Networks. *VLDB 1997*, pp. 326–335, 1997.
29. N. Vaidya and S. Hameed. Data Broadcast in Asymmetric Wireless Environments. *WOSBIS*, November 1996.

30. N. Vaidya and S. Hameed. Improved Algorithms for Scheduling Data Broadcast. Tech. rep., Texas A & M University, 1996.
31. N. H. Vaidya and S. Hameed. Scheduling Data Broadcast in Asymmetric Communication Environments. *ACM Wireless Networks* 5(3):171–182, 1999.
32. J. Xu, W.-C. Lee, and X. Tang. Exponential Index: A Parameterized Distributed Indexing Scheme for Data on Air. *MobiSys 2004*, pp. 153–164, 2004.
33. W. G. Yee and S. B. Navathe. Efficient Data Access to Multi-channel Broadcast Programs. *CIKM 2003*, pp. 153–160, 2003.
34. W. G. Yee, S. B. Navathe, E. Omiecinski, and C. Jermaine. Efficient Data Allocation over Multiple Channels at Broadcast Servers. *IEEE Transactions on Computers* 51(10):1231–1236, 2002.
35. J. X. Yu, T. Sakata, and K.-L. Tan. Statistical Estimation of Access Frequencies in Data Broadcasting Environments. *ACM Wireless Networks* 6(2):89–98, 1999.
36. G. K. Zipf. *Human Behavior and Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley, 1949.