Towards Realizable, Low-Cost Broadcast Systems for Dynamic Environments

Christos K. Liaskos, *Member, IEEE*, Sophia G. Petridou, *Member, IEEE*, and Georgios I. Papadimitriou, *Senior Member, IEEE*

***Abstract*—A main design issue in a wireless data broadcasting system is to choose between push-based and pull-based logic: The former is used as a low-cost solution, while the latter is preferred when performance is of utmost importance. Therefore, the most significant advantage of a push system is the minimal cost. This fact implies that hardware limitations do exist in the case of push systems. As a consequence, every related proposed algorithm should primarily be cost-effective. This attribute, however, has been overlooked in related research. In this paper, popular broadcast scheduling approaches are tested from an implementation cost aspect, and the results render them only conditionally realizable. Moreover, a new, cost-effective, adaptivity oriented schedule constructor is proposed as a realistic, minimal-cost solution.**

***Index Terms*—Adaptivity, analysis, CPU-memory cost, data serialization, push systems.**

I. Introduction

RECENT years have witnessed the wide-spreading use of wireless push-based systems. Simple in architecture and implementation, lightweight and energy efficient—especially from the client's point of view and both hardware- and software-wise—the push-based approach has been adopted for use in a variety of information dissemination applications and has been incorporated in almost every single mobile telecommunications device. Popular uses include airport and hospital information systems, instant messaging services, and multimedia on demand over the Internet or cellular networks. Consequently, the on-growing interest of the telecommunications industry has spurred the research on the performance optimization of these systems.

Pure push-based systems typically employ a central server continuously transmitting data through a channel, while the clients simply retrieve useful data from the stream, without being able to perform any kind of queries. Pull-based systems on the other hand adhere to the classical client–server scheme, where the server transmits only the data that have been requested by a client, and only on the event of a request. While the functionality separation is clear, a common misconception lies to the reasons that render a pull or a push choice more suitable for a telecommunications system case. It must be clarified that a typical pull-based scheme generally performs better, response time-wise. However, the push-oriented systems have one unique advantage: very low implementation cost with regard to scalability.

The reader is encouraged to visualize a push server as a device of the scale of a standard, mainstream computer at most. It is therefore obvious that such a device has limited computational power and caching capabilities. Thus, the computational power and required memory should be attributes of the highest impact factor when designing broadcasting-related algorithms. Network congestion minimization—high due to the continuous data emission scheme of push systems—should also be a major concern. However, to the best of our knowledge, this has not been the case so far in related research.

A. Related Work

Research on the field of broadcast systems can be roughly split into two main categories: the mathematical foundation and analysis attempts of the broadcast problem in general, and algorithmic approaches that tend to provide simplified, algorithmically applicable solutions of the general problem and/or examine other advanced aspects. In the definition of the data broadcast problem, each to-be-broadcasted item is assigned a generic "broadcast weight," which abstractedly represents the impact of broadcasting on system resources. The classic problem is then to find a schedule over an infinite-time horizon so that the mean client waiting time as well as the mean broadcast weight are minimized. In this context, the problem is proved to be NP-hard [1]. [2] discusses the generalized maintenance problem (a known NP-hard problem) and proves that the broadcasting of equally sized items is a subcase of it. A lower bound for the client's mean waiting time is also provided. [3] proves the existence of a possible solution for teletext systems and also defines a lower bound for the client's mean waiting time in this case.

From this point on, related research ignores broadcast weights. In this case, uniform and nonuniform item sizes are discussed. [4] and [5] present the lower bound for the client mean waiting time in the case of both uniform and nonuniform sizes. A scheduling algorithm is also defined, which even today achieves the minimum client waiting times. The lower bound in the nonuniform case is not tight though, and this case is further analyzed in [1]. [6] introduces a "reservation" system that favors client waiting time in the case of big-sized items.

Retaining the no-broadcast weights consideration, several algorithmic approaches have been proposed to simplify the data broadcasting problem, the most influential of all being the Broadcast Disks model [7], which is also discussed in this paper. A great deal of work has been done based on this model,

Manuscript received July 30, 2009; revised April 08, 2010; accepted July 21, 2010; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. Rodriguez.

The authors are with the Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece (e-mail: cliaskos@csd.auth.gr; spetrido@csd.auth.gr; gp@csd.auth.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2010.2062534

studying data prefetching, caching [8] and indexing [9], [10], hybrid data broadcasting [11], as well as scheduling strategies and noise interference [12], [13].

Another branch studies data broadcasting from a higher level, transaction aspect (e.g., queries that must be answered in a limited amount of time [14], among other limitations) and was introduced in [15]. This approach has since been extended with fail-over tactics [16] and techniques [17] that increase the concurrency of transactions.

Finally, other recent studies deal with the case of correlated client queries, e.g., complex queries requesting multiple single data items [18]. [19] deals with the case of requesting successive single items, while [20] and [21] examine the case of random access order of single items. [22] and [23] present algorithms that achieve the lower bounds of average access time in the case of broadcasting a pair of files.

B. Authors' Related Work

The authors have so far contributed in both analytical and algorithmic approaches. In [24], an effort to combine clustering techniques with the broadcast disks model was made, with satisfactory results. The analytical approach of [25] added means of projecting the optimal parameters required to construct the broadcast schedule. The cost parameter was introduced in [26], targeting the cost efficiency of the scheduling procedure. Moreover, research on adaptive push systems has been carried out in [24] and [27]–[29].

C. Contribution

As it is evident from the brief presentation of the related work on the field, every study focuses on a specific subject of data broadcasting without however taking the implementation cost into consideration. Thus, a good amount of algorithms have been proposed, which may be hard to be implemented in the real world. Moreover, the classic system model of [7] employed by all related studies is simulation oriented and does not claim any effectiveness in cost assessment tasks.

In the present paper, a more realistic system model is presented, designed to augment the network's functionality through the addition of Web connectivity, distinction between broadcast schedulers and beacons, and broadcast schedule lifespan parameters. The new model is then used to perform cost assessment studies of broadcast scheduling techniques, aiming at the following:

1) the demonstration of the fact that implementation cost assessments are critical for algorithms related to push systems;
2) the testing of adaptivity capabilities of popular broadcast scheduling approaches;
3) the presentation and testing of a new cost-effective, adaptivity-oriented broadcast schedule constructor algorithm.

The proposed scheduling scheme is compared to the analytically optimal [5] and modern, broadcast disks-based scheduling algorithms. Results indicate that the novel scheme combines optimal performance and minimal cost, while the compared algorithms are even conditionally realizable in several cases.

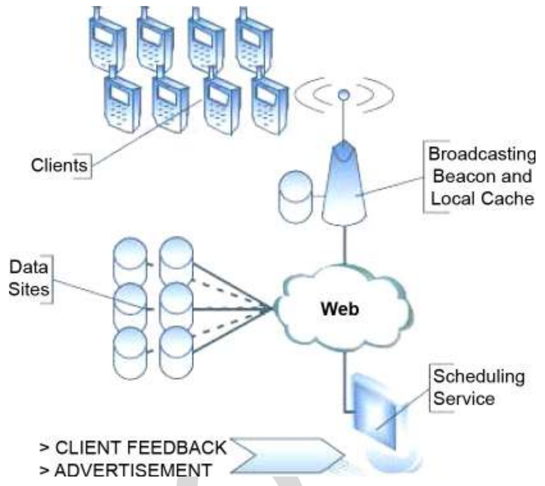

Fig. 1. Novel broadcasting model.

D. Summary of Contents

Section II provides the basics for understanding the push system's model, terms, architecture, and operation. In Section III, the broadcast scheduling procedure is theoretically analyzed. In Section IV, the proposed scheduling algorithm is presented. Performance assessment configurations and results are given in Section V. Our conclusion is given in Section VI.

II. Broadcasting Basics

A. Network Topology and Operation: Advancing the Classic Model

The proposed topology for a realistic push-based system is depicted in Fig. 1. A set of clients receives a common broadcast schedule emitted from a central source.

Concerning the clients, the following is commonly assumed.

- They share common information needs and preferences with regard to their topological distribution [28].
- Their number is large enough to rule out a point-to-point-based style of communication from being a viable option with regard to implementation cost.

The broadcast source emits series of data items—better known as "pages"—according to a schedule that minimizes the clients' mean waiting time. In the classic system model, the server is standalone, i.e., unplugged from any network and self-sustaining regarding the origin of the broadcast data [1]–[23]. However, this scheme was introduced mostly for conceptual purposes [7] and is of limited practical use. It is safer to assume that the data originates from independent and distributed Web sources, as shown in Fig. 1.

The classic model also considers the scheduling service to be located at the beacon. While a beacon-local cache may be useful, placing the scheduler at the beacon would require costly changes at the totality of the base stations of a modern cellular network, which tend to act as simple terminals. It is thus more practical to consider the broadcast beacon and scheduling service being at separate locations.

In any case, however, the scheduling service is assumed to know [1]–[23]:

- which pages the clients need;
- the popularity of each data item.

This knowledge can be attributed to timely speculations, forecasting, client feedback, or data advertising.

Once constructed, the broadcast schedule has limited lifespan. After a certain time interval, it must be renewed to match the clients' demands. At this point, the scheduling service must:

- schedule data fetching over the Web;
- schedule storage at the beacon-local cache;
- enter a state of client preference monitoring in order to prepare for the next renewal.

B. Entry Point of the Present Work

The present work assumes that the system has just entered a renewal state and aims to produce the optimal schedule:

- as fast as possible on mainstream hardware;
- minimizing the caching needs at the beacon-local point;
- minimizing the required communication lifetime between the scheduling service and the data sites, thus limiting the required operational bandwidth and the dependence on network congestion.

As it will be shown, these goals can be effectively reached through the minimization of the broadcast schedule size.

C. Theoretical Overview: Ideal Scheduling

Input of the Ideal Schedule Constructor (ISC) are the pairs $p_i : \{l_i, u_i\},\ i = 1 \ldots N$, where p_i denotes the ith page of the server's database (arbitrarily enumerated), l_i the page's size, and u_i the page's speed (i.e., the the number of occurrences of p_i inside the broadcast schedule, BS). N denotes the total number of pages eligible for broadcasting. The length of the broadcast schedule is denoted by L, and it stands that

$$L = \sum_{i=1}^{N} u_i \cdot l_i. \tag{1}$$

The objective of the ISC is simple: arrange the occurrences of each page so as to achieve the minimum mean client waiting time. It has been shown that this criterion is satisfied only when the time interval between same page occurrences is steady, i.e., periodic schedule [7].

For arbitrary page speeds u_i and access probabilities π_i, the mean client waiting time is

$$\bar{D} = \frac{L}{2} \sum_{i=1}^{N} \frac{\pi_i}{u_i}. \tag{2}$$

The optimal page speeds are strictly defined through analysis [5]

$$u_i^{\text{ideal}} = L \cdot \sqrt{\frac{\pi_i}{l_i}} \frac{1}{\sum\limits_{i=1}^{N} \sqrt{\pi_i \cdot l_i}}, \quad i = 1 \ldots N \tag{3}$$

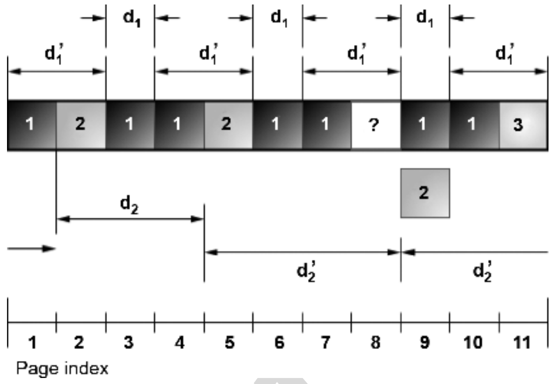

Fig. 2. Example of conflicting page spacings d_i.

which result into a minimal waiting time of

$$\overline{D_{\min}^{ideal}} = \frac{\left(\sum\limits_{i=1}^{N} \sqrt{\pi_i \cdot l_i}\right)^2}{2} \tag{4}$$

where (2)–(4) are only valid when $\frac{l_i}{L} \approx \frac{l_j}{L}\ \forall i,\ j$ [5].

D. Forced Deviations From Ideality

Two factors hinder strict appliance of the ideal scheduling scheme.

- A forced deviation stems from (4), which promises independence from the BS length L. However, as shown in [5], (3) yields noninteger results, and thus a form of rounding must be applied. The page speeds are thus altered, and the real mean waiting time is given by (2), which shows dependence from L.
- Creating a periodic broadcast schedule with constant $d_i = \frac{L}{u_i}$ spacings [5] between consecutive occurrences of a page may be impossible due to collisions. For example, consider a BS comprising three pages, $p_{1\ldots3}$, with corresponding speeds $u_{1\ldots3} = \{7, 3, 1\}$ and $l_1 = l_2 = l_3 = 1$. The ideal spacings should then be $d_1^{ideal} = \frac{L}{u_1} = \frac{11}{7}$, $d_2^{ideal} = \frac{L}{u_2} = \frac{11}{3}$, and $d_3^{ideal} = \frac{L}{u_3} = 11$, given that $L = \sum_{i=1}^{3} u_i \cdot l_i = 11$. The problem of integer divisions is evident, but suppose that in order to overcome it, we choose varying spacing values. For example, $u_1 = \{d_1 \times 3 + d_1' \times 4\}$, where $d_1 = 1$ and $d_1' = 2$; $u_2 = \{d_2 \times 1 + d_2' \times 2\}$, where $d_2 = 3$ and $d_2' = 4$; and $u_3 = \{11 \times 1\}$, where $d_3^* = 11$. As shown in Fig. 2, the spacings collide at the ninth position.

