Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 49 (2004) 1–21
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Broadcast program generation for Webcasting

Dimitrios Katsaros, Yannis Manolopoulos *

Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece

Received 30 October 2002; received in revised form 28 May 2003; accepted 30 July 2003

Abstract

The advances in computer and communication technologies have made possible an ubiquitous computing environment were clients equipped with portable devices can send and receive data anytime and from anyplace. In such an *asymmetric communication environment*, *data push* has emerged as a very effective and scalable way to deliver information. Recently, the combination of push technology with the Internet and the Web [IEEE Trans. Comput. 50 (2001) 506, ACM/Kluwer Mobile Networks Appl. 7 (2002) 67] (referred to as *Webcasting*) has emerged "as a way out of the Web maze". Any broadcast program employed for Webcasting must be able to scale to the large number of transmitted pages.

We study the issue of creating hierarchical Webcasting programs. We propose a new algorithm, *CascadedWebcasting*, for the generation of Webcasting programs, scalable in terms of the number of items transmitted and able to produce programs very close to optimal. We give an analytic model of the time complexity of the proposed method and we present a performance evaluation of *CascadedWebcasting* and a detailed comparison with existing algorithms using synthetic as well as real data. The experiments show that the *CascadedWebcasting* has negligible execution time and achieves an average access delay very close to that of the optimal algorithm.

Keywords: Webcasting; Broadcast disks; Scheduling; Push-based delivery; Wireless data dissemination; Mobile computing

1. Introduction

The recent advances in computer and wireless communications technology promise to make possible an ubiquitous computing environment where users equipped with portable computers

* Corresponding author. Tel.: +30-2310-996363; fax: +30-2310-996360.
E-mail addresses: dkatsaro@csd.auth.gr (D. Katsaros), manolopo@skyblue.csd.auth.gr (Y. Manolopoulos).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2003.07.001

(PDAs, cellular phones) can receive the data of interest anytime and from anyplace [3]. In this environment there is one (or more) data providers (*servers*) supplying data to a number of clients. Served data include newspapers headlines, stock prices, sports news, etc. One challenge that must be met in order to fully realize this environment is to address its particularities, which mainly stem from the communication asymmetry [4]. Asymmetry in communications may arise because the communication capacity from the server(s) to clients is much larger than that from clients to servers. In a wireless mobile network, mobile support stations have a transmission channel with relatively high bandwidth whereas clients have little or no transmission capability.

Data delivery in such *asymmetric communication environments* can be either *pull-based* or *push-based* [5,6]. In pull-based delivery, there exist two channels, a *donwlink channel*, which is used by the server to send out data, and an *uplink channel* onto which clients send their requests to the server. In pull-based delivery, data transfers are initiated by the clients, just like in traditional client-server database systems [7] which manage data for clients that explicitly request them. Obviously, this request–response method of delivery suffers from scalability problems, since the server can become the "hot-spot" of the communication.

Push-based delivery (and its variations [8]) relies on a server that continuously and repetitive sends out information to a (possibly unknown) number of clients usually through a *broadcast channel*. Clients tune into this channel waiting for the data of interest to arrive. Due to the one way broadcast, system state information such as the number of pending requests for a data item is not available, but one can only rely on information such as how frequently a given item is requested by the users or on the registered user preference (e.g., profiles or subscriptions) on a given item [9]. Broadcast delivery is attractive, because of the excellent scalability it offers. In broadcast delivery, the response is not affected by the load, since the single broadcast of a data item satisfies all pending requests for it. Thus it can support an infinite number of clients. On the other hand, the response time depends on the number of transmitted items and increases with increasing number of items.

Early examples of broadcast delivery are the *teletext/videotex* [10], the Boston Community Information System [11] and the Datacycle database machine [12].

1.1. *Web + Push technology = Webcasting*

Recently, the combination of push technology with the Internet and the Web, referred to as *Webcasting*, has emerged "as a way out of the Web maze" [6].

The application of push technology to the Internet and the Web grew significantly after the PointCast Network was deployed in 1996. Since then, it develops rapidly and several new systems and architectures have been born, such as the Global Information Broadcast [13] or SkyCache [14]. These types of systems can be used over satellite links, where the consumers may be human end-users, file servers or *HTTP* proxy cache servers [1,2]. The aforementioned systems utilize scheduled transmission, where the data source transmits information resources on a regular time schedule.

These schedules are characterized by two properties: (a) the large number of data items transmitted and (b) the need for frequently updating the schedule. The latter property can be understood if we consider that the access probabilities of the transmitted items change quite frequently or because new items must be transmitted. Change in access probabilities can arise as a

result of client mobility, since the clients leaving (or entering) a cell can affect the data item demand probabilities. Change in access probabilities can also arise due to the varying client interests.

As a concrete example of this kind of applications, consider a broadcast that transmits a large number of data items referring to stock prices. The "volatility" of the prices of the stocks makes the frequent schedule update absolutely necessary, since it is known that the interest of the investors is a function of the current prices. As another example consider the *cache-satellite system* described in [2,14]. There, the *master server* broadcasts to the cooperating proxies all the documents that resulted as cache misses. In this case, it is obvious that the schedule needs to be frequently updated, because different documents with different popularities must be transmitted. Needless to say that in both examples the number of transmitted items is by far larger than a few hundreds of items. Similar are the applications that employ a satellite-terrestrial wireless system in order to serve Web data via a satellite from a main server to a number of ground stations [1].

It is obvious, that in these applications the derived schedule will not be maintained for days, but will probably change after a few minutes. Thus, the schedule determination procedure should be done really fast.

In Webcasting, like in traditional broadcasting systems, the information is organized into units, called *pages*, and is broadcast to the clients through one or multiple broadcast channels. Such an environment is depicted in Fig. 1, where mobile clients tune into the broadcast channel(s) to receive the information of interest.

The allocation of data items to the channel(s) is critical for efficient data access. Access efficiency is usually measured in terms of the *access time*, that is, the time elapsed from the moment a client tunes into the channel(s) to retrieve the data of interest to the moment these data are acquired. Thus, a program, the *broadcast program*, needs to be constructed that will schedule the data transmissions, such that the client response time is minimized.

When the access probabilities of the data are equal, then a *flat* broadcast program can be constructed which is optimal in that it achieves the minimum average response time. During a *broadcast cycle* in a flat program, all items are broadcast once and this cycle is repeated.

Several studies though, indicate that Web accesses are skewed, following the Zip's law [15]. Thus, hierarchical broadcast programs that provide improved performance for non-uniformly

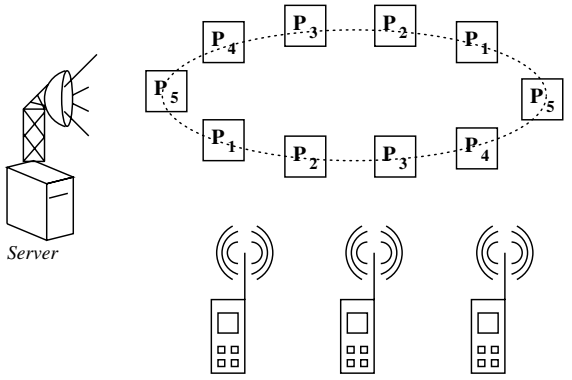

Fig. 1. A server broadcasting data to mobile clients.

accessed data are necessary for Webcasting. *Broadcast disks* were introduced in [4] (and generalized in [16]) as an alternative to flat programs. Broadcast disks superimpose multiple disks of different sizes, spinning at different speeds on a single broadcast channel creating thus an arbitrarily fine-grained memory hierarchy. The program generated by the broadcast disks emphasizes the most popular items (the items stored at the smallest, fast spinning disks) and de-emphasizes the less popular ones (the items stored at the largest, slow spinning disks). The number of disks and the items assigned to each disk determine the average delay. The broadcast disk program guarantees a fixed interarrival time between all successive transmissions of an item and also guarantees fixed broadcast cycle. (More details regarding the program composition for broadcast disks can be found in [4].)