III. Impact of Deviations From Ideality

In this section, the form of deviation that has the smallest impact on the performance of a scheduling algorithm is studied.

A. Altering the BS Length: Cost-Efficiency and Performance Issues

As described in Section II-D, the real mean waiting time of the clients is dependent on the BS length L.

Let $L_I = \sum_{i=1}^{N} u_i$ be the BS length measured in pages, $L_I \in [N, L_I^{\max}],\ L_I \in Z$, N be the total number of pages, and $L_I^{\max}$

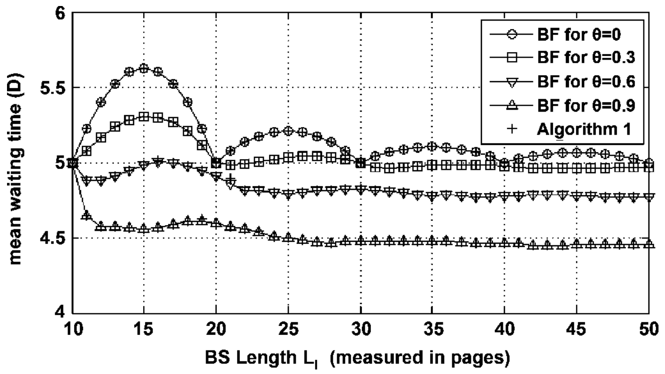

Fig. 3. Mean waiting time $\bar{D}$ as a function of the broadcast schedule length L_I measured in pages, as produced by brute force (BF) and Algorithm 1. Zipf pdf is assumed with $\theta = [0,\ 0.3,\ 0.6,\ 0.9]$. Number of items is $N = 10$.

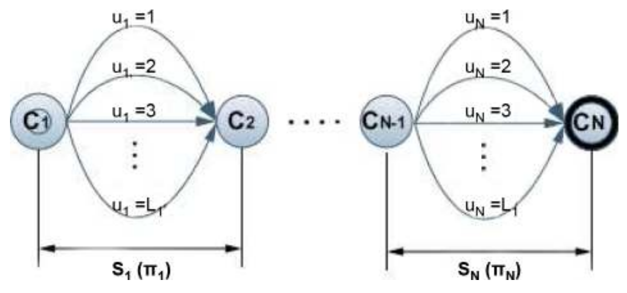

Fig. 4. Analogy of the BS length minimization with the optimal vehicle routing problem. A vehicle traverses a series of checkpoints $C_i,\ i = 1 \cdots N$, located at $S_i = \pi_i$ normalized distance units away from each other. Starting at C_1 with fuel L_I, the vehicle must reach C_N as soon as possible with the available fuel, making proper speed choices along the way. For simplicity, the fuel consumption is assumed to be proportional (1:1) to the chosen speed.

be the maximum examined value. For $L_I = N$, the BS is flat, and the speed of each page unary. For every $L_I > N$, exactly

$$P = L_I - N \tag{5}$$

unary speed increments must take place. The objective is to distribute the P increments optimally among the pages.

Since the ideal speeds u_i^{ideal} are given by (3), a serial distribution of increments can be employed, where the next unary raise affects the page p_j for which it holds that

$$\left|u_j - u_j^{\text{ideal}}\right| > \left|u_i - u_i^{\text{ideal}}\right| \qquad \forall i \neq j. \tag{6}$$

The procedure is formulated as Algorithm 1. Implementation with strict weak ordered maps [30] provides $O(\log N)$ complexity.

Algorithm 1 Calculating Optimal u_i for Given L_I

```
Input: The pages $p_i : \{\pi_i\},\ i = 1 \ldots N$ and the BS length $L_I \geq N$.
Output: The optimal page speeds $u_i^{\text{opt}}$.
1:  Set $P = L_I - N$.
2:  Set $u_i^{\text{opt}} = 1,\ i = 1 \ldots N$.
3:  Set $u_i^{\text{ideal}},\ i = 1 \ldots N$ by (3).
4:  for $i := 1$ to $P$ do
5:      Calculate page index $j$ by (6).
6:      $u_j^{\text{opt}} = u_j^{\text{opt}} + 1$
7:  end for
```

While the serial distribution is not analytically optimal, it typically achieves 99%–100% of the best possible performance. To demonstrate this fact, in Fig. 3, Algorithm 1 is compared $\bar{D}$-wise to the optimal results produced by brute force, i.e., trying all possible combinations of speeds and choosing the one achieving the minimum $\bar{D}$. The results coincide, with insignificant exceptions.

An interesting remark stems from the fact that the system quickly reaches a point $(L_I \approx 3 \cdot N)$, where 95%–100% of the possible waiting time has been achieved. This holds for any number of pages and any pdf***["probability density function"? Pls define]***: results are identical to that of Fig. 3. The system then tends to oscillate around the minimum of (4). The oscillation fades as pdf skewness and L_I increases.

The aforementioned remark has a direct impact on the overall cost of the system. The generic consequences can be better understood if we map the scheduling problem to the case of optimal vehicle routing described in Fig. 4. Minimizing L essentially translates to reaching C_N at minimal delay $D = \sum_{i=1}^{N} \Delta t_i = \sum_{i=1}^{N} \frac{s_i}{u_i}$—notice the similarity to (2)—with only a fraction of the required fuel. Further increasing L_I is an ineffective waste of resources.

The broadcasting-specific cost-related consequences include the following.

- The smaller the BS, the less computing time/processing power is required to construct it.
- A minimal BS has a higher probability of being calculated and sent from the scheduler to the beacon in one pass. A large BS must be calculated and forwarded in chunks, thus being constantly dependent on network congestion and requiring extra operational bandwidth.
- A small BS favors the implementation of cheap, dumb beacons that simply cache the binary data of the whole schedule without need for added complexity imposed by databases, data organization, and indexing/caching techniques.
- A BS small enough to be disclosed to the beacon in one brief pass in the form of indexes enables better scheduling of data fetching from the Web sites to the beacon.
- Minimal schedules may enable direct storage to RAM. Such a scheme is far less hardware-intensive than constant hard-disk accesses.

B. Page-Spacing Deviations: Altering the Pages' Speeds

The next form of deviation from ideality is that of page-spacing collisions. A purely theoretical way of overcoming this problem would be to alter the page speeds. However, as first shown in [5], the ideal page speeds already need to be tampered with in order to revert to an integer form. Moreover, as shown in Fig. 3, slight BS length alterations that can be caused by page-speed manipulation may degrade the mean waiting time. Thus, further tampering with the page speeds must be avoided as ineffective and performance degrading.

C. Impact of Local Concentrations of Single-Page Occurrences in the BS on the Client Waiting Time

As a next step, we examine whether the optimal scheduling algorithm should incline toward slight but distributed or major but local deviations of page spacings from their ideal values.

Consider a BS with fixed length L, in which a page p_i is repeated u_i times in total, with spacings

$$d_{ij} = d_{i1} \cdot a^{j-1}, \quad \{j = 1 \dots u_i,\ a > 1\} \tag{7}$$

where j denotes the spacing between the jth and $(j-1)$th page occurrence. If $\frac{l_i}{L} \to 0$, it will hold that

$$L = \sum_{i=1}^{u_i} d_{ij} = d_{i1} \frac{a^{u_i} - 1}{a - 1}. \tag{8}$$

The mean delay time attributed to this page will be

$$\overline{D(p_i)} = \sum_{j=1}^{u_i} \frac{d_{ij}^2}{2L} \overset{\text{eq. (7)}}{=} \frac{1}{2L} d_{i1}^2 \frac{a^{2u_i} - 1}{a^2 - 1} \tag{9}$$

and because of (8)

$$\overline{D(p_i)} = \frac{L}{2} \frac{(a-1)(a^{u_i} + 1)}{(a+1)(a^{u_i} - 1)}. \tag{10}$$

It is trivial to show that (10) is a strictly rising function of a. Thus for

$$a' > a \Leftrightarrow \overline{D_{p_i}(a')} > \overline{D_{p_i}(a)}. \tag{11}$$

Since higher values of a indicate more major and local spacing deviations (L is fixed, and (7) represents a geometric progression), we conclude through (11) that slight, distributed, global spacing deviations should be preferred over sharp deviations of distinct page speeds.

D. Slight Page-Spacing Deviations versus Zero Padding

So far, it has been proved that tampering with the page speeds, in order to overcome the problems posed by the forced deviations from ideality, is ineffective and potentially performance-degrading. Instead, slight and distributed spacing deviations should be preferred. Introduced in [7], however, zero padding has been adopted to produce periodic BS with steady page spacings. The logic behind this approach is simple: Increase the BS length by a sufficient number of "dummy" pages in order to overcome the deviations problem, and then substitute them with e.g., the most popular of pages. Thus, the new question posed is whether zero padding should be preferred over distributed slight spacing deviations.

To this end, we consider a single page with spacings $d_1 = \left\lfloor \frac{L}{u_1} \right\rfloor$ and $d_1' = \left\lceil \frac{L}{u_1} \right\rceil$, similarly to the case depicted in Fig. 2. Should we define $\sigma = \text{modulo}(\frac{L}{u_1})$, $\sigma \in [1, u_1)$, then:

1) d_1 will be used exactly $(u_1 - \sigma)$ times;
2) while d_1' will be used exactly σ times;

over a BS of fixed length L. It is obvious that

$$d_1' = d_1 + 1. \tag{12}$$

Then, if $\frac{l_i}{L} \to 0$, it will hold that

$$\begin{aligned} L &= \sum_{j=1}^{u_1} d_{1j} \\ &= d_1 \times (u_1 - \sigma) + d_1' \times \sigma \overset{\text{eq. (12)}}{\Rightarrow} d_1 = \frac{L - \sigma}{u_1} \end{aligned} \tag{13}$$

while the mean client waiting time for this page is

$$\begin{aligned} \overline{D(p_i)} &= \frac{d_1^2 \times (u_1 - \sigma) + d_1'^2 \times \sigma}{2L} \overset{\text{eq. (12,13)}}{\Rightarrow} \\ \overline{D(p_i)} &= \frac{L}{2u_1} - \frac{\sigma^2}{2Lu_1} + \frac{\sigma u_1}{2Lu_1}. \end{aligned} \tag{14}$$

On the the other hand, should zero padding be used, L should be increased by $(u_1 - \sigma)$. Then, $L' = L + u_1 - \sigma$, and the spacing becomes steady and equal to $d_1 = \frac{L'}{u_1}$. The mean client waiting time in this case is

$$\overline{D_Z(p_i)} = \frac{L'}{2u_1} = \frac{L}{2u_1} + \frac{1}{2} - \frac{\sigma}{2u_1}. \tag{15}$$

It is now easy to show that

$$\overline{D_Z(p_i)} - \overline{D(p_i)} = \frac{(L - \sigma)(u_1 - \sigma)}{2Lu_1} > 0 \tag{16}$$

since $\sigma \in [1, u_1)$. Therefore, $\overline{D_Z(p_i)} > \overline{D(p_i)}$ in any case.

Even though this analysis proves the inferiority of zero padding, it does not take into account the spacing conflicts described in Section II-D. Yet, the zero padding method introduced in [7] is known to overcome this problem as well. However, this method has severe weaknesses that can become evident through a simple example.

1) Brief Method Description: Begin by sorting the pages in descending request probability fashion. Perform grouping by page speed. Calculate the least common multiple (LCM) of the page groups' speeds. Split each group of pages into $\frac{\text{LCM}}{u_i}$ chunks, employing zero padding in the process where necessary. Finally, broadcast the chunks in a round-robin manner.

Consider the following example. Three pages have request probabilities $\pi_{1\dots3} = \{\frac{3}{4}, \frac{1}{6}, \frac{1}{12}\}$. For an $L = 20$-pages-long BS, their approximate optimal speeds are $u_{1\dots3} = \{11, 5, 4\}$, the least common multiple of which is 220. Thus, we have the following:

1) Group 1 (i.e., page p_1) must be split into 110 chunks (i.e., 109 dummy pages must be added to it).
2) Groups 2 and 3 are similarly expanded by 43 and 54 dummy pages correspondingly.

The new BS has length $L' = 209$ instead of the original $L = 20$ and corresponds to a mean client waiting time of $\overline{D'} = 11.55$ (2) instead of $\overline{D} = 1.22$ (4). Thus, the system's performance is reduced to a mere 10% of its ideal value.

In conclusion, zero padding should be avoided in data broadcasting due to serious performance penalties.

E. On the Necessity of Data Segmentation

It is interesting to examine the limitations of the analysis of [5], an overview of which has been given in Section II. [5] is

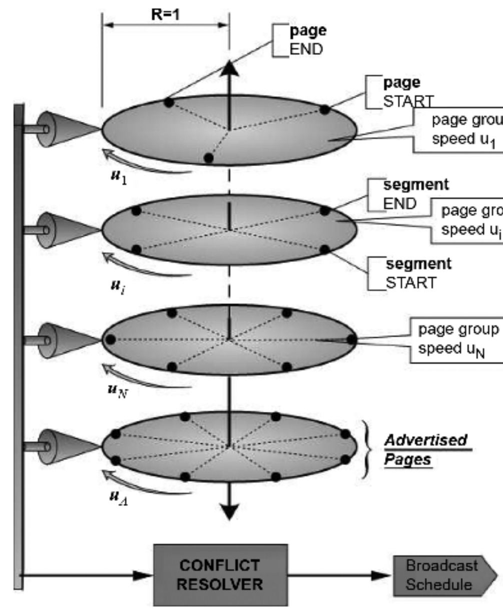

Fig. 5. CABS schedule constructor.

considered the analytically optimal state of the art in the field of broadcast scheduling.

In the event of uneven page-occurrence spacings, the mean client waiting time is given by

$$\bar{D} = \sum_{i=1}^{N} \pi_i \sum_{j=1}^{u_i} \frac{d_{ij}^2}{2L} \tag{17}$$

where d_{ij} is the spacing between the jth and $(j-1)$th occurrence of page p_i in the BS. Spacings are measured from one page-occurrence start (i.e., header) to the next. It has been proven [1] that (17) is minimized when the spacings of each page are constant, $d_{ij} = \frac{L}{u_i}\ \forall j$. Equation (17) is then reduced to the form of (2). However, $\frac{L}{u_i}$ spacings can be impossible in the event of uneven page sizes l_i.

Consider three pages $p_{1\ldots3}$: $[\{u_1 = 3,\ l_1 = 1\},\ \{u_2 = 2,\ l_2 = 1\},\ \{u_3 = 1,\ l_3 = 3000\}]$. The corresponding constant spacings would be $d_1 = \frac{3002}{3} \approx 1000$, $d_2 = 1501$, $d_3 = 3002$. However, p_3 fits nowhere between the occurrences of either p_1 or p_2. The analysis of [5] is then essentially invalidated, yielding $\overline{D(p_1)} = \frac{d_1}{2} = 500$, $\overline{D(p_2)} = 750.5$, $\overline{D(p_3)} = 1501$ instead of the real values $\overline{D(p_1)^r} = 1501$, $\overline{D(p_2)^r} = 1501$, $\overline{D(p_3)^r} = 1501$ produced by (17).

The scheduling algorithm of [5] attempts to tackle this deficiency by increasing L infinitely, effectively incorporating thousands of redundant copies of pages p_1, p_2 in the BS. Moreover, simulations run in [5] consider page sizes $l_i \in [1, 10]$ while $L \to \infty$. This affects the ratio $\frac{l_i}{L} \to 0$, essentially reducing the scenario to that of equally sized pages. Indeed, the results of the scheduling algorithm in simulations of [5] are almost identical for both unevenly and evenly sized pages.

A better approach would be to segment pages to smaller packets. Large pages then become "filling material" between occurrences of small pages. Thus, L is unaffected, and spacings of $\frac{L}{u_i}$ become possible. Notice that the client waiting time is not affected either: The page headers are placed at their optimal positions. The technique is essentially an application of time multiplexing. The channel bandwidth is divided among the clients with better temporal granularity. While the present paper does not aim at introducing novel multiplexing schemes, a safe packet size limit can be set as

$$\Pi_{\min} = \min\{l_i\} \tag{18}$$

while a less segmentation-greedy packet size is obviously

$$\Pi = \frac{1}{N} \min\{d_i\}. \tag{19}$$

Packet sizes do not have an impact on waiting time, provided that the aforementioned problem is avoided, but do affect client data receival rate, smoothness, and noise resilience. Optimizing packet size for these purposes goes beyond the scope of the present work.

F. *Priorities Over Conflict*

Data segmentation and slight spacing deviations have been proven to be beneficial to the performance of a push system. However, the problem posed by conflicting spacings discussed in Section II-D has not yet been resolved.

As it has already been stated, conflicting spacings are unavoidable when zero padding is not used. Thus, measures must be taken to prioritize pages in the event of conflict. Equation (2) states that pages with higher $\frac{\pi_i}{u_i}$ value have heavier impact on the mean waiting time. Thus, the impact factor of a page p_i can be defined as

$$I_i = \frac{\pi_i}{u_i}. \tag{20}$$

Therefore, it is logical that in the event of conflicting page or packet spacings, items with higher impact factor should take priority and be handled accordingly.

G. *Theoretical Summary*

Through the analysis and remarks of the preceding sections, the following has become evident.

1) An optimal scheduler should provide optimal client waiting times with minimal BS length.
2) Each page-spacing value in the BS must deviate as little as possible from its ideal values in a uniform, distributed way. If not possible due to excessive $\frac{l_i}{L}$ ratio, big pages should be split into smaller packets. Zero padding should be avoided.
3) Finally, on the event of spacing conflicts, pages or packets should be prioritized according to their impact factor defined by (20).

IV. Cost-Effective, Adaptivity-Oriented Broadcast Scheduler

Having defined its optimal characteristics, the corresponding scheduler can be easily formulated.

The proposed Cost-effective, Adaptivity-oriented Broadcast Scheduler (CABS) employs a rotating disks scheme, depicted in Fig. 5.

We consider pages $p_i : \{\pi_i,\ l_i\},\ i = 1 \ldots N$. The optimal page speeds u_i^{opt} and L are set by Algorithm 1. The pages are clustered by their $\frac{l_i}{L}$ ratio through hierarchical clustering [31] with ending criterion

$$\left|\frac{l_i}{L} - \frac{l_\text{c}}{L}\right| \leq \frac{\Pi}{L} \qquad \forall p_i \in C \tag{21}$$

where C denotes the cluster owning page p_i, l_c indicates the page length of the corresponding centroid, and Π is the packet size of (19). Each formed cluster is then divided into group of pages with equal speeds. Each group has its contents segmented and is placed on the periphery of a circle with unary radius. The disks are set to rotate around a common axis with their corresponding speeds, and a set of stationary heads detects the start/end of pages or packets as shown in Fig. 5. In the event of the detection of a page/packet start/end, the page/packet is immediately broadcast. Should two or more pages conflict, they are sorted by descending impact factor I_i and are broadcast in that order. To minimize the probability of a conflict occurring, the starting angle of each disk is randomized in the range $[0, 2\pi]$. The procedure lasts for a time interval of $T = \frac{2\pi R}{\min\{u_i\}} \min\{u_i\} = 2\pi R = 2\pi$, at which point a BS of exactly $L = \sum_{i=1}^{N} u_i \cdot l_i$ has been produced.

The procedure is formulated as Algorithm 2.

Algorithm 2 The CABS Schedule Constructor

```
Input: The pages $p_i : \{\pi_i, l_i\},\ i = 1 \dots N$.
Output: The broadcast schedule.
 1:  Set $u_i^{\text{opt}}$ by Algorithm 1.
 2:  Perform Hierarchical Clustering of pages by $\frac{l_i}{L}$ and
     criterion of (21).
 3:  Group pages in the clusters by $u_i$.
 4:  Set $NoD$ as the number of produced groups.
 5:  Segment pages by (19).
 6:  Set initial position of heads:
     $S_0^i = \text{random}\{[0, 2\pi]\},\ i = 1 \dots NoD$
 7:  Set $\Delta t = \min\{\frac{2\pi}{\text{DiskSize}_i \cdot u_i},\ i = 1 \dots NoD\}$.
 8:  Set $T = 2\pi$.
     Create the broadcast schedule
 9:  for $t := \Delta t$ to $T$, step by $\Delta t$ do
10:      Calculate current head positions:
         $S_t^i = S_0^i + u_i \cdot t,\ i = 1 \dots NoD$
11:      Collect pages/segments whose boundaries have
         been crossed in $[t, t - \Delta t]$, in array $B$.
12:      Sort $B$ by descending impact factors $I_i$.
13:      Broadcast items of $B$.
14:  end for
```

As an extra feature, CABS can also easily set custom speeds for any custom group (i.e., disk), e.g., when advertising certain pages with a guaranteed client waiting time variance [32].

V. Performance and Cost Assessment

Even though the proposed algorithm adheres to the guidelines of the theoretical analysis, it is imperative that its performance be assessed and compared to other approaches.

To this end, the CABS algorithm is compared to the analytically optimal scheduling algorithm introduced in [5] (SOA) and the latest advancement (BDISK-SOA) of the Broadcast Disks family of algorithms [7] (BDISK-CLASSIC). The former, though introduced in 1999, still represents the top-performing algorithm client waiting-time-wise. BDISK-CLASSIC represents a very popular scheduling approach, as discussed in Section I-A. BDISK-SOA was presented by the authors in [25] and [26], where it surpassed the previous state of the art. Performance of the ideal scheduler of Section II-C (IDEAL) is given where applicable.

TABLE I
Feature Support of Scheduling Algorithms

Feature	SOA*1	BDISKS, BDISKS-SOA*2	CABS
Periodic BS	no	yes	yes
Varying page size	yes	no	yes
Data segmentation	no	no	yes
Page advertising	no	no	yes

*1: *The analytically optimal algorithm presented in [5]*
*2: *The algorithms of [7, 25]* respectively

The performance assessment is twofold.

1) The classic criteria, i.e., the mean client waiting times achieved by all three algorithms are compared for a variety of different client probabilistic configurations.
2) The length of the produced broadcast schedules and the CPU times required to construct them are compared as a metric of the system's implementation and operational cost, as described in Section III-A.

Notice that to the best of our knowledge, the second criterion is overlooked in the totality of the research on push systems. As the results prove, though, it is exactly these criteria that determine whether an algorithm can be implemented in the real world: Push systems are *de facto* used as a minimal-cost solution. Should an algorithm require exquisite, powerful, and expensive hardware to run or excessive operational resources, it would be wise to abandon the push logic altogether and resort to classic client–server architectures that are bound to perform better.

A. Simulation Setup and Results

As shown in Table I, not all compared algorithms support all the features discussed in the theoretical analysis. Thus, two comparisons take place: All algorithms are initially compared in an evenly sized pages scenario. Then, only SOA and CABS are compared in the varying-page-sizes case.

The comparison is simulation-based, and the corresponding parameters are presented in Table II. The parameter values used are typically used in similar papers. Worthy of special note are the total number of pages N and the client probabilistic model.

A typical 3.5 G telephony cell can typically handle 10 000 P2P client connections. For the broadcasting scenario to make sense, a logical and feasible number of clients would thus be 30 000. Assuming that the clients share a common interest for a good 60% of the available pages—broadcasting would have dubious meaning otherwise—and their access patterns are comprised of 10 pages each, the total number of items is $N = 30\,000 \cdot 10 \cdot (1 - 0.6) = 120\,000$.

The only client pdf used in all related work is the Zipf model. This global choice has a sound basis: All pages are typically sorted by descending popularity π_i. Should this be applied to

TABLE II
SIMULATION PARAMETERS

Parameter	Value
TOPOLOGY	STAR $\{1$ node-server: n wireless clients$\}$
Client Query p.d.f θ	ZIPF: $\pi_i = \frac{\sum_{i=1}^{N} \frac{1}{i^\theta}}{i^\theta}, i = 1 \ldots N$ $\{[0.2 : 0.1 : 1.8]\} \cap \overline{\{1\}}$
SYSTEM STATE	BS Renewal triggered (see Section II-B)
PAGE SPEEDS DEFINITION	SOA: $round(u_i^{ideal})$ CABS: Algorithm 1 BDISKS-SOA: Procedure of [26] BDISKS: $round(u_i^{ideal})$
Number of pages **Simulation Duration** **ThinkTime***	$N = 120,000$ $600,000$ client queries 2 page segments

* *Introduced in [7]: upon receiving a wanted page, the client halts queries for this time interval.*

any of the other popular distributions, e.g., Pareto, Gaussian, they can precisely be substituted by a Zipf pdf with a proper θ value. To this end, the authors maintain the Zipf convention and examine a wide set of θ values. As a side note, analysis of this paper is pdf-independent.

Concerning the calculation of the CPU time required by each algorithm, two approaches are most popular:

1) calculate and compare the complexity $O(.)$ of each algorithm;
2) implement each algorithm in assembly and count CPU cycles required by each one.

In the present case, the first option is not a choice since the complexity for SOA is not provided [5], and the complexity of CABS cannot be calculated since the algorithm employs grouping of pages by their speed u_i, which is function of their access probabilities (i.e., random). The assembly implementation choice also has severe weaknesses.

1) Any assembly implementation is too hardware-specific to be verifiable.
2) A few decades of higher language code typically correspond to several thousands lines of assembly code. Thus, implementation optimization is a problem that is bound to be more influential in the case of assembly.
3) Handwritten assembly code is obviously much less reliable than the corresponding one produced by a well-known and globally trusted compiler or interpreter.

Thus, all algorithms were implemented in MATLAB as standalone functions. The CPU time spent in each one was then measured [33], [34] by the MATLAB Code Profiler tool. The reasons for this choice were the following.

1) Each algorithm has typically an implementation of 10 or less lines of MATLAB code, leaving little, if any, room for implementation issues to be considered.
2) The MATLAB interpreter and profiler are popular and well-known in the scientific community globally.
3) Only one official MATLAB interpreter exists, as opposed to other programming languages like $\mathrm{C/C++}$ or Fortran. The results are thus more verifiable and trustworthy.

TABLE III
CPU TIME ASSESSMENT SPECIFICATIONS

Parameter	Value
Test CPU model **Number of cores** **Test RAM**	x86 Family 6 model 15 Stepping 11 GenuineIntel 4× 2672MHz 4GB DDR2 800 MHz
Operating System **Version**	Windows XP Professional 5.1.2600 SP3 Build 2600
Programming language **Environment**	Single-threaded MATLAB M-code MATLAB R2009b
CPU Usage during execution **Application Priority**	Steady 100% on one dedicated core* 13 ("High")

*: *Measured by means of Windows Task Manager*

4) The MATLAB version used supports JIT compiling, providing execution times comparable to corresponding $\mathrm{C/C++}$ implementation.