In parallel, in order to deal with the communication asymmetry, the client can perform caching [4,17,18] and prefetching [19–21] or the server can broadcast indices [22]. Although the problem of broadcast disks is twofold, that is, it consists of the determination of the disk frequencies and the contents of each disk, the latter one is much harder, as the extensive research work on it reveals. Thus, the fundamental decision in any Webcasting system implementing the broadcast disks paradigm is the partitioning of items into groups, each one of the groups corresponding to a broadcast disk [23–27]. The present work deals with the issue of creating hierarchical broadcast programs for Webcasting and the issues of caching/prefetching and index broadcast are orthogonal to our work.

1.2. Related work and motivation

The characteristics of the schedule, as described in Section 1.1, imply that the speed of the schedule determination for the Webcasting is equally important to the quality (measured in terms of the average access time) of the derived schedule. Moreover, if we consider that the exact access probabilities of the data items in this environment are not precisely known and thus there is little chance to derive a program that matches exactly the user interests, we can understand that it would be of little practical importance to devise a new algorithm that improves the delay in average time by a few percentage points. Under this setting, the time complexity of the algorithm employed for the schedule determination is of more practical interest.

Five approaches considered so far this partitioning problem [23–27]—we defer the detailed presentation of these methods until Section 2.2, where we derive some bound for their time complexity. Nevertheless, none of these approaches can produce programs able to address both requirements of the Webcasting environment (or any other large scale broadcasting environment e.g., [12]), namely (a) skewed access pattern and (b) very large number of items transmitted.

The work in [23] assigns items to disks making only one pass over the vector of frequencies of the data, assigning items to disks by comparing their frequency with the difference in frequency between the most popular and least popular item. Consequently, it is very fast in creating the respective program, but it is vulnerable to skewed access probabilities. The works in [24–27] are more sophisticated, in that they try to deal with non-uniformity in access probabilities and thus to produce better programs, but require very large processing time when the number of items to be transmitted is more than a thousand, thus they do not exhibit good scalability in terms of the data set size.

The algorithm in [24] starts with an initial assignment of the items to the disks (segments) and then, based on a greedy approach, keeps growing the segment which leads to the lowest average

delay. The work in [25] takes the opposite approach and greedily splits the segment, which incurs the largest average access delay. The work in [26] remedies a disadvantage of the previous one which appears when the number of disks is not a power of 2. Finally, the work in [27] is based on [25], but its nature is closer to a dynamic programming approach rather than to a greedy one.

Similar to the work in [28], which identified the need for scalable algorithms in terms of data set sizes, client population and broadcast bandwidth for on-demand data broadcast, we address the same issues in push-based delivery systems. The aim of the present work is to develop a fast algorithm, which will be very efficient in handling skewed access frequencies and will be able to produce programs very close to the optimal.

1.3. Contributions

This paper focuses on *Webcasting* and makes the following contributions:

- We identify the two requirements [1] that any *Webcasting* algorithm should satisfy, namely sensitivity to skewed access probabilities and scalability to the number of data items being broadcast.
- We observe that the algorithms used for assigning items to multiple channels can be used in the determination of the contents of the broadcast disks as well. [2]
- We develop cost formulas for the time complexity of the schemes described in [24] and [26], since they lack one.
- We develop a new algorithm for *Webcasting* and a cost formula for its time complexity.
- Finally, through an extensive experimental comparison of all algorithms, we deduce that the proposed algorithm outperforms existing ones when running times are considered, whereas it is not much worse than the optimal algorithm.

The rest of the paper is organized as follows: Section 2 presents the necessary background information regarding the broadcast disks. Also it reviews briefly the related work and presents the cost models for the time complexity of the corresponding algorithms. Section 3 describes the proposed algorithm whereas Section 4 provides the experimental results. Finally, Section 5 concludes the present work.

2. Preliminaries

2.1. Problem formulation

Due to the nice characteristics of the broadcast disk paradigm as described in [4] (fixed interarrival times and fixed length broadcast cycle) we will adopt that model in the present work. We consider a database $\mathscr{D}$ of equal-sized data items for which the access probabilities are given.

[1] The first one was implicitly stated in earlier works as well (see [4]).
[2] The authors in [26] made the reverse observation.

(Techniques for estimating the access probabilities can be found in [29,30].) Let our database be comprised by n items, that is, $\mathscr{D} = \{d_1, d_2, \ldots, d_n\}$. The items are ordered from the most popular to the least popular and have access probabilities $\mathscr{P} = \{p_1, p_2, \ldots, p_n\}$, respectively, where $\sum_{i=1}^{n} p_i = 1$. This ordering means that $p_i \geqslant p_j$, $1 \leqslant i < j \leqslant n$. W.l.o.g. we can assume that $n = 2^v - 1$. Each *item* (or *data item* or *page*) is assigned a rank according to its position in that order. The rank of the most popular item is 1.

Suppose that we have a set $\mathbb{D}$ of k broadcast disks $\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_k$, with $\mathbb{D}_1$ being the fastest disk and $\mathbb{D}_k$ being the slowest disk. The frequencies (spin) of these disks are equal to $f_1, f_2, \ldots, f_k$, respectively, for which $f_i > f_j$, $1 \leqslant i < j \leqslant k$. The number of database items comprising a disk will be called the *size* of the disk and will be denoted as $|\mathbb{D}_i|$, $1 \leqslant i \leqslant k$. The spin of each disk is a function of its size, with the smallest disk spinning faster ($|\mathbb{D}_i| < |\mathbb{D}_j|$, $1 \leqslant i < j \leqslant k$). We assume a slotted time model in which the server broadcasts one page per time slot.

A broadcast program can be characterized by an assignment $\mathscr{G} : [1, \ldots, n] \to [1, \ldots, k]$ of pages to the disks. Since more popular pages should be transmitted more frequently and thus to be accommodated in faster disks, it is obvious that there can exist no pair d_i, d_j with $i < j$ in the above ordering such that $\mathscr{G}(d_i) > \mathscr{G}(d_j)$. This means that either the two items will be accommodated into the same disk or the more popular of the two items will be accommodated into a faster disk. Thus, creating a broadcast program for k disks is equivalent to determining a *partition* of the interval $[1, \ldots, n]$.

It is obvious that a page d_j assigned to a particular disk $\mathbb{D}_i$ will have the same broadcast frequency with that of the disk, that is, $\mathrm{freq}(d_j) = f_i$.

For our convenience we recall some definitions from [4,24]:

Definition 1 (*Major Cycle*). The *Major Cycle* C of the broadcast is defined as

$$C = \sum_{d \in \mathscr{D}} \mathrm{freq}(d). \tag{1}$$

Lemma 2.1. *The Major Cycle C satisfies the property*:

$$C = \sum_{i=1}^{k} n_i * f_i, \tag{2}$$

where n_i denotes the number of items of disk $\mathbb{D}_i$.

Definition 2. The expected delay for a page d_i, denoted as $\omega(d_i)$ is

$$\omega(d_i) = \frac{p_i * C}{2 * \mathrm{freq}(d_i)} \tag{3}$$

or, equivalently

$$\omega(d_i) = \frac{p_i * w_i}{2}, \tag{4}$$

where w_i is the (fixed) interarrival time of item i.