Throughout the execution time, great care was taken to keep the CPU usage steady at 100% and dedicated to the MATLAB interpreter. Specification details are given in Table III.

Results are displayed in Figs. 6–8, depicting achieved $\overline{D}$, L, and CPU times of the compared algorithms.

For the unevenly-sized-pages scenario, we assume $N = 50$ pages with sizes uniformly distributed in $[0.5, 1.5]$. One large page with index $I \in [1, 50]$ is selected, and its size is set to 50 000. The ratio $\frac{1}{50\,000}$ refers to the size of a typical 2-kB HTML page divided by the size of a streaming video file of 100 MB. Zipf distribution of page popularity is assumed once more, with the θ-parameter ranging again in $[0.2, 1.8]$. Results for this scenario are displayed in Fig. 9, depicting achieved $\bar{D}$ of the algorithms supporting unevenly sized pages (SOA and CABS).

B. Remarks

The waiting times achieved by the compared algorithms (Figs. 6 and 9) indicate that CABS is the only algorithm achieving ideality in both evenly- and unevenly-sized-pages scenarios. Notice that apart from CABS, only SOA supports unevenly sized pages, and in accordance with the remarks of Section III-E, fails to achieve optimal results even by a difference of $2.7 \cdot 10^7$ length units. SOA tends to perform better as the large page becomes less popular either by increasing the large page index or the θ-parameter of the Zipf pdf. Its performance, however, is not satisfactory in the majority of the cases.

In the evenly-sized-pages scenario, CABS and SOA achieve ideality (Fig. 6). However, CABS requires $\approx$ 20 s on the test machine to calculate the required BS in one pass, while SOA requires an aggregate amount of days (Fig. 8). This effectively means that to answer merely 20 queries for each of the 30 000 clients (thus the total of 600 000 queries of the simulation), SOA would require several days of optimization, thus creating a major bottleneck. The only possible solution would be to run SOA on specialized, powerful, and expensive hardware. Notice that SOA cannot be divided to threads.

BS sizes achieved by SOA and CABS (Fig. 7) denote a difference of a factor of 10^5. CABS achieves a value of one

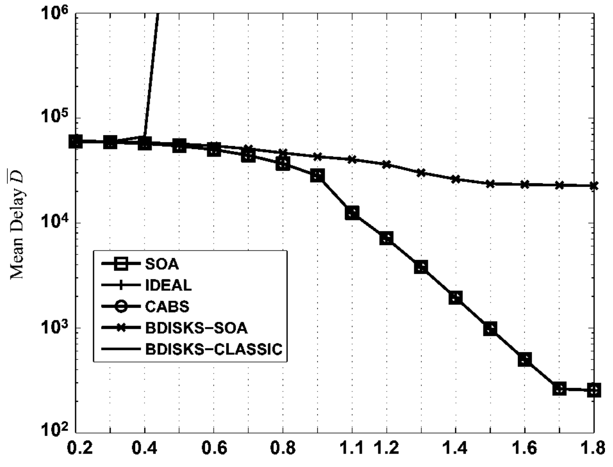

Fig. 6. Mean client waiting times achieved by the competing algorithms in the evenly-sized-pages scenario.

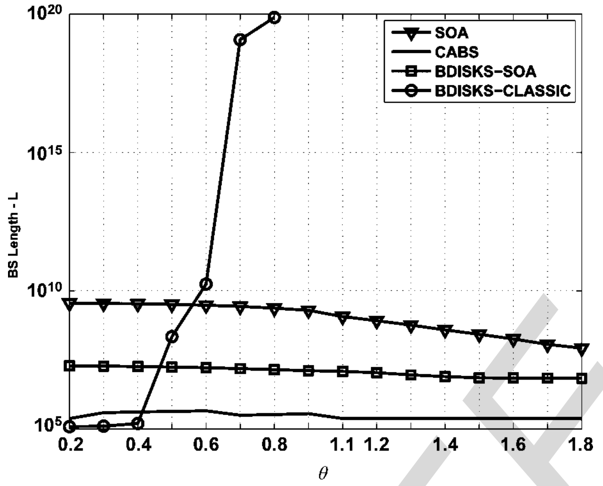

Fig. 7. Illustration of the schedule sizes required by each algorithm.

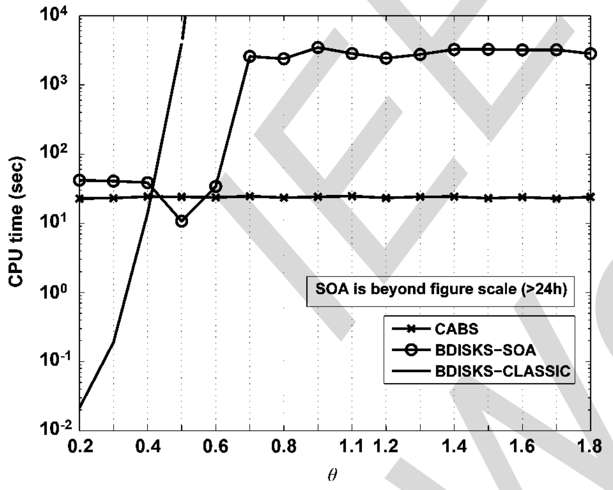

Fig. 8. CPU time required by each algorithm on the test machine.

to three times the database size. SOA, requiring 10^5 times greater BS, nullifies all cost-effective advantages mentioned in Section III-A.

BDISKS-CLASSIC and BDISKS-SOA only refer to the evenly-sized-pages scenario. They both offer suboptimal performance, with BDISKS-SOA being the only one performing acceptably (Fig. 6). BDISKS-CLASSIC performs well only for $\theta \in [0.2, 0.4]$, where the optimal schedule is nearly flat with no need for exquisite optimization. For greater θ values, both the CPU times and BS sizes of BDISKS-CLASSIC render it nonrealizable. BDISKS-SOA greatly outperforms SOA in terms of resources (Figs. 7 and 8), but not CABS. Finally, the least common multiple employed by the BDISKS-x algorithms may sporadically yield conveniently small values. In this case, the required CPU time is minimized as shown in Fig. 8. However, this only happens for small θ values where the algorithms achieve the performance of a nearly flat broadcast schedule (Fig. 6).

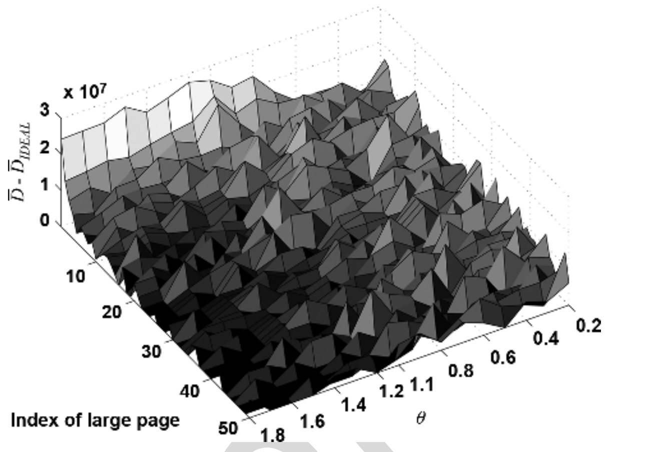

Fig. 9. Deviation of mean client waiting times achieved by SOA from the IDEAL scheduler unevenly-sized-pages scenario. CABS achieves ≈ 0 deviation in all cases.

VI. Conclusion

This paper focused on the real-world implementation perspectives of algorithms designed to construct the schedule of broadcast-based (or push-based) systems. Push systems are primarily designed as a cheap telecommunications solution, and their applications typically include hospital and airport informative systems, instant messaging services, interactive television, and more. However, the main research on the field has focused on performance optimization, disregarding the most vital characteristic a push system should possess: low implementation cost.

By theoretically analyzing many aspects of the broadcast scheduling procedure, a new broadcast scheduler constructor algorithm, the Cost-effective Adaptivity-oriented Broadcast Scheduler (CABS) has been proposed. CABS was compared in terms of broadcast schedule size and CPU cost with the current state-of-the-art SOA algorithm, and algorithms of the Broadcast Disks model family. The results have proven that these algorithms have neglected the cost attributes to the point that may be only conditionally realizable.

As a general paper conclusion, it is shown that any push-related study should take the implementation cost into account, apart from waiting-time-based performance. Successful compliance with the aforementioned criteria results in realistic and practically useful systems.

References

[1] C. Kenyon and N. Schabanel, "The data broadcast problem with non-uniform transmission times," in *Proc. 10th Annu. ACM-SIAM SODA*, Baltimore, MD, Jan. 1999, pp. 547–556.

[2] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieberaruch, "Minimizing service and operation costs of periodic scheduling," in *Proc. 9th Annu. ACM-SIAM SODA*, San Francisco, CA, Jan. 1998, pp. 11–20.

[3] M. Ammar and J. Wong, "On the optimality of cyclic transmission in teletext systems," *IEEE Trans. Commun.*, vol. 35, no. 1, pp. 68–73, 1987.

[4] S. Hameed and N. Vaidya, "Efficient algorithms for scheduling data broadcast," *Wireless Netw.*, vol. 5, no. 3, pp. 183–193, 1999.

[5] N. Vaidya and S. Hameed, "Scheduling data broadcast in asymmetric communication environments," *Wireless Netw.*, vol. 5, no. 3, pp. 171–182, 1999.

[6] N. Schabanel, "The data broadcast problem with preemption," in *Proc. 17th STACS*, Feb. 2000, pp. 181–192.

[7] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data management for asymmetric communication environments," in *Proc. ACM SIGMOD*, New York, May 1995, pp. 199–210.

[8] K. Wu, P. Yu, and M. Chen, "Energy-efficient caching for bandwidth-limited wireless mobile computing," in *Proc. 12th IEEE ICDE*, Feb. 1996, pp. 335–343.

[9] M. Chen, P. Yu, and K. Wu, "Optimizing index allocation for sequential data broadcasting in wireless mobile computing," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 1, pp. 161–173, Jan.–Feb. 2003.

[10] J. Xu, W. C. Lee, X. Tang, Q. Gao, and S. Li, "An error-resilient and tunable distributed indexing scheme for wireless data broadcast," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 3, pp. 392–404, Mar. 2006.

[11] C. Hu and M. Chen, "Adaptive multichannel data dissemination: Support of dynamic traffic awareness and push-pull time balance," *IEEE Trans. Veh. Technol.*, vol. 54, no. 2, pp. 192–203, Mar. 2005.

[12] Y. Chang, C. Yang, and J. Shen, "A binary-number-based approach to data broadcasting in wireless information systems," in *Proc. IEEE Int. Conf. Wireless Netw.*, 2005, vol. 1, pp. 348–353.

[13] W. Wang and C. Ravishankar, "Adaptive data broadcasting in asymmetric communication environments," in *Proc. 8th IDEAS*, 2004, pp. 27–36.

[14] H. Leung, J. Yuen, K. Lam, and E. Chan, "Enhanced multiversion data broadcast scheme for time-constrained mobile computing systems," in *Proc. 13th DEXA*, 2002, pp. 747–751.

[15] T. F. Bowen, G. Gopal, G. Herman, T. Hickey, K. C. Lee, W. H. Mansfield, J. Raitz, and A. Weinrib, "The Datacycle architecture," *Commun. ACM*, vol. 35, no. 12, pp. 71–81, 1992.

[16] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham, "Efficient concurrency control for broadcast environment," in *Proc. ACM SIGMOD*, 1999, pp. 85–96.

[17] E. Pitoura and R. Chrusanthis, "Multiversion data broadcast," *IEEE Trans. Comput.*, vol. 51, no. 10, pp. 1224–1230, Oct. 2002.

[18] G. Lee, S. Lo, and A. Chen, "Data allocation on wireless broadcast channels for efficient query processing," *IEEE Trans. Comput.*, vol. 51, no. 10, pp. 1237–1252, Oct. 2002.

[19] J. Huang and M. Chen, "Broadcasting dependent data for ordered queries without replication in a multi-channel mobile environment," in *Proc. 19th IEEE ICDE*, Mar. 2003, pp. 692–694.

[20] G. Lee, M. Yeh, S. Lo, and A. Chen, "A strategy for efficient access of multiple data items in mobile environments," in *Proc. 3rd Int. Conf. Mobile Data Manage.*, 2002, pp. 71–78.

[21] J. Huang and M. Chen, "Broadcast program generation for unordered queries with data replication," in *Proc. 18th ACM SAC*, Mar. 2003, pp. 866–870.

[22] F. Martinwz, U. Gonzalez, and I. Stojmenovic, "A parallel hill climbing algorithm for pushing dependent data in clients-providers-servers systems," in *Proc. 7th IEEE ISCC*, 2002, pp. 611–616.

[23] A. Bar-Noy, J. Naor, and B. Schiber, "Pushing dependent data in clients-providers-servers systems," in *Proc. ACM MobiCom*, 2000, pp. 222–230.

[24] C. Liaskos, S. Petridou, G. Papadimitriou, P. Nicopolitidis, M. Obaidat, and A. Pomportsis, "Clustering-driven wireless data broadcasting," *IEEE Wireless Commun. Mag.*, vol. 16, no. 6, pp. 80–87, Dec. 2009.

[25] C. Liaskos, S. Petridou, P. Nicopolitidis, and G. Papadimitriou, "On the analytical performance optimization of wireless data broadcasting," *IEEE Trans. Veh. Technol.*, vol. 59, no. 2, pp. 884–895, Feb. 2010.

[26] C. Liaskos, S. Petridou, and G. Papadimitriou, "Cost-aware wireless data broadcasting," *IEEE Trans. Broadcast.*, vol. 56, no. 1, pp. 66–76, Mar. 2010.

[27] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Using learning automata for adaptive push-based data broadcasting in asymmetric wireless environments," *IEEE Trans. Veh. Technol.*, vol. 51, no. 6, pp. 1652–1660, Nov. 2002.

[28] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Exploiting locality of demand to improve the performance of wireless data broadcasting," *IEEE Trans. Veh. Technol.*, vol. 55, no. 4, pp. 1347–1361, Jul. 2006.

[29] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Multiple-antenna data broadcasting for environments with locality of demand," *IEEE Trans. Veh. Technol.*, vol. 56, no. 5, pt. 1, pp. 2807–2816, Sep. 2007.

[30] P. Fishburn, "Intransitive indifference in preference theory: A survey," *Oper. Res.*, vol. 18, no. 2, pp. 207–228.

[31] A. Jain and R. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ: Prentice-Hall, 1988.

[32] S. J. N. Vaidya, "Scheduling algorithms for a data broadcast system: Minimizing variance of the response time," Comput. Sci. Dept., Texas A&M Univ., College Station, TX, TR 98-005, 1998.

[33] E. Alexandre, A. Pena, and M. Sobreira, "Low-complexity bit-allocation algorithm for MPEG AAC audio coders," *IEEE Signal Process. Lett.*, vol. 12, no. 12, pp. 824–826, Dec. 2005.

[34] R. Sukkar, S. Herman, A. Setlur, and C. Mitchell, "Reducing computational complexity and response latency through the detection of contentless frames," in *Proc. IEEE ICASSP*, Antwerp, Belgium, Jun. 2000, pp. 3751–3754.

Christos K. Liaskos (M'08) received the Diploma degree in electrical and computer engineering in 2004 and the M.S. degree in medical informatics in 2008 from the Aristotle University of Thessaloniki, Thessaloniki, Greece, where he is currently pursuing the Ph.D. degree in communication networks.

His research interests include wireless networks, neural networks, and fractal compression algorithms.

Sophia G. Petridou (M'08) received the Diploma and Ph.D. degrees in computer science from the Aristotle University of Thessaloniki, Thessaloniki, Greece, in 2000 and 2008, respectively.

[Current position at university?]Her research interests include clustering and optical and wireless networks.

Georgios I. Papadimitriou (M'89–SM'02) received the Diploma and Ph.D. degrees in computer engineering from the University of Patras, Patras, Greece, in 1989 and 1994, respectively.

In 1997, he joined the faculty of the Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece, where he is currently an Associate Professor. He is coauthor of three books published by Wiley. He is author or coauthor of 90 journal and 100 conference papers. His main research interests include optical networks and wireless networks.

Prof. Papadimitriou is Associate Editor of five IEEE journals.

Towards Realizable, Low-Cost Broadcast Systems for Dynamic Environments

Christos K. Liaskos, *Member, IEEE*, Sophia G. Petridou, *Member, IEEE*, and Georgios I. Papadimitriou, *Senior Member, IEEE*

***Abstract*—A main design issue in a wireless data broadcasting system is to choose between push-based and pull-based logic: The former is used as a low-cost solution, while the latter is preferred when performance is of utmost importance. Therefore, the most significant advantage of a push system is the minimal cost. This fact implies that hardware limitations do exist in the case of push systems. As a consequence, every related proposed algorithm should primarily be cost-effective. This attribute, however, has been overlooked in related research. In this paper, popular broadcast scheduling approaches are tested from an implementation cost aspect, and the results render them only conditionally realizable. Moreover, a new, cost-effective, adaptivity oriented schedule constructor is proposed as a realistic, minimal-cost solution.**

***Index Terms*—Adaptivity, analysis, CPU-memory cost, data serialization, push systems.**

I. Introduction

RECENT years have witnessed the wide-spreading use of wireless push-based systems. Simple in architecture and implementation, lightweight and energy efficient—especially from the client's point of view and both hardware- and software-wise—the push-based approach has been adopted for use in a variety of information dissemination applications and has been incorporated in almost every single mobile telecommunications device. Popular uses include airport and hospital information systems, instant messaging services, and multimedia on demand over the Internet or cellular networks. Consequently, the on-growing interest of the telecommunications industry has spurred the research on the performance optimization of these systems.

Pure push-based systems typically employ a central server continuously transmitting data through a channel, while the clients simply retrieve useful data from the stream, without being able to perform any kind of queries. Pull-based systems on the other hand adhere to the classical client–server scheme, where the server transmits only the data that have been requested by a client, and only on the event of a request. While the functionality separation is clear, a common misconception lies to the reasons that render a pull or a push choice more suitable for a telecommunications system case. It must be clarified that a typical pull-based scheme generally performs better, response time-wise. However, the push-oriented systems have one unique advantage: very low implementation cost with regard to scalability.

The reader is encouraged to visualize a push server as a device of the scale of a standard, mainstream computer at most. It is therefore obvious that such a device has limited computational power and caching capabilities. Thus, the computational power and required memory should be attributes of the highest impact factor when designing broadcasting-related algorithms. Network congestion minimization—high due to the continuous data emission scheme of push systems—should also be a major concern. However, to the best of our knowledge, this has not been the case so far in related research.

A. Related Work

Research on the field of broadcast systems can be roughly split into two main categories: the mathematical foundation and analysis attempts of the broadcast problem in general, and algorithmic approaches that tend to provide simplified, algorithmically applicable solutions of the general problem and/or examine other advanced aspects. In the definition of the data broadcast problem, each to-be-broadcasted item is assigned a generic "broadcast weight," which abstractedly represents the impact of broadcasting on system resources. The classic problem is then to find a schedule over an infinite-time horizon so that the mean client waiting time as well as the mean broadcast weight are minimized. In this context, the problem is proved to be NP-hard [1]. [2] discusses the generalized maintenance problem (a known NP-hard problem) and proves that the broadcasting of equally sized items is a subcase of it. A lower bound for the client's mean waiting time is also provided. [3] proves the existence of a possible solution for teletext systems and also defines a lower bound for the client's mean waiting time in this case.

From this point on, related research ignores broadcast weights. In this case, uniform and nonuniform item sizes are discussed. [4] and [5] present the lower bound for the client mean waiting time in the case of both uniform and nonuniform sizes. A scheduling algorithm is also defined, which even today achieves the minimum client waiting times. The lower bound in the nonuniform case is not tight though, and this case is further analyzed in [1]. [6] introduces a "reservation" system that favors client waiting time in the case of big-sized items.

Retaining the no-broadcast weights consideration, several algorithmic approaches have been proposed to simplify the data broadcasting problem, the most influential of all being the Broadcast Disks model [7], which is also discussed in this paper. A great deal of work has been done based on this model,

Manuscript received July 30, 2009; revised April 08, 2010; accepted July 21, 2010; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. Rodriguez.

The authors are with the Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece (e-mail: cliaskos@csd.auth.gr; spetrido@csd.auth.gr; gp@csd.auth.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2010.2062534

studying data prefetching, caching [8] and indexing [9], [10], hybrid data broadcasting [11], as well as scheduling strategies and noise interference [12], [13].

Another branch studies data broadcasting from a higher level, transaction aspect (e.g., queries that must be answered in a limited amount of time [14], among other limitations) and was introduced in [15]. This approach has since been extended with fail-over tactics [16] and techniques [17] that increase the concurrency of transactions.

Finally, other recent studies deal with the case of correlated client queries, e.g., complex queries requesting multiple single data items [18]. [19] deals with the case of requesting successive single items, while [20] and [21] examine the case of random access order of single items. [22] and [23] present algorithms that achieve the lower bounds of average access time in the case of broadcasting a pair of files.

B. Authors' Related Work

The authors have so far contributed in both analytical and algorithmic approaches. In [24], an effort to combine clustering techniques with the broadcast disks model was made, with satisfactory results. The analytical approach of [25] added means of projecting the optimal parameters required to construct the broadcast schedule. The cost parameter was introduced in [26], targeting the cost efficiency of the scheduling procedure. Moreover, research on adaptive push systems has been carried out in [24] and [27]–[29].

C. Contribution

As it is evident from the brief presentation of the related work on the field, every study focuses on a specific subject of data broadcasting without however taking the implementation cost into consideration. Thus, a good amount of algorithms have been proposed, which may be hard to be implemented in the real world. Moreover, the classic system model of [7] employed by all related studies is simulation oriented and does not claim any effectiveness in cost assessment tasks.

In the present paper, a more realistic system model is presented, designed to augment the network's functionality through the addition of Web connectivity, distinction between broadcast schedulers and beacons, and broadcast schedule lifespan parameters. The new model is then used to perform cost assessment studies of broadcast scheduling techniques, aiming at the following:

1) the demonstration of the fact that implementation cost assessments are critical for algorithms related to push systems;
2) the testing of adaptivity capabilities of popular broadcast scheduling approaches;
3) the presentation and testing of a new cost-effective, adaptivity-oriented broadcast schedule constructor algorithm.

The proposed scheduling scheme is compared to the analytically optimal [5] and modern, broadcast disks-based scheduling algorithms. Results indicate that the novel scheme combines optimal performance and minimal cost, while the compared algorithms are even conditionally realizable in several cases.

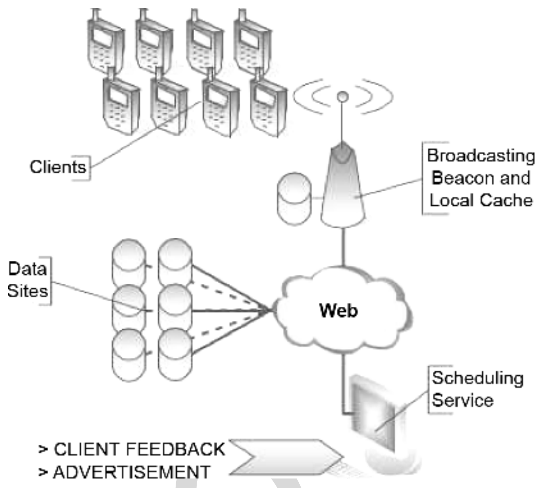

Fig. 1. Novel broadcasting model.

D. Summary of Contents

Section II provides the basics for understanding the push system's model, terms, architecture, and operation. In Section III, the broadcast scheduling procedure is theoretically analyzed. In Section IV, the proposed scheduling algorithm is presented. Performance assessment configurations and results are given in Section V. Our conclusion is given in Section VI.

II. Broadcasting Basics

A. Network Topology and Operation: Advancing the Classic Model

The proposed topology for a realistic push-based system is depicted in Fig. 1. A set of clients receives a common broadcast schedule emitted from a central source.

Concerning the clients, the following is commonly assumed.

- They share common information needs and preferences with regard to their topological distribution [28].
- Their number is large enough to rule out a point-to-point-based style of communication from being a viable option with regard to implementation cost.

The broadcast source emits series of data items—better known as "pages"—according to a schedule that minimizes the clients' mean waiting time. In the classic system model, the server is standalone, i.e., unplugged from any network and self-sustaining regarding the origin of the broadcast data [1]–[23]. However, this scheme was introduced mostly for conceptual purposes [7] and is of limited practical use. It is safer to assume that the data originates from independent and distributed Web sources, as shown in Fig. 1.

The classic model also considers the scheduling service to be located at the beacon. While a beacon-local cache may be useful, placing the scheduler at the beacon would require costly changes at the totality of the base stations of a modern cellular network, which tend to act as simple terminals. It is thus more practical to consider the broadcast beacon and scheduling service being at separate locations.

In any case, however, the scheduling service is assumed to know [1]–[23]:

- which pages the clients need;
- the popularity of each data item.

This knowledge can be attributed to timely speculations, forecasting, client feedback, or data advertising.

Once constructed, the broadcast schedule has limited lifespan. After a certain time interval, it must be renewed to match the clients' demands. At this point, the scheduling service must:

- schedule data fetching over the Web;
- schedule storage at the beacon-local cache;
- enter a state of client preference monitoring in order to prepare for the next renewal.

B. Entry Point of the Present Work

The present work assumes that the system has just entered a renewal state and aims to produce the optimal schedule:

- as fast as possible on mainstream hardware;
- minimizing the caching needs at the beacon-local point;
- minimizing the required communication lifetime between the scheduling service and the data sites, thus limiting the required operational bandwidth and the dependence on network congestion.

As it will be shown, these goals can be effectively reached through the minimization of the broadcast schedule size.

C. Theoretical Overview: Ideal Scheduling

Input of the Ideal Schedule Constructor (ISC) are the pairs $p_i : \{l_i, u_i\},\ i = 1 \ldots N$, where p_i denotes the ith page of the server's database (arbitrarily enumerated), l_i the page's size, and u_i the page's speed (i.e., the the number of occurrences of p_i inside the broadcast schedule, BS). N denotes the total number of pages eligible for broadcasting. The length of the broadcast schedule is denoted by L, and it stands that

$$L = \sum_{i=1}^{N} u_i \cdot l_i. \tag{1}$$

The objective of the ISC is simple: arrange the occurrences of each page so as to achieve the minimum mean client waiting time. It has been shown that this criterion is satisfied only when the time interval between same page occurrences is steady, i.e., periodic schedule [7].