Lemma 2.2. *Let B_i be the sum of the access probabilities of the pages in disk $\mathbb{D}_i$, that is, $B_i = \sum_{d_j \in \mathbb{D}_i} p(d_j)$. The total expected delay ω can be expressed as*

$$\omega = \frac{k}{2} * \sum_{i=1}^{k} n_i * B_i. \tag{5}$$

Now we can define the *Webcasting* problem as follows:

Definition 3 (*Webcasting problem*). Suppose that we have a database $\mathscr{D}$ of $n = 2^v - 1$ data items, for which the access probabilities are given, and a set $\mathbb{D}$ of k broadcast disks ($\mathbb{D}_1$ being the fastest disk). The *Webcasting problem* is the task of partitioning these items into the given disks, so that the average delay is minimized (Eq. (5)).

Obviously, in the above definition we assume that it is possible to achieve the *equal-spacing criterion* [5,23] in the transmission of the data items.

In what follows in this section, we will briefly present how five approaches deal with the *Webcasting* problem.

2.2. Time complexities of the current approaches for program generation

The procedure of determining a partition can be "top–down", "bottom–up" or "one-scan". In the top–down [25–27] we start from a large partition, possibly including all the items, and gradually split it into smaller pieces. In the bottom–up [24] we start with many small partitions which gradually grow, whereas the one-scan approach makes a single scan over $\mathscr{P}$ assigning items to disks based on some criterion. The top–down and bottom–up schemes make multiple passes over $\mathscr{P}$ (or parts of it) and make splitting or concatenating decisions based on the computation of Eq. (5) over $\mathscr{P}$ (or portions of it).

The cost of an algorithm is entirely determined by the number of passes it will make over the vector $\mathscr{P}$. The total cost incurred is the number of passes times the cost to compute the value of Eq. (5). Of course, when we compute the value of Eq. (5) for two candidate partitions that differ only in the allocation of one item in two consecutive disks, this computation can be done incrementally and thus avoid the recomputation of Eq. (5) for all the disks of the partitions. Nevertheless, the exploitation of such an incremental computation is not expected to reduce significantly the actual running time. This observation is confirmed by the experimental results in Section 4, where all the algorithms have been implemented using the incremental computation.

The *Bucketing scheme* [23] is the simplest approach for the assignment of items to disks. It makes a single scan over the vector of access probabilities of the data and assigns them to disks based on the following criterion. Let $A_{\min}$ ($A_{\max}$) denote the minimum (maximum) value of $\sqrt{p_i}$, $1 \leqslant i \leqslant n$. Let $\delta = A_{\max} - A_{\min}$. If, for the ith item, $\sqrt{p_i} = A_{\min}$, then i is assigned at disk $\mathbb{D}_k$. Any other item i is assigned to disk $\mathbb{D}_{k-j}$ $(1 \leqslant j \leqslant k)$ if $(j-1)\delta/k < (\sqrt{p_i} - A_{\min}) \leqslant (j\delta/k)$. This partitioning criterion is suitable for the case when the access probabilities are uniformly distributed over the "probability interval" and performs poorly for skewed access patterns. On the other hand, this approach has the lowest complexity. It executes in $O(n)$ time and never tries any "candidate" partitions in order to select the most appropriate.

Proposition 2.3 [23]. *The complexity of the Bucketing scheme is* $O(n)$.

The *Growing Segments scheme* [24] starts with an initial "minimal" allocation assigning one item to each disk, which acts as the initial "seed" partition. Then, enlarges each segment by including a number of items equal to the user-defined parameter *increment* and computes which of these enlargements gives the greatest reduction in average delay. Then, selects the corresponding partition as the new "seed" partition and continues until the partition covers the whole $\mathscr{P}$. The parameter *increment* is very important and has the following trade-off associated with it: the greater is the value of *increment*, the lower complexity the algorithm has and the lower quality the produced broadcast program has. Since the computation of Eq. (5) takes $O(n)$ time, the algorithm makes $O\left(\frac{n-k}{\text{increment}}\right)$ steps and at each step computes $O(k)$ candidate partitions, it follows that:

Proposition 2.4. *The complexity of the Growing Segments method is* $O\left(n^2 * \frac{k}{\text{increment}}\right)$.

The *Variant-Fanout Tree scheme with the constraint K* (VF^K) [25] adopts a top–down approach. It starts with an initial allocation, where all the items have been assigned to the first disk. Then repetitively, determines which disk incurs the largest cost so far and partitions its contents into two groups. The partitioning is done by the routine *Partition*, which tries all possible partitions that respect the property that no disk can have more items than its next disk. The first group remains in the current disk and the newly created group is allocated to the next disk, shifting all the other disks downwards. This procedure repeats until all available disks are allocated. The cost of this algorithm is $O(k)$ times the cost of the *Partition* procedure. The cost of this procedure depends on the number of items of the disk that is to be partitioned, which in turn depends on the distribution of the access probabilities. It can be shown that:

Proposition 2.5 [25]. *The complexity of the* VF^K *method is* $O(k * n)$.

The *Greedy scheme* [27] adopts the top–down approach and it is very similar to the VF^K scheme. It performs several iterations. At each iteration, it chooses to partition the contents of the disk, whose split will bring the largest reduction in access time. The partitioning point is determined by calling the routine *Partition* (see above). Thus, at each iteration the *Greedy* computes (if not already computed) and stores the optimal split points for all disks that have not been split so far. Hence, it differs from VF^K in two aspects. Firstly, in the partitioning criterion (recall that VF^K splits the disk which incurs the largest access time). Secondly because after each split, it will compute and store the optimal split points of every disk. Based on these observations it easy to deduce that the cost of the *Greedy* algorithm is:

Proposition 2.6 [27]. *The complexity of the Greedy method is* $O((n+k) * log(k))$.

The *Data-Based* (*DB*) *scheme* [26] is similar to VF^K, but avoids taking the local optimal decision of the *Partition* routine of VF^K, which splits a disk into two. *DB* has several phases. At each phase decides about the contents of a particular disk, starting from the fastest disk which will accommodate the more frequently accessed data. First, it determines which is the maximum allowable number of items that can be accommodated into the considered disk. This number can be

computed with the help of Lemmas 3 and 4 in [26]. Then, it computes the average access delay for all the allowable allocations of items into the considered disk and selects the allocation with the minimum cost. The computation of the average access delay takes also into account the delay that will be incurred due to the items that will be allocated to the rest of the disks. This is the difference from VF^K. This above procedure continues until the allocation of all the items into the disks.

Let us investigate the time complexity of the *DB* method. Let a_i denotes the number of items that have been allocated to disk i after the execution of the algorithm, where $a_0 = 0$. Also, let $\mu_i = \sum_{j=1}^{i-1} a_j$ be the total number of items allocated to disks $\mathbb{D}_j$, $1 \leqslant j \leqslant i-1$. During the ith phase (the determination of the contents of the ith disk, $1 \leqslant i \leqslant k$) the *DB* needs to examine ρ_i possible allocations, where ρ_i is the range of the number of items in $\mathbb{D}_i$ as determined by Lemma 4 of [26]. Obviously, $\rho_i \geqslant (a_i - a_{i-1})$. It is also clear that ρ_i increases with decreasing μ_i. Thus *DB* will incur a cost of:

$$c_i = \rho_i * \left(n - \sum_{j=1}^{i-1} a_i \right).$$

Summing over all phases, we have the total cost:

$$C_{\text{tot}}^{db} = \sum_{i=1}^{k} (\rho_i * (n - \mu_i)). \tag{6}$$

Proposition 2.7. *The complexity of the DB method is*

$$C_{\text{tot}}^{db} = \sum_{i=1}^{k} (\rho_i * (n - \mu_i)).$$

Concluding this section, we observe that all the algorithms incur significant time complexity for the generation of the broadcast program trying to differentiate between frequencies which are almost identical or contribute only a little in the total delay. This comprises our motivation in pursuing a scheme that will have low time complexity and at the same time will produce broadcast programs close to optimal.