For arbitrary page speeds u_i and access probabilities π_i, the mean client waiting time is

$$\bar{D} = \frac{L}{2} \sum_{i=1}^{N} \frac{\pi_i}{u_i}. \tag{2}$$

The optimal page speeds are strictly defined through analysis [5]

$$u_i^{\text{ideal}} = L \cdot \sqrt{\frac{\pi_i}{l_i}} \frac{1}{\sum\limits_{i=1}^{N} \sqrt{\pi_i \cdot l_i}}, \quad i = 1 \ldots N \tag{3}$$

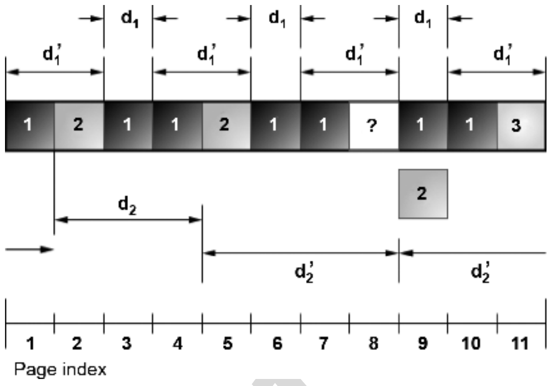

Fig. 2. Example of conflicting page spacings d_i.

which result into a minimal waiting time of

$$\overline{D_{\min}^{ideal}} = \frac{\left(\sum\limits_{i=1}^{N} \sqrt{\pi_i \cdot l_i}\right)^2}{2} \tag{4}$$

where (2)–(4) are only valid when $\frac{l_i}{L} \approx \frac{l_j}{L}\ \forall i,\ j$ [5].

D. Forced Deviations From Ideality

Two factors hinder strict appliance of the ideal scheduling scheme.

- A forced deviation stems from (4), which promises independence from the BS length L. However, as shown in [5], (3) yields noninteger results, and thus a form of rounding must be applied. The page speeds are thus altered, and the real mean waiting time is given by (2), which shows dependence from L.
- Creating a periodic broadcast schedule with constant $d_i = \frac{L}{u_i}$ spacings [5] between consecutive occurrences of a page may be impossible due to collisions. For example, consider a BS comprising three pages, $p_{1\ldots3}$, with corresponding speeds $u_{1\ldots3} = \{7,3,1\}$ and $l_1 = l_2 = l_3 = 1$. The ideal spacings should then be $d_1^{ideal} = \frac{L}{u_1} = \frac{11}{7}$, $d_2^{ideal} = \frac{L}{u_2} = \frac{11}{3}$, and $d_3^{ideal} = \frac{L}{u_3} = 11$, given that $L = \sum_{i=1}^{3} u_i \cdot l_i = 11$. The problem of integer divisions is evident, but suppose that in order to overcome it, we choose varying spacing values. For example, $u_1 = \{d_1 \times 3 + d_1' \times 4\}$, where $d_1 = 1$ and $d_1' = 2$; $u_2 = \{d_2 \times 1 + d_2' \times 2\}$, where $d_2 = 3$ and $d_2' = 4$; and $u_3 = \{11 \times 1\}$, where $d_3^* = 11$. As shown in Fig. 2, the spacings collide at the ninth position.

III. Impact of Deviations From Ideality

In this section, the form of deviation that has the smallest impact on the performance of a scheduling algorithm is studied.

A. Altering the BS Length: Cost-Efficiency and Performance Issues

As described in Section II-D, the real mean waiting time of the clients is dependent on the BS length L.

Let $L_I = \sum_{i=1}^{N} u_i$ be the BS length measured in pages, $L_I \in [N, L_I^{\max}],\ L_I \in Z$, N be the total number of pages, and $L_I^{\max}$

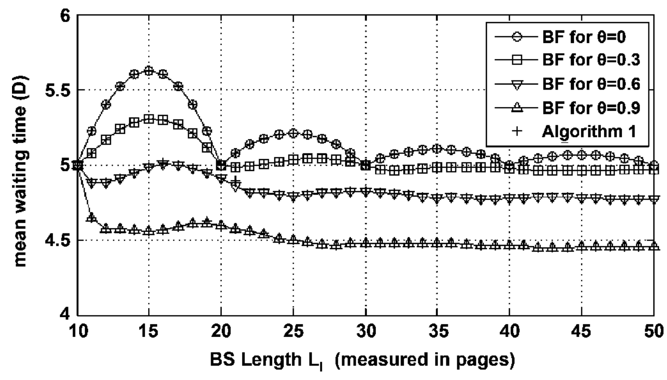

Fig. 3. Mean waiting time $\bar{D}$ as a function of the broadcast schedule length L_I measured in pages, as produced by brute force (BF) and Algorithm 1. Zipf pdf is assumed with $\theta = [0,\ 0.3,\ 0.6,\ 0.9]$. Number of items is $N = 10$.

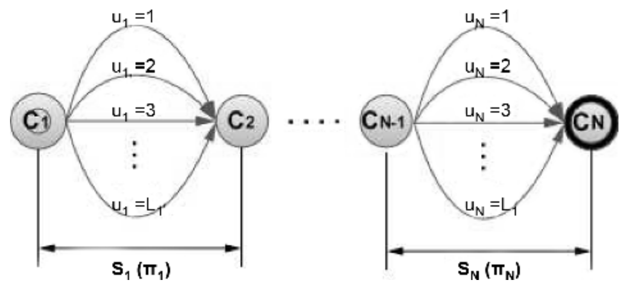

Fig. 4. Analogy of the BS length minimization with the optimal vehicle routing problem. A vehicle traverses a series of checkpoints $C_i,\ i = 1 \cdots N$, located at $S_i = \pi_i$ normalized distance units away from each other. Starting at C_1 with fuel L_I, the vehicle must reach C_N as soon as possible with the available fuel, making proper speed choices along the way. For simplicity, the fuel consumption is assumed to be proportional (1:1) to the chosen speed.

be the maximum examined value. For $L_I = N$, the BS is flat, and the speed of each page unary. For every $L_I > N$, exactly

$$P = L_I - N \tag{5}$$

unary speed increments must take place. The objective is to distribute the P increments optimally among the pages.

Since the ideal speeds u_i^{ideal} are given by (3), a serial distribution of increments can be employed, where the next unary raise affects the page p_j for which it holds that

$$\left|u_j - u_j^{\text{ideal}}\right| > \left|u_i - u_i^{\text{ideal}}\right| \qquad \forall i \neq j. \tag{6}$$

The procedure is formulated as Algorithm 1. Implementation with strict weak ordered maps [30] provides $O(\log N)$ complexity.

Algorithm 1 Calculating Optimal u_i for Given L_I

```
Input: The pages $p_i : \{\pi_i\},\ i = 1 \ldots N$ and the BS length $L_I \geq N$.
Output: The optimal page speeds $u_i^{\text{opt}}$.
1:  Set $P = L_I - N$.
2:  Set $u_i^{\text{opt}} = 1,\ i = 1 \ldots N$.
3:  Set $u_i^{\text{ideal}},\ i = 1 \ldots N$ by (3).
4:  for $i := 1$ to $P$ do
5:      Calculate page index $j$ by (6).
6:      $u_j^{\text{opt}} = u_j^{\text{opt}} + 1$
7:  end for
```

While the serial distribution is not analytically optimal, it typically achieves 99%–100% of the best possible performance. To demonstrate this fact, in Fig. 3, Algorithm 1 is compared $\bar{D}$-wise to the optimal results produced by brute force, i.e., trying all possible combinations of speeds and choosing the one achieving the minimum $\bar{D}$. The results coincide, with insignificant exceptions.

An interesting remark stems from the fact that the system quickly reaches a point $(L_I \approx 3 \cdot N)$, where 95%–100% of the possible waiting time has been achieved. This holds for any number of pages and any pdf***["probability density function"? Pls define]***: results are identical to that of Fig. 3. The system then tends to oscillate around the minimum of (4). The oscillation fades as pdf skewness and L_I increases.

The aforementioned remark has a direct impact on the overall cost of the system. The generic consequences can be better understood if we map the scheduling problem to the case of optimal vehicle routing described in Fig. 4. Minimizing L essentially translates to reaching C_N at minimal delay $D = \sum_{i=1}^{N} \Delta t_i = \sum_{i=1}^{N} \frac{s_i}{u_i}$—notice the similarity to (2)—with only a fraction of the required fuel. Further increasing L_I is an ineffective waste of resources.

The broadcasting-specific cost-related consequences include the following.

- The smaller the BS, the less computing time/processing power is required to construct it.
- A minimal BS has a higher probability of being calculated and sent from the scheduler to the beacon in one pass. A large BS must be calculated and forwarded in chunks, thus being constantly dependent on network congestion and requiring extra operational bandwidth.
- A small BS favors the implementation of cheap, dumb beacons that simply cache the binary data of the whole schedule without need for added complexity imposed by databases, data organization, and indexing/caching techniques.
- A BS small enough to be disclosed to the beacon in one brief pass in the form of indexes enables better scheduling of data fetching from the Web sites to the beacon.
- Minimal schedules may enable direct storage to RAM. Such a scheme is far less hardware-intensive than constant hard-disk accesses.

B. Page-Spacing Deviations: Altering the Pages' Speeds

The next form of deviation from ideality is that of page-spacing collisions. A purely theoretical way of overcoming this problem would be to alter the page speeds. However, as first shown in [5], the ideal page speeds already need to be tampered with in order to revert to an integer form. Moreover, as shown in Fig. 3, slight BS length alterations that can be caused by page-speed manipulation may degrade the mean waiting time. Thus, further tampering with the page speeds must be avoided as ineffective and performance degrading.

C. Impact of Local Concentrations of Single-Page Occurrences in the BS on the Client Waiting Time

As a next step, we examine whether the optimal scheduling algorithm should incline toward slight but distributed or major but local deviations of page spacings from their ideal values.

Consider a BS with fixed length L, in which a page p_i is repeated u_i times in total, with spacings

$$d_{ij} = d_{i1} \cdot a^{j-1}, \quad \{j = 1 \ldots u_i,\ a > 1\} \tag{7}$$

where j denotes the spacing between the jth and $(j-1)$th page occurrence. If $\frac{l_i}{L} \to 0$, it will hold that

$$L = \sum_{i=1}^{u_i} d_{ij} = d_{i1} \frac{a^{u_i} - 1}{a - 1}. \tag{8}$$

The mean delay time attributed to this page will be

$$\overline{D(p_i)} = \sum_{j=1}^{u_i} \frac{d_{ij}^2}{2L} \overset{\text{eq. (7)}}{=} \frac{1}{2L} d_{i1}^2 \frac{a^{2u_i} - 1}{a^2 - 1} \tag{9}$$

and because of (8)

$$\overline{D(p_i)} = \frac{L}{2} \frac{(a-1)(a^{u_i}+1)}{(a+1)(a^{u_i}-1)}. \tag{10}$$

It is trivial to show that (10) is a strictly rising function of a. Thus for

$$a' > a \Leftrightarrow \overline{D_{p_i}(a')} > \overline{D_{p_i}(a)}. \tag{11}$$

Since higher values of a indicate more major and local spacing deviations (L is fixed, and (7) represents a geometric progression), we conclude through (11) that slight, distributed, global spacing deviations should be preferred over sharp deviations of distinct page speeds.

D. Slight Page-Spacing Deviations versus Zero Padding

So far, it has been proved that tampering with the page speeds, in order to overcome the problems posed by the forced deviations from ideality, is ineffective and potentially performance-degrading. Instead, slight and distributed spacing deviations should be preferred. Introduced in [7], however, zero padding has been adopted to produce periodic BS with steady page spacings. The logic behind this approach is simple: Increase the BS length by a sufficient number of "dummy" pages in order to overcome the deviations problem, and then substitute them with e.g., the most popular of pages. Thus, the new question posed is whether zero padding should be preferred over distributed slight spacing deviations.

To this end, we consider a single page with spacings $d_1 = \left\lfloor \frac{L}{u_1} \right\rfloor$ and $d_1' = \left\lceil \frac{L}{u_1} \right\rceil$, similarly to the case depicted in Fig. 2. Should we define $\sigma = \text{modulo}(\frac{L}{u_1}),\ \sigma \in [1, u_1)$, then:

1) d_1 will be used exactly $(u_1 - \sigma)$ times;
2) while d_1' will be used exactly σ times;

over a BS of fixed length L. It is obvious that

$$d_1' = d_1 + 1. \tag{12}$$

Then, if $\frac{l_i}{L} \to 0$, it will hold that

$$\begin{aligned} L &= \sum_{j=1}^{u_1} d_{1j} \\ &= d_1 \times (u_1 - \sigma) + d_1' \times \sigma \overset{\text{eq. (12)}}{\Rightarrow} d_1 = \frac{L - \sigma}{u_1} \end{aligned} \tag{13}$$

while the mean client waiting time for this page is

$$\begin{aligned} \overline{D(p_i)} &= \frac{d_1^2 \times (u_1 - \sigma) + d_1'^2 \times \sigma}{2L} \overset{\text{eq. (12,13)}}{\Rightarrow} \\ \overline{D(p_i)} &= \frac{L}{2u_1} - \frac{\sigma^2}{2Lu_1} + \frac{\sigma u_1}{2Lu_1}. \end{aligned} \tag{14}$$

On the the other hand, should zero padding be used, L should be increased by $(u_1 - \sigma)$. Then, $L' = L + u_1 - \sigma$, and the spacing becomes steady and equal to $d_1 = \frac{L'}{u_1}$. The mean client waiting time in this case is

$$\overline{D_Z(p_i)} = \frac{L'}{2u_1} = \frac{L}{2u_1} + \frac{1}{2} - \frac{\sigma}{2u_1}. \tag{15}$$

It is now easy to show that

$$\overline{D_Z(p_i)} - \overline{D(p_i)} = \frac{(L - \sigma)(u_1 - \sigma)}{2Lu_1} > 0 \tag{16}$$

since $\sigma \in [1, u_1)$. Therefore, $\overline{D_Z(p_i)} > \overline{D(p_i)}$ in any case.