3. The Cascaded scheme for Webcasting

3.1. The basic intuition

Every approach for program generation is pursuing a partition of the vector $\mathscr{P}$ into groups of items with approximately the same probability. In order to discover these groups, the approaches described in [24–27] make several passes over $\mathscr{P}$, thus resulting into significant computation complexity. On the other hand, the advantage of making several passes over $\mathscr{P}$ or portions of it, is that they are able to produce better programs.

Alternatively to these approaches, we can use our intuition about the number and the size of the partitions of $\mathscr{P}$. The intuition for such a partitioning is that there exist three classes of items (see [31]):

- A practically constant number of items with *high probability*.
- Items belonging to a few *large groups*.
- Leftover items, which contribute *negligibly* to the total delay.

The items belonging to the first category are those that very "hot" and constitute the great majority of requests from all users. They represent the high overlap between the clients' interests. The items of the second group are definitely less popular than the respective of the first category and they represent the common interests of large groups ("communities") of users. Finally, the items belonging to the last category represent the individual client (or small group of clients) interests.

The intuition for the existence of this partitioning is that, due to the rounding, the probabilities of the successive groups decrease exponentially fast [31] and thus we need a relatively large number of items of the second category to "compensate" for the popularity of a first category item and a fairly large number of third category items to "compensate" for the popularity of a first category item.

The question that arises is how to design a low time complexity scheme that uses the above intuition in order to derive a partition of $\mathscr{P}$.

3.2. The CascadedWebcasting scheme

The above intuition suggests that, regardless of the number of disks that must be used, initially the items can be clustered into groups using as clustering criterion their popularity. Such a clustering criterion is present in all the bottom–up and one-scan schemes used for program generation, but the generated clusters reflect also the different philosophies adopted by each scheme. For instance, the *Bucketing* scheme implements this clustering, although it is done in a very crude way (cluster the items using the value of $\sqrt{p_i} - \min_{d_i \in \mathscr{D}}\{\sqrt{p_i}\}$). The *Growing Segments* scheme has a similar clustering criterion, but it is very primitive since it assigns one item to each partition. Of course, this is not a problem for this algorithm because it will subsequently try successively larger partitions.

Using this intuition about the clustering, we propose a bottom–up scheme that uses "predetermined" initial "seed" partitions, which subsequently are greedily concatenated, until the number of partitions becomes equal to the number of available broadcast disks. Our scheme requires us to make two decisions. The first concerns the sizes of the "seed" partitions and the second one concerns the criterion for performing partition merging.

The decision of the sizes of these "seed" partitions is crucial because it affects both the speed of the partitioning and the quality (in terms of average delay) of the generated program. To achieve a balance between these two factors, the sizes of these "seed" partitions are selected to be powers of 2, that is, $2^0, 2^1, 2^2, \ldots$.

The second decision regards the criterion for concatenating two successive partitions. Adopting a greedy approach we choose to merge the two successive partitions that result in the greatest reduction in the total average access delay.

Putting them together, we have the following algorithm for the *CascadedWebcasting*: initially, we generate the "seed" partitions $P_1^0, P_2^0, \ldots, P_v^0$. Then we perform $v-k$ *merging steps* ($v = \lfloor \log_2(n+1) \rfloor$). At the ith merging step ($1 \leqslant i \leqslant v-k$) the algorithm tries $v-i-1$ concatenations

and selects the one, which incurs the least expected cost. Due to the way the partitions are concatenated and due to the size of the initial "seed" partitions, it is obvious that at each step of the algorithm the size of a partition is always smaller than the size of its successive partition, (thus respecting Lemma 3 of [26]).

If the number n of items is not equal to $2^v - 1$, but it is $2^v - 1 < n(= 2^v - 1 + \beta) < 2^{v+1} - 1$ for some $\beta > 0$, then we treat the first $2^v - 1$ items with the procedure mentioned above and simply append the last β items to the last disk. Fig. 2 presents the pseudocode for the *CascadedWebcasting* scheme for broadcast program generation.

The above procedure requires that $v(= \lfloor \log_2(n+1) \rfloor) \geqslant k$, otherwise it will not be able to exploit all the k broadcast disks. Of course, there are some application environments where the above inequality does not hold, for instance when we have to broadcast, say, a hundred items over 7 six disks or in an even more extreme example when we have to broadcast 16 items over 10 channels. Although we can always find such an application environment, in the Webcasting application we expect that the number of items to be broadcasted will be very large and the number of disks fairly small. We have already presented in Section 1.1 that the number of items can grow to a few thousands. Moreover, due to the fact that a very large number of disks causes significant increase in the total expected delay, we expect that the number of disks (as suggested in [32], pp. 32) will vary between 2 and 5.

3.3. Time complexity of CascadedWebcasting

We will give the time complexity of the algorithm for two cases: (a) assuming minimal available storage space for intermediate results, and (b) assuming enough storage space to hold intermediate results.

Algorithm *CascadedWebcasting*(int n, int k, float vector $\mathcal{P}$)
// n: number of items, k: number of disks, $\mathcal{P}$: access probabilities
begin
sort($\mathcal{P}$)
$\nu = \lfloor \log_2(n+1) \rfloor$
create ν "seed" partitions, $P_1^0, P_2^0, \ldots, P_\nu^0$ with $size(P_i^0) = 2^{i-1}$
$P_l = \mathcal{P} - (P_1^0 \cup P_2^0 \cup \ldots \cup P_\nu^0)$
$P^0 = \{P_1^0, P_2^0, \ldots, P_\nu^0\}$ // the set of "seed" partitions
for($i = 1$; $i <= (\nu - k)$; $i = i + 1$) // perform $\nu - k$ merging steps.
{
  for($j = 1$; $j <= (\nu - i - 1)$; $j = j + 1$)
    $C^j = \{C_1^i(= P_1^{i-1}), \ldots, C_j^i(= \text{merge}(P_j^{i-1}, P_{j+1}^{i-1})), \ldots, C_{\nu-i-1}^i(= P_{\nu-i}^{i-1})\}$
  $P^i = \text{minDelay}(C^j)$ // The partition incurring the minimum delay
}
$P^{final} = \{P_1^{\nu-k}, \ldots, P_k^{\nu-k} \cup P_l\}$
return P^{final}
end

Fig. 2. The *Cascaded* scheme for Webcasting.

Minimal storage space. Under this setting the available space is large enough to hold only the vector $\mathscr{P}$ of access probabilities and a few numbers that indicate the partition borders. The space required for holding $\mathscr{P}$ is n. The maximum of space required to hold the partition borders during the execution of the *Webcasting* is v and the minimum is k. Under this model, at each *merging step* the algorithm needs to compute the probability sums of the items of every candidate partition (see Eq. (5)) in order to select the partition that gives the smallest access delay. This implies repeated access to the vector $\mathscr{P}$.

Since the number of passes over $\mathscr{P}$, is $\sum_{i=1}^{v-k}(v-i)$, we deduce that, with storage overhead equal to $n+v$:

Proposition 3.1. *The time complexity of the CascadedWebcasting scheme is* $\mathrm{O}\left(n * \frac{(\log(n)-k)(\log(n)+k-1)}{2}\right)$.

or

Proposition 3.2. *The time complexity of the CascadedWebcasting scheme is* $\mathrm{O}(n * (\log^2(n) - k^2))$.

Adequate storage space. Under this setting, apart from the space to store the vector $\mathscr{P}$ and the partition borders, we are allowed to store the sum of access probabilities of the individual partitions after each merging step and the number of items they contain. Thus, we need another $2 * v$ space (at most). Therefore, we need at most $n + 3 * v$ storage space. We realize that this setting is the realistic one, since it represents the typical way that the *Webcasting* algorithm would be implemented.