Even though this analysis proves the inferiority of zero padding, it does not take into account the spacing conflicts described in Section II-D. Yet, the zero padding method introduced in [7] is known to overcome this problem as well. However, this method has severe weaknesses that can become evident through a simple example.

1) Brief Method Description: Begin by sorting the pages in descending request probability fashion. Perform grouping by page speed. Calculate the least common multiple (LCM) of the page groups' speeds. Split each group of pages into $\frac{\text{LCM}}{u_i}$ chunks, employing zero padding in the process where necessary. Finally, broadcast the chunks in a round-robin manner.

Consider the following example. Three pages have request probabilities $\pi_{1\ldots3} = \{\frac{3}{4}, \frac{1}{6}, \frac{1}{12}\}$. For an $L = 20$-pages-long BS, their approximate optimal speeds are $u_{1\ldots3} = \{11, 5, 4\}$, the least common multiple of which is 220. Thus, we have the following:

1) Group 1 (i.e., page p_1) must be split into 110 chunks (i.e., 109 dummy pages must be added to it).
2) Groups 2 and 3 are similarly expanded by 43 and 54 dummy pages correspondingly.

The new BS has length $L' = 209$ instead of the original $L = 20$ and corresponds to a mean client waiting time of $\overline{D'} = 11.55$ (2) instead of $\overline{D} = 1.22$ (4). Thus, the system's performance is reduced to a mere 10% of its ideal value.

In conclusion, zero padding should be avoided in data broadcasting due to serious performance penalties.

E. On the Necessity of Data Segmentation

It is interesting to examine the limitations of the analysis of [5], an overview of which has been given in Section II. [5] is

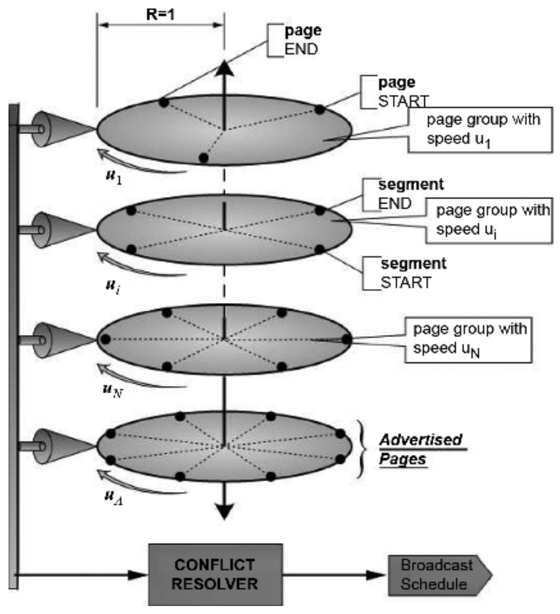

Fig. 5. CABS schedule constructor.

considered the analytically optimal state of the art in the field of broadcast scheduling.

In the event of uneven page-occurrence spacings, the mean client waiting time is given by

$$\bar{D} = \sum_{i=1}^{N} \pi_i \sum_{j=1}^{u_i} \frac{d_{ij}^2}{2L} \tag{17}$$

where d_{ij} is the spacing between the jth and $(j-1)$th occurrence of page p_i in the BS. Spacings are measured from one page-occurrence start (i.e., header) to the next. It has been proven [1] that (17) is minimized when the spacings of each page are constant, $d_{ij} = \frac{L}{u_i}\ \forall j$. Equation (17) is then reduced to the form of (2). However, $\frac{L}{u_i}$ spacings can be impossible in the event of uneven page sizes l_i.

Consider three pages $p_{1\ldots3} : [\{u_1 = 3,\ l_1 = 1\},\ \{u_2 = 2,\ l_2 = 1\},\ \{u_3 = 1,\ l_3 = 3000\}]$. The corresponding constant spacings would be $d_1 = \frac{3002}{3} \approx 1000$, $d_2 = 1501$, $d_3 = 3002$. However, p_3 fits nowhere between the occurrences of either p_1 or p_2. The analysis of [5] is then essentially invalidated, yielding $\overline{D(p_1)} = \frac{d_1}{2} = 500$, $\overline{D(p_2)} = 750.5$, $\overline{D(p_3)} = 1501$ instead of the real values $\overline{D(p_1)^r} = 1501$, $\overline{D(p_2)^r} = 1501$, $\overline{D(p_3)^r} = 1501$ produced by (17).

The scheduling algorithm of [5] attempts to tackle this deficiency by increasing L infinitely, effectively incorporating thousands of redundant copies of pages $p_1,\ p_2$ in the BS. Moreover, simulations run in [5] consider page sizes $l_i \in [1, 10]$ while $L \to \infty$. This affects the ratio $\frac{l_i}{L} \to 0$, essentially reducing the scenario to that of equally sized pages. Indeed, the results of the scheduling algorithm in simulations of [5] are almost identical for both unevenly and evenly sized pages.

A better approach would be to segment pages to smaller packets. Large pages then become "filling material" between occurrences of small pages. Thus, L is unaffected, and spacings of $\frac{L}{u_i}$ become possible. Notice that the client waiting time is not affected either: The page headers are placed at their optimal positions. The technique is essentially an application of time multiplexing. The channel bandwidth is divided among the clients with better temporal granularity. While the present paper does not aim at introducing novel multiplexing schemes, a safe packet size limit can be set as

$$\Pi_{\min} = \min\{l_i\} \tag{18}$$

while a less segmentation-greedy packet size is obviously

$$\Pi = \frac{1}{N}\min\{d_i\}. \tag{19}$$

Packet sizes do not have an impact on waiting time, provided that the aforementioned problem is avoided, but do affect client data receival rate, smoothness, and noise resilience. Optimizing packet size for these purposes goes beyond the scope of the present work.

F. Priorities Over Conflict

Data segmentation and slight spacing deviations have been proven to be beneficial to the performance of a push system. However, the problem posed by conflicting spacings discussed in Section II-D has not yet been resolved.

As it has already been stated, conflicting spacings are unavoidable when zero padding is not used. Thus, measures must be taken to prioritize pages in the event of conflict. Equation (2) states that pages with higher $\frac{\pi_i}{u_i}$ value have heavier impact on the mean waiting time. Thus, the impact factor of a page p_i can be defined as

$$I_i = \frac{\pi_i}{u_i}. \tag{20}$$

Therefore, it is logical that in the event of conflicting page or packet spacings, items with higher impact factor should take priority and be handled accordingly.

G. Theoretical Summary

Through the analysis and remarks of the preceding sections, the following has become evident.

1) An optimal scheduler should provide optimal client waiting times with minimal BS length.
2) Each page-spacing value in the BS must deviate as little as possible from its ideal values in a uniform, distributed way. If not possible due to excessive $\frac{l_i}{L}$ ratio, big pages should be split into smaller packets. Zero padding should be avoided.
3) Finally, on the event of spacing conflicts, pages or packets should be prioritized according to their impact factor defined by (20).

IV. Cost-Effective, Adaptivity-Oriented Broadcast Scheduler

Having defined its optimal characteristics, the corresponding scheduler can be easily formulated.

The proposed Cost-effective, Adaptivity-oriented Broadcast Scheduler (CABS) employs a rotating disks scheme, depicted in Fig. 5.

We consider pages $p_i : \{\pi_i,\ l_i\},\ i = 1 \ldots N$. The optimal page speeds u_i^{opt} and L are set by Algorithm 1. The pages are clustered by their $\frac{l_i}{L}$ ratio through hierarchical clustering [31] with ending criterion

$$\left|\frac{l_i}{L}-\frac{l_c}{L}\right| \leq \frac{\Pi}{L} \qquad \forall p_i \in C \tag{21}$$

where C denotes the cluster owning page p_i, l_c indicates the page length of the corresponding centroid, and Π is the packet size of (19). Each formed cluster is then divided into group of pages with equal speeds. Each group has its contents segmented and is placed on the periphery of a circle with unary radius. The disks are set to rotate around a common axis with their corresponding speeds, and a set of stationary heads detects the start/end of pages or packets as shown in Fig. 5. In the event of the detection of a page/packet start/end, the page/packet is immediately broadcast. Should two or more pages conflict, they are sorted by descending impact factor I_i and are broadcast in that order. To minimize the probability of a conflict occurring, the starting angle of each disk is randomized in the range $[0, 2\pi]$. The procedure lasts for a time interval of $T = \frac{2\pi R}{\min\{u_i\}} \min\{u_i\} = 2\pi R = 2\pi$, at which point a BS of exactly $L = \sum_{i=1}^{N} u_i \cdot l_i$ has been produced.

The procedure is formulated as Algorithm 2.

Algorithm 2 The CABS Schedule Constructor

```
Input: The pages $p_i : \{\pi_i, l_i\},\ i = 1 \ldots N$.
Output: The broadcast schedule.
 1: Set $u_i^{\text{opt}}$ by Algorithm 1.
 2: Perform Hierarchical Clustering of pages by $\frac{l_i}{L}$ and
    criterion of (21).
 3: Group pages in the clusters by $u_i$.
 4: Set $NoD$ as the number of produced groups.
 5: Segment pages by (19).
 6: Set initial position of heads:
    $S_0^i = \text{random}\{[0, 2\pi]\},\ i = 1 \ldots NoD$
 7: Set $\Delta t = \min\{\frac{2\pi}{\text{DiskSize}_i \cdot u_i},\ i = 1 \ldots NoD\}$.
 8: Set $T = 2\pi$.
    Create the broadcast schedule
 9: for $t := \Delta t$ to $T$, step by $\Delta t$ do
10:     Calculate current head positions:
        $S_t^i = S_0^i + u_i \cdot t,\ i = 1 \ldots NoD$
11:     Collect pages/segments whose boundaries have
        been crossed in $[t, t - \Delta t]$, in array $B$.
12:     Sort $B$ by descending impact factors $I_i$.
13:     Broadcast items of $B$.
14: end for
```

As an extra feature, CABS can also easily set custom speeds for any custom group (i.e., disk), e.g., when advertising certain pages with a guaranteed client waiting time variance [32].

V. Performance and Cost Assessment

Even though the proposed algorithm adheres to the guidelines of the theoretical analysis, it is imperative that its performance be assessed and compared to other approaches.

To this end, the CABS algorithm is compared to the analytically optimal scheduling algorithm introduced in [5] (SOA) and the latest advancement (BDISK-SOA) of the Broadcast Disks family of algorithms [7] (BDISK-CLASSIC). The former, though introduced in 1999, still represents the top-performing algorithm client waiting-time-wise. BDISK-CLASSIC represents a very popular scheduling approach, as discussed in Section I-A. BDISK-SOA was presented by the authors in [25] and [26], where it surpassed the previous state of the art. Performance of the ideal scheduler of Section II-C (IDEAL) is given where applicable.

TABLE I
Feature Support of Scheduling Algorithms

Feature	SOA*1	BDISKS, BDISKS-SOA*2	CABS
Periodic BS	no	yes	yes
Varying page size	yes	no	yes
Data segmentation	no	no	yes
Page advertising	no	no	yes

*1: *The analytically optimal algorithm presented in [5]*
*2: *The algorithms of [7, 25]* respectively

The performance assessment is twofold.

1) The classic criteria, i.e., the mean client waiting times achieved by all three algorithms are compared for a variety of different client probabilistic configurations.
2) The length of the produced broadcast schedules and the CPU times required to construct them are compared as a metric of the system's implementation and operational cost, as described in Section III-A.

Notice that to the best of our knowledge, the second criterion is overlooked in the totality of the research on push systems. As the results prove, though, it is exactly these criteria that determine whether an algorithm can be implemented in the real world: Push systems are *de facto* used as a minimal-cost solution. Should an algorithm require exquisite, powerful, and expensive hardware to run or excessive operational resources, it would be wise to abandon the push logic altogether and resort to classic client–server architectures that are bound to perform better.

A. Simulation Setup and Results

As shown in Table I, not all compared algorithms support all the features discussed in the theoretical analysis. Thus, two comparisons take place: All algorithms are initially compared in an evenly sized pages scenario. Then, only SOA and CABS are compared in the varying-page-sizes case.

The comparison is simulation-based, and the corresponding parameters are presented in Table II. The parameter values used are typically used in similar papers. Worthy of special note are the total number of pages N and the client probabilistic model.

A typical 3.5 G telephony cell can typically handle 10 000 P2P client connections. For the broadcasting scenario to make sense, a logical and feasible number of clients would thus be 30 000. Assuming that the clients share a common interest for a good 60% of the available pages—broadcasting would have dubious meaning otherwise—and their access patterns are comprised of 10 pages each, the total number of items is $N = 30\,000 \cdot 10 \cdot (1 - 0.6) = 120\,000$.