Initially, at a cost of $\Omega(n)$ we make one pass over $\mathscr{P}$ to compute the sums $(S_1^0, S_2^0, \ldots, S_v^0)$ of the access probabilities of the items belonging to each "seed" partitions $P_1^0, P_2^0, \ldots, P_v^0$. Thus, we reduce the vector $\mathscr{P}$ to another vector S of length v, where the sums S_i^0, $(1 \leqslant i \leqslant v)$ are stored. The *merging steps* of the *CascadedWebcasting* will run over this vector. Next, we try all candidate mergings. This costs $(v-1)^2$, since the size of the vector S at this point is $v-1$ (we have performed a partition merge) and there are $v-1$ candidates. We assume a $\mathrm{O}(1)$ to select the minimum cost candidate partition and a $\mathrm{O}(1)$ cost to compute the sum between two successive partitions S_i^0, S_j^0, $(i < j)$. This procedure (testing of candidate merges) will repeat for all the remaining steps of the algorithm until we have k partitions. In each successive step the size of the vector S shrinks at one position and so does the number of candidates. Thus:

Proposition 3.3. *The cost* $C_{\mathrm{merge}}^{\mathrm{casc}}$ *of partition merging is* $C_{\mathrm{merge}}^{\mathrm{casc}} = \sum_{i=1}^{k+1}(v-i)^2$.

It is obvious that

Proposition 3.4. *The complexity of the merging procedure is* $\mathrm{O}(k * v^2)$ *or* $\mathrm{O}(k * log^2(n))$.

and thus:

Proposition 3.5. *The complexity of the CascadedWebcasting is* $\mathrm{O}(n + k * log^2(n))$. *For most practical cases this cost will be dominated by the* $\mathrm{O}(n)$ *factor*.

4. Performance evaluation

In order to evaluate the performance of the algorithms, we used synthetic as well as real data. We used as performance measures the running times [3] of the algorithms and the average access delay, as determined by Eq. (5). We evaluated the following algorithms: Bucketing (*bucket*), CascadedWebcasting (*casc*), Growing Segments (*gs*), Data Based (*db*), Greedy (*Safok*) and Variant Fanout with the constraint K (VF^K). We present also the performance of the Flat (*flat*) broadcast scheme and of an Optimal (*opt*) Webcasting scheme. The *Flat* algorithm is the one that uses only one broadcast disk. The Optimal algorithm can be implemented either as a variation of the A^* search algorithm (described in [25]) or with the use of dynamic programming (described in [27]). Both versions of the Optimal scheme avoid the exhaustive enumeration of all possible partitionings. In this work we implemented the latter scheme because it is much faster. The performance of the *flat* (*opt*) acts as the upper (lower) bound on the average access delay of the algorithms. We do not report the running times of the Optimal, since they are more than one order of magnitude larger than the respective times of the slowest algorithm.

4.1. Experiments with synthetic data

In order to evaluate the performance of the algorithms over a wide range of data characteristics we generated synthetic data. We assume that the demand probabilities follow the Zipfian distribution. A similar setting was followed by other works as well (see [23,24,27,33]). The Zipfian distribution can be expressed as

$$p_i = \frac{(1/i)^{\theta}}{\sum_{i=1}^{n}(1/i)^{\theta}} \quad 1 \leqslant i \leqslant n, \tag{7}$$

where the parameter θ controls the *skewness* of the distribution. For $\theta = 0$ the Zipfian reduces to the uniform distribution, whereas larger values of θ derive increasingly skewer distributions. A plot of this function can be seen at Fig. 3. The generated access probabilities of the vector $\mathscr{P}$ were not ordered, so the reported execution time of the algorithms includes the time required to sort the vector $\mathscr{P}$. For all data sets that were tested, the cost of sorting the vector $\mathscr{P}$ did not exceed 0.01 s, which is an order of magnitude smaller than the execution time of *Growing Segments*, *Safok*, VF^K and of the *Data Based* method. Thus, the *sorting time is not the bottleneck on the performance of the algorithms*.

The parameters of our model that will be studied are the number k of broadcast disks, the skewness of the access probability and the size n of the database $\mathscr{D}$. A data set will be denoted as $Dv_DTv_TNv_N$, where v_D reveals the number of disks, v_T is the value of parameter θ and v_N is the number of data items. An underscore (_) instead of v_D, v_T, v_N will imply that we vary this parameter across a range of values. Our default setting will be $D4T0.91N4K$ denoting a database of 4096 items to be partitioned in four disks, where the value of θ is 0.91. For the *Growing Segments* we had to select a value for the *increment*. Since, [24] provides no method for setting the value of

[3] All experiments were carried out in a Pentium III 933 MHz with 512 MB RAM running under Windows 2000.

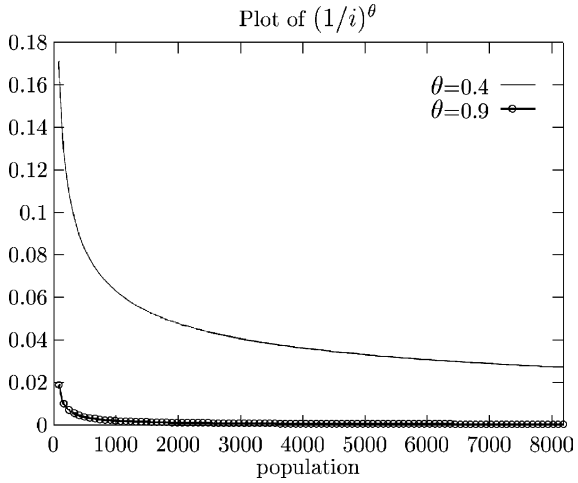

Fig. 3. Plot of the function used to generate synthetic data.

increment, we selected a value of 32, which is a moderate choice, and used it for all the experiments we conducted.

In the first set of experiments, we evaluated the performance of the algorithms with respect to the number of broadcast disks. The results can be seen at the top part of Fig. 4. The general observation is that in terms of average delay the *CascadedWebcasting* scheme performs very close (around 10%) to the *Optimal*. In particular, its performance deviates from the optimal (around 20%) only for the cases when either the number of disks is very small (2 or 3) or when it is very large (e.g., 9) compared the number of the initial "seed" partitions. This is expected because in the former case the partition points required are very few whereas in the latter case the algorithm does not have many opportunities to perform partition merges. Regarding the performance of the rest of the algorithms, we can see that they all perform equally well and their differences are not easily noticeable. This fact strengthens the main motive of our work, which states that slight improvements in the total expected delay do not favor one algorithm over the other, especially in the case of Webcasting where we deal with the "average" client and approximate knowledge of the item probabilities.

As long as the execution time of the algorithms is concerned, the *CascadedWebcasting* and the *Bucketing scheme* are the best and incur a constant cost of only a few milliseconds, practically independent on the number of disks. The performance of the *Growing Segments* methods grows linearly with the number of disks. Similarly, linear is the performance of VF^K and of the *Safok*, but with much smaller rate of increase. The time performance of the *Data Based* method presents a particularity: it reduces with increasing number of disks. Intuitively, this result can be explained by the fact that an increasing number of disks results in smaller ρ_i. This can also be seen by Proposition 2.7, where a decrease in ρ_i (implying also an increase in μ_i) results in smaller execution time.

The second set of experiments measures the scalability of the algorithms in terms of the size of the database $\mathscr{D}$. The results can be seen at the middle part of Fig. 4. As long as the average access delay is concerned, we observe that all algorithms incur an increase, which is linear in the database

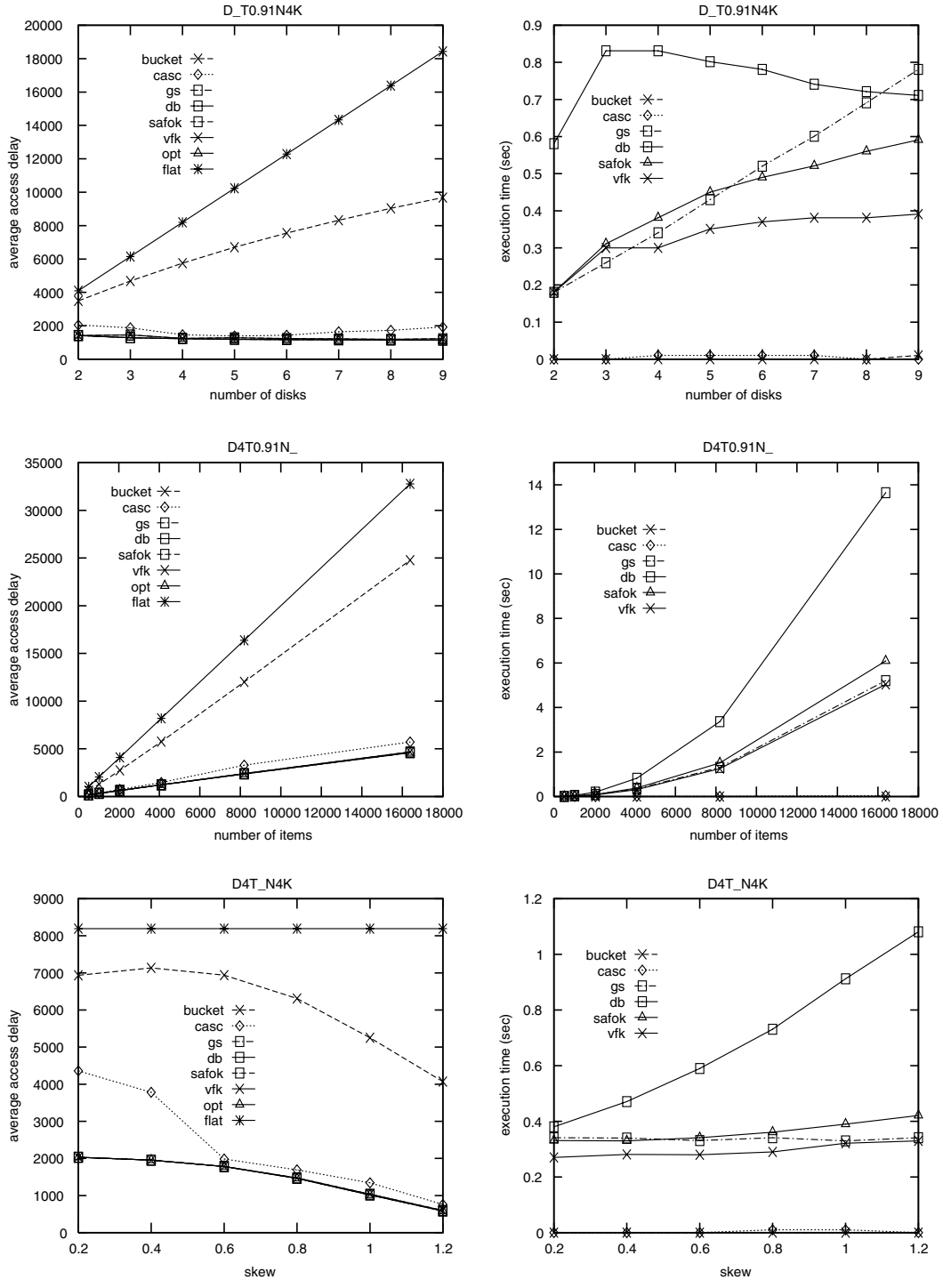

Fig. 4. *Left* average access delay, *Right* execution time.

size. This is expected since larger $\mathscr{D}$ implies a larger broadcast cycle. Apart from the *Bucketing* scheme, all the other algorithms incur alike average access delay. The deviation of the *Cascaded-Webcasting* from the optimal is approximately 15%.

As long as the time complexity is concerned, we can see that the *CascadedWebcasting* and the *Bucketing* scheme incur practically zero time cost. The cost of all the other methods increases parabolically with the database size, implying a dependence on the square of the number of items. It is interesting to note here that although Proposition 2.7 does not involve the square of the database size into the cost of *Data Based*, this is implicitly involved. For skewed access probabilities the value of ρ_i remains very high for all the phases of the algorithm, because few items are allocated to the first fast disks.

In the third set of experiments, we evaluated the sensitivity of the algorithms to the skewness of the access probabilities. The results can be seen at the bottom part of Fig. 4. In general, the access delay of all the schemes decreases with increasing value of θ. As expected, the performance of the *CascadedWebcasting* in not very good in the absence of skewness on the access pattern. But, when the skewness becomes apparent its performance improves considerably and approaches that of the other algorithms. We can see that there is "*characteristic*" value for θ (here is equal to 0.6), beyond which the performance of the *CascadedWebcasting* converges to the optimal one. For low values of skewness the performance of the *CascadedWebcasting* is around 50% from the optimal, but when the skewness becomes apparent this percentage drops to less than 10%. Again, the performance of the *Bucketing* scheme is the worst of all schemes.

In terms of execution time, the performance of the *Bucketing* and the *CascadedWebcasting* scheme is independent on the skewness. This is due to the fact that both algorithms make "prespecified" decisions. For the *Bucketing* scheme, this decision is embedded into the computation of δ, whereas for the *CascadedWebcasting* scheme this decision is embedded into the definition of the initial "seed" partitions $P_1^0, P_2^0, \ldots, P_v^0$. The performance of the *Growing Segments* is constant (ignoring some insignificant variance). This observation is consistent with Proposition 2.4. The execution time of *Data Based* grows linearly with increasing skewness. This can be explained by Proposition 2.7. The term ρ_i increases with increasing skewness, whereas the term μ_i decreases with increasing skewness. Finally, the execution time of VF^K grows quadratically with a very small multiplicative constant. We performed an experiment with the dataset D4T0.91N4K and measured the average size of the disk that is split at each phase of VF^K as a function of skewness. The results are depicted in Table 1. Similar to the VF^K, are the observations for the *Safok*, although this algorithm incurs a larger cost, since at each split decision it makes some more "bookkeeping".

Table 1
The average size of the disk that is split by VF^K at each phase

θ	Average examined disk size
0.2	2024.8
0.4	2026.8
0.6	2036.4
0.8	2139.2
1.0	2339.8
1.2	2392.4
1.4	2420.4

In light of the above results, we conclude that the *CascadedWebcasting* and the VF^K method are the most *robust*, managing to achieve average access delay very close to the optimal one and small execution time.

4.2. What if $n \neq 2^{\lfloor \log(n+1) \rfloor}$?

In this subsection, we investigate the impact of our decision to treat in the partitioning phase of the *CascadedWebcasting* only a part of the data, when the number of database items is not equal to $2^v - 1$, and append the rest to the last (slowest) disk. We examined the interaction of this decision with both the number of items that do not participate in the partitioning phase and with the varying skewness. We also give the performance of the *Optimal* and of the VF^K scheme.

Fig. 5 demonstrates two representative cases of the impact of this decision. The left part of the figure shows the results for a database of 3000 items with varying skewness in access probability (dataset D4T_N3000) and the right part of the figure shows the results for fixed skewness ($\theta = 0,91$) for a database of varying size (dataset D4T0.91N_). The *CascadedWebcasting* scheme in these cases will seek for a partitioning of the first 2047 items and will append the rest of the items to the last disk.

The left part of the figure is similar to the respective part of Fig. 4. The only difference is between the "*characteristic*" value for θ, which is now larger (0.8 vs. 0.6). Thus, we can understand that even though we ignore some database items during the partitioning phase and append them lastly, the existence of skewness in the access pattern prevents serious deteriorations to the performance of the *CascadedWebcasting*.

From the right part of the figure we observe that although the performance of the *CascadedWebcasting* diverges from the optimal with increasing number of ignored items, it manages to maintain very good average access time even in the case that ignores almost half of the database items (process 2047 items and appends the rest 1953($= 4000 - 2074$)). This can be explained from the fact that the contribution of the last items to the total access delay is negligible (see Section 3).

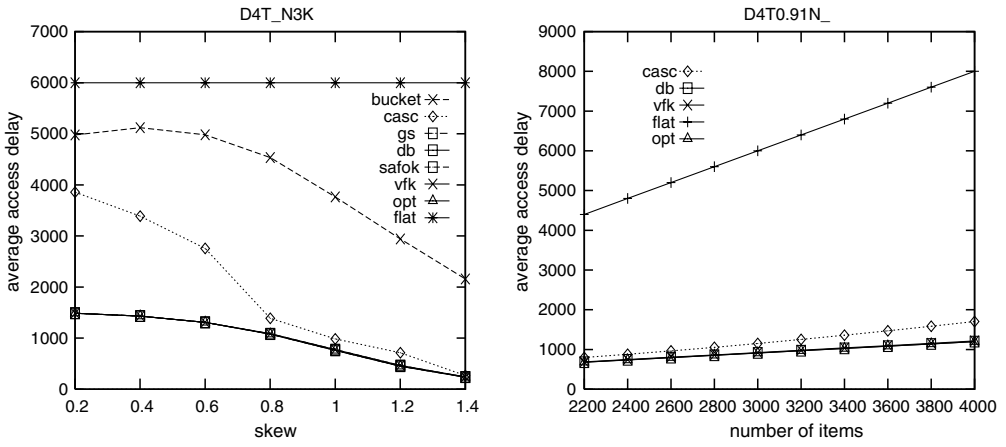

Fig. 5. Average access delay as: (Left) a function of the skew when large number of items is ignored and (Right) a function of the number of items ignored.

It is easily understood that the divergence from the optimal will be smaller in larger databases (e.g., $|\mathscr{D}| = 8500$) because in these cases the contribution of the last items is even smaller.

4.3. Experiments with real Web data

In order to confirm the performance of the algorithms with real data, we conducted an experiment with a Web server trace. Moreover, the interest in validating our results with real data lies in the fact that the access probabilities in real traces sometimes are not modelled very good with the Eq. (7). We experimented with data available at http://ita.ee.lbl.gov/html/traces.html. Specifically, we used the ClarkNet trace. We used the first week of requests and we cleansed the log file (e.g., by removing CGI scripts, staled requests, etc.). Then, we removed all the items whose *support* (normalized number of appearances in the trace) was less than 0.00001. After this pre-

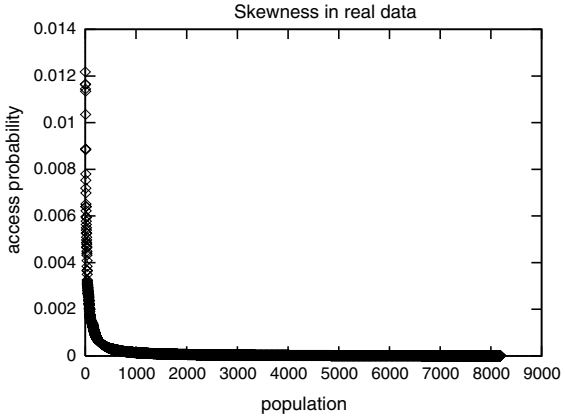

Fig. 6. Access probabilities for the items of the Web server trace.

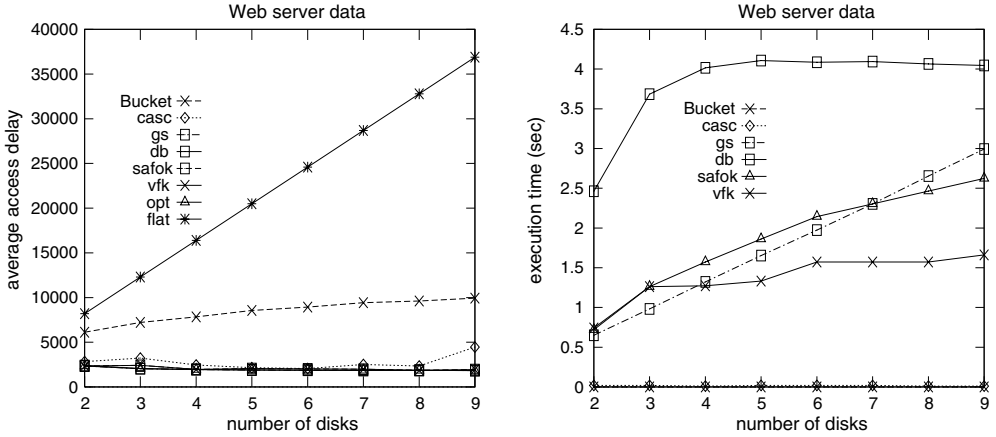

Fig. 7. *Left* access delay, *Right* execution time for real data.

processing, remained around 8300 items. We discarded some of them so as to keep only the first $8192(=2^{14})$ items. The plot of the probability distribution of the items is shown in Fig. 6. Observe the striking similarity of this plot with that shown in Fig. 3.

The results of this experiment are depicted in Fig. 7. In general, the results are very similar with that illustrated at the top part of Fig. 4. We can see that the performance of the *CascadedWebcasting* scheme with respect to average access time is not very good only for the case of a large number of disks (9), but becomes almost identical to the best performing schemes for all other number of disks. A large number of disks does not give many alternatives for performing disk merges to the *CascadedWebcasting* scheme. The observations for the time performance of the scheme is similar with those reported in Section 4.1.

5. Conclusions

We considered the problem of creating Webcasting programs for push-based environments when the items are equi-sized and their access probabilities are known. When the size of the items are not equal, techniques such as those presented in [27] can be adopted. Based on the *broadcast-disks* paradigm, we identified the need for scalable in terms of database size algorithms. No prior work exists that addressed the issue of scalable algorithms for push-based environments.

We analyzed the time complexity of the existing algorithms and showed their dependency on the database size and/or the skewness of the access probabilities.

We proposed a new algorithm, *CascadedWebcasting*, which is designed to be fast in execution and to take advantage of the skewness in access probabilities. Its execution time is almost linear in the number of transmitted items and the average access delay it incurs, is very close to that of the optimal algorithm.

Using a synthetic data generator, the performance of *CascadedWebcasting* was compared against that of *Bucketing*, *Growing Segments*, *Data-Based*, *Greedy* and *Variant fanout with the constraint K* and also that of the two baseline algorithms, *Flat* and *Optimal*. Our experiments showed that *CascadedWebcasting incurs the smallest execution time*, *never more than* 0.02 seconds. Its performance (in execution time) is not affected by the number of broadcast disks or the skewness of the access pattern. The execution time of *Greedy*, *Variant fanout with the constraint K*, and *Growing Segments* grows linearly with the number of disks, whereas the running time of *Greedy*, *Variant fanout with the constraint K*, and *Data Based* grows linearly with the skewness. Moreover, *CascadedWebcasting* achieves constant execution time (<0.02 seconds) for large databases comprised by a few thousands items, whereas the execution time of all the other algorithms (except from *Bucketing*) grows quadratically with the database size.

In addition, *CascadedWebcasting* is very robust and capable of producing hierarchical broadcast programs with average access time, which is very close to the optimal one. *The average access delay it incurs is never more than 15%* (*on the average*) *from that of the optimal algorithm.*

Finally, we confirmed all the above results using real data, taken from the trace file of a Web server.

In summary, *CascadedWebcasting* is a very fast, robust and efficient algorithm for creating hierarchical Webcasting programs for large-scale push-based delivery.

References

[1] O. Ercetin, L. Tassiulas, Push-based information delivery in two stage satellite-terrestrial wireless systems, IEEE Transactions on Computers 50 (5) (2001) 506–518.
[2] P. Rodriguez, E. Biersack, Bringing the Web to the network edge: large caches and satellite distribution, ACM/Kluwer Mobile Networks and Applications 7 (2002) 67–78.
[3] E. Pitoura, G. Samaras, Data Management for Mobile Computing, Kluwer Academic Publishers, 1998.
[4] S. Acharya, R. Alonso, M. Franklin, S.B. Zdonik, Broadcast disks: data management for asymmetric communications environments, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), 1995, pp. 199–210.
[5] J.W. Wong, Broadcast delivery, Proceedings of the IEEE 76 (12) (1988) 1566–1577.
[6] M. Franklin, S. Zdonik, Data in your face: Push technology in perspective, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), 1998, pp. 516–519.
[7] M. Franklin, Client Data Caching: A Foundation for High Performance Object Database Systems, Kluwer Academic Publishers, 1996.
[8] M. Franklin, S. Zdonik, A framework for scalable dissemination-based systems, in: Proceedings of the ACM International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA), 1997, pp. 94–105.
[9] M. Cherniack, M. Franklin, S. Zdonik, Expressing user profiles for data recharging, IEEE Personal Communications 8 (4) (2001) 32–38.
[10] A. Alber, Videotex/Teletext: Principles and Practices, McGraw-Hill, 1985.
[11] D. Gifford, Polychannel systems for mass digital communications, Communications of the ACM 33 (2) (1990) 141–151.
[12] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, A. Weinrib, The Datacycle architecture, Communications of the ACM 35 (12) (1992) 71–81.
[13] T. Liao, Global information broadcast: an architecture for internet push channels, IEEE Internet Computing 4 (4) (2000) 16–25.
[14] SkyCache, http://www.skycache.com.
[15] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, in: Proceedings of the IEEE Conference on Computer Communications (INFOCOMM), 1999, pp. 126–134.
[16] S. Anily, C. Glass, R. Hassin, The scheduling of maintenance service, Discrete Applied Mathematics 82 (1998) 27–42.
[17] C.-J. Su, Tassiulas, Joint broadcast scheduling and user's cache management for efficient information delivery, ACM/Baltzer Wireless Networks 6 (2000) 137–147.
[18] E. Pitoura, P. Chrysanthis, Exploiting versions for handling updates in broadcast disks, in: Proceedings of 25th International Conference on Very Large Data Bases (VLDB), 1999, pp. 114–125.
[19] S. Acharya, M. Franklin, S.B. Zdonik, Prefetching from a broadcast disk, in: Proceedings of the IEEE Conference on Data Engineering (ICDE), 1996, pp. 276–285.
[20] L. Tassiulas, C. Su, Optimal memory management strategies for a mobile user in a broadcast data delivery system, IEEE Journal on Selected Areas in Communications 15 (7) (1997) 1226–1238.
[21] S. Khanna, V. Liberatore, On broadcast disk paging, SIAM Journal on Computing 29 (5) (2000) 1683–1702.
[22] T. Imielinski, S. Viswanathan, B. Badrinath, Data on air: organization and access, IEEE Transactions of Knowledge and Data Engineering 9 (3) (1997) 353–372.
[23] N. Vaidya, H. Sohail, Scheduling data broadcast in asymmetric communication environments, ACM/Baltzer Wireless Networks 5 (3) (1999) 171–182.
[24] J.-H. Hwang, S. Cho, C.-S. Hwang, Optimized scheduling on broadcast disks, in: K.-L. Tan, et al. (Eds.), Proceedings of the International Conference on Mobile Data Management (MDM), Vol. 1987 of Lecture Notes in Computer Science, Springer-Verlag, 2001, pp. 91–104.
[25] W.-C. Peng, M.-S. Chen, Efficient channel allocation tree generation for data broadcasting in a mobile computing environment, ACM/Kluwer Wireless Networks 9 (2) (2003) 117–129.

[26] C.-H. Hsu, G. Lee, A. Chen, A near optimal algorithm for generating broadcast programs on multiple channels, in: Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), 2001, pp. 303–309.
[27] W. Yee, S. Navathe, E. Omiecinski, C. Jermaine, Bridging the gap between response time and energy-efficiency in broadcast schedule design, in: Proceedings of the International Conference on Extending Data Base Technology (EDBT), Vol. 2287 of Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 572–589.
[28] D. Aksoy, M. Franklin, R×W: a scheduling approach for large-scale on-demand data broadcast, IEEE/ACM Transactions on Networking 7 (6) (1999) 846–860.
[29] K. Stathatos, N. Roussopoulos, J.S. Baras, Adaptive data broadcast in hybrid networks, in: Proceedings of 23rd International Conference on Very Large Data Bases (VLDB), 1997, pp. 326–335.
[30] J. Yu, T. Sakata, K.-L. Tan, Statistical estimation of access frequencies in data broadcasting environments, ACM/Baltzer Wireless Networks 6 (2) (2000) 89–98.
[31] C. Kenyon, N. Schabanel, N. Young, Polynomial-time approximation scheme for data broadcast, in: Proceedings of the 32nd ACM Symposium on the Theory of Computing (STOC), 2000, pp. 659–666.
[32] S. Acharya, Broadcast Disks: Dissemination-based Data Management for Asymmetric Communication Environments, Ph.D. Thesis, Department of Computer Science, Brown University, May 1998.
[33] C.-J. Su, L. Tassiulas, V. Tsotras, Broadcast scheduling for information distribution, ACM/Baltzer Wireless Networks 5 (2) (1999) 137–147.

Dimitrios Katsaros was born in Thetidio-Farsala, Greece in 1974. He received a B.Sc. in Computer Science from the Aristotle University of Thessaloniki, Greece (1997). He spent a year (July 1997–June 1998) as a visiting researcher at the Department of Pure and Applied Mathematics at the University of L'Aquila, Italy. Currently, he is a Ph.D. candidate at the Computer Science Department of Aristotle University. His research interests include Web databases, semistructured data and mobile data management. He is an editor of the book Wireless Information Highways, which will be published by IDEA Inc. in 2004.

Yannis Manolopoulos was born in Thessaloniki, Greece in 1957. He received a B.E. (1981) in Electrical Engineering and a Ph.D. (1986) in Computer Engineering, both from the Aristotle University of Thessaloniki. Currently, he is Professor at the Department of Informatics of the latter university. He has been with the Department of Computer Science of the University of Toronto, the Department of Computer Science of the University of Maryland at College Park and the University of Cyprus. He has published over 130 papers in refereed scientific journals and conference proceedings. He is co-author of a book on "Advanced Database Indexing" and "Advanced Signature Indexing for Multimedia and Web Applications" by Kluwer. He is also author of two textbooks on Data Structures and File Structures, which are recommended in the vast majority of the computer science/engineering departments in Greece. He served/serves as PC Co-chair of the 8th Panhellenic Conference in Informatics (2001), the 6th ADBIS Conference (2002) the 5th WDAS Workshop (2003), the 8th SSTD Symposium (2003) and the 1st Balkan Conference in Informatics (2003). Also, currently he is Vice-chairman of the Greek Computer Society. His research interests include access methods and query processing for databases, data mining, and performance evaluation of storage subsystems. For more information, please visit http://delab.csd.auth.gr/~manolopo/yannis.html.