The only client pdf used in all related work is the Zipf model. This global choice has a sound basis: All pages are typically sorted by descending popularity π_i. Should this be applied to

TABLE II
SIMULATION PARAMETERS

Parameter	**Value**
TOPOLOGY	STAR $\{1$ node-server: n wireless clients$\}$
Client Query p.d.f	ZIPF: $\pi_i = \frac{\sum_{i=1}^{N} \frac{1}{i^\theta}}{i^\theta}, i = 1 \dots N$
θ	$\{[0.2:0.1:1.8]\} \cap \overline{\{1\}}$
SYSTEM STATE	BS Renewal triggered (see Section II-B)
PAGE SPEEDS DEFINITION	SOA: $round(u_i^{ideal})$ CABS: Algorithm 1 BDISKS-SOA: Procedure of [26] BDISKS: $round(u_i^{ideal})$
Number of pages	$N = 120,000$
Simulation Duration	$600,000$ client queries
ThinkTime*	2 page segments

* *Introduced in [7]: upon receiving a wanted page, the client halts queries for this time interval.*

TABLE III
CPU TIME ASSESSMENT SPECIFICATIONS

Parameter	**Value**
Test CPU model	x86 Family 6 model 15 Stepping 11 GenuineIntel
Number of cores	4× 2672MHz
Test RAM	4GB DDR2 800 MHz
Operating System Version	Windows XP Professional 5.1.2600 SP3 Build 2600
Programming language Environment	Single-threaded MATLAB M-code MATLAB R2009b
CPU Usage during execution	Steady 100% on one dedicated core*
Application Priority	13 ("High")

*: *Measured by means of Windows Task Manager*

any of the other popular distributions, e.g., Pareto, Gaussian, they can precisely be substituted by a Zipf pdf with a proper θ value. To this end, the authors maintain the Zipf convention and examine a wide set of θ values. As a side note, analysis of this paper is pdf-independent.

Concerning the calculation of the CPU time required by each algorithm, two approaches are most popular:

1) calculate and compare the complexity $O(.)$ of each algorithm;
2) implement each algorithm in assembly and count CPU cycles required by each one.

In the present case, the first option is not a choice since the complexity for SOA is not provided [5], and the complexity of CABS cannot be calculated since the algorithm employs grouping of pages by their speed u_i, which is function of their access probabilities (i.e., random). The assembly implementation choice also has severe weaknesses.

1) Any assembly implementation is too hardware-specific to be verifiable.
2) A few decades of higher language code typically correspond to several thousands lines of assembly code. Thus, implementation optimization is a problem that is bound to be more influential in the case of assembly.
3) Handwritten assembly code is obviously much less reliable than the corresponding one produced by a well-known and globally trusted compiler or interpreter.

Thus, all algorithms were implemented in MATLAB as standalone functions. The CPU time spent in each one was then measured [33], [34] by the MATLAB Code Profiler tool. The reasons for this choice were the following.

1) Each algorithm has typically an implementation of 10 or less lines of MATLAB code, leaving little, if any, room for implementation issues to be considered.
2) The MATLAB interpreter and profiler are popular and well-known in the scientific community globally.
3) Only one official MATLAB interpreter exists, as opposed to other programming languages like $\mathrm{C/C++}$ or Fortran. The results are thus more verifiable and trustworthy.
4) The MATLAB version used supports JIT compiling, providing execution times comparable to corresponding $\mathrm{C/C++}$ implementation.

Throughout the execution time, great care was taken to keep the CPU usage steady at 100% and dedicated to the MATLAB interpreter. Specification details are given in Table III.

Results are displayed in Figs. 6–8, depicting achieved $\overline{D}$, L, and CPU times of the compared algorithms.

For the unevenly-sized-pages scenario, we assume $N = 50$ pages with sizes uniformly distributed in $[0.5, 1.5]$. One large page with index $I \in [1, 50]$ is selected, and its size is set to 50 000. The ratio $\frac{1}{50\,000}$ refers to the size of a typical 2-kB HTML page divided by the size of a streaming video file of 100 MB. Zipf distribution of page popularity is assumed once more, with the θ-parameter ranging again in $[0.2, 1.8]$. Results for this scenario are displayed in Fig. 9, depicting achieved $\bar{D}$ of the algorithms supporting unevenly sized pages (SOA and CABS).

B. Remarks

The waiting times achieved by the compared algorithms (Figs. 6 and 9) indicate that CABS is the only algorithm achieving ideality in both evenly- and unevenly-sized-pages scenarios. Notice that apart from CABS, only SOA supports unevenly sized pages, and in accordance with the remarks of Section III-E, fails to achieve optimal results even by a difference of $2.7 \cdot 10^7$ length units. SOA tends to perform better as the large page becomes less popular either by increasing the large page index or the θ-parameter of the Zipf pdf. Its performance, however, is not satisfactory in the majority of the cases.

In the evenly-sized-pages scenario, CABS and SOA achieve ideality (Fig. 6). However, CABS requires $\approx$ 20 s on the test machine to calculate the required BS in one pass, while SOA requires an aggregate amount of days (Fig. 8). This effectively means that to answer merely 20 queries for each of the 30 000 clients (thus the total of 600 000 queries of the simulation), SOA would require several days of optimization, thus creating a major bottleneck. The only possible solution would be to run SOA on specialized, powerful, and expensive hardware. Notice that SOA cannot be divided to threads.

BS sizes achieved by SOA and CABS (Fig. 7) denote a difference of a factor of 10^5. CABS achieves a value of one

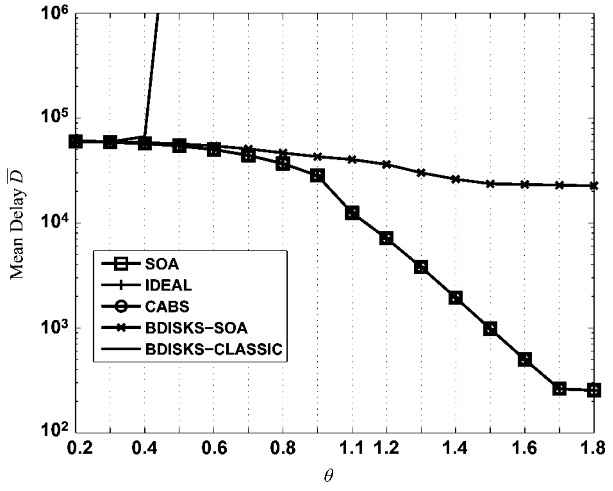

Fig. 6. Mean client waiting times achieved by the competing algorithms in the evenly-sized-pages scenario.

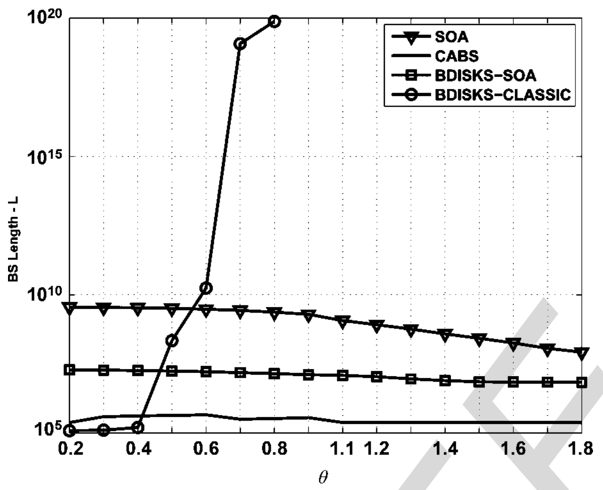

Fig. 7. Illustration of the schedule sizes required by each algorithm.

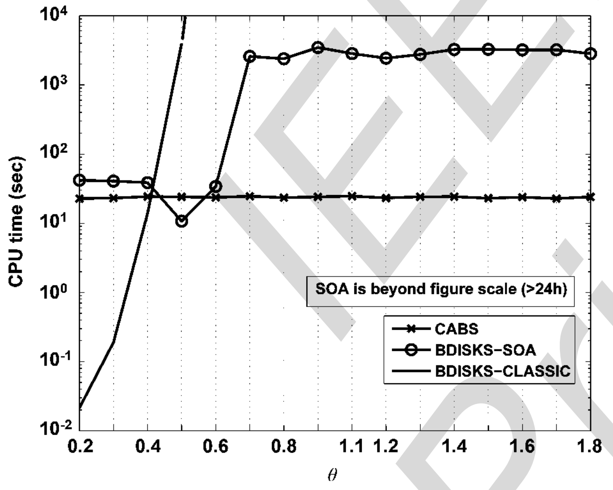

Fig. 8. CPU time required by each algorithm on the test machine.

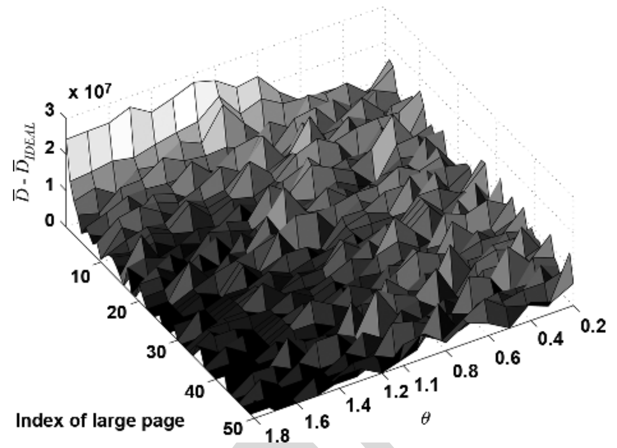

Fig. 9. Deviation of mean client waiting times achieved by SOA from the IDEAL scheduler unevenly-sized-pages scenario. CABS achieves ≈ 0 deviation in all cases.

to three times the database size. SOA, requiring 10^5 times greater BS, nullifies all cost-effective advantages mentioned in Section III-A.

BDISKS-CLASSIC and BDISKS-SOA only refer to the evenly-sized-pages scenario. They both offer suboptimal performance, with BDISKS-SOA being the only one performing acceptably (Fig. 6). BDISKS-CLASSIC performs well only for $\theta \in [0.2, 0.4]$, where the optimal schedule is nearly flat with no need for exquisite optimization. For greater θ values, both the CPU times and BS sizes of BDISKS-CLASSIC render it nonrealizable. BDISKS-SOA greatly outperforms SOA in terms of resources (Figs. 7 and 8), but not CABS. Finally, the least common multiple employed by the BDISKS-x algorithms may sporadically yield conveniently small values. In this case, the required CPU time is minimized as shown in Fig. 8. However, this only happens for small θ values where the algorithms achieve the performance of a nearly flat broadcast schedule (Fig. 6).

VI. Conclusion

This paper focused on the real-world implementation perspectives of algorithms designed to construct the schedule of broadcast-based (or push-based) systems. Push systems are primarily designed as a cheap telecommunications solution, and their applications typically include hospital and airport informative systems, instant messaging services, interactive television, and more. However, the main research on the field has focused on performance optimization, disregarding the most vital characteristic a push system should possess: low implementation cost.

By theoretically analyzing many aspects of the broadcast scheduling procedure, a new broadcast scheduler constructor algorithm, the Cost-effective Adaptivity-oriented Broadcast Scheduler (CABS) has been proposed. CABS was compared in terms of broadcast schedule size and CPU cost with the current state-of-the-art SOA algorithm, and algorithms of the Broadcast Disks model family. The results have proven that these algorithms have neglected the cost attributes to the point that may be only conditionally realizable.

As a general paper conclusion, it is shown that any push-related study should take the implementation cost into account, apart from waiting-time-based performance. Successful compliance with the aforementioned criteria results in realistic and practically useful systems.

References

[1] C. Kenyon and N. Schabanel, "The data broadcast problem with non-uniform transmission times," in *Proc. 10th Annu. ACM-SIAM SODA*, Baltimore, MD, Jan. 1999, pp. 547–556.

[2] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieberaruch, "Minimizing service and operation costs of periodic scheduling," in *Proc. 9th Annu. ACM-SIAM SODA*, San Francisco, CA, Jan. 1998, pp. 11–20.

[3] M. Ammar and J. Wong, "On the optimality of cyclic transmission in teletext systems," *IEEE Trans. Commun.*, vol. 35, no. 1, pp. 68–73, 1987.

[4] S. Hameed and N. Vaidya, "Efficient algorithms for scheduling data broadcast," *Wireless Netw.*, vol. 5, no. 3, pp. 183–193, 1999.

[5] N. Vaidya and S. Hameed, "Scheduling data broadcast in asymmetric communication environments," *Wireless Netw.*, vol. 5, no. 3, pp. 171–182, 1999.

[6] N. Schabanel, "The data broadcast problem with preemption," in *Proc. 17th STACS*, Feb. 2000, pp. 181–192.

[7] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data management for asymmetric communication environments," in *Proc. ACM SIGMOD*, New York, May 1995, pp. 199–210.

[8] K. Wu, P. Yu, and M. Chen, "Energy-efficient caching for bandwidth-limited wireless mobile computing," in *Proc. 12th IEEE ICDE*, Feb. 1996, pp. 335–343.

[9] M. Chen, P. Yu, and K. Wu, "Optimizing index allocation for sequential data broadcasting in wireless mobile computing," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 1, pp. 161–173, Jan.–Feb. 2003.

[10] J. Xu, W. C. Lee, X. Tang, Q. Gao, and S. Li, "An error-resilient and tunable distributed indexing scheme for wireless data broadcast," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 3, pp. 392–404, Mar. 2006.

[11] C. Hu and M. Chen, "Adaptive multichannel data dissemination: Support of dynamic traffic awareness and push-pull time balance," *IEEE Trans. Veh. Technol.*, vol. 54, no. 2, pp. 192–203, Mar. 2005.

[12] Y. Chang, C. Yang, and J. Shen, "A binary-number-based approach to data broadcasting in wireless information systems," in *Proc. IEEE Int. Conf. Wireless Netw.*, 2005, vol. 1, pp. 348–353.

[13] W. Wang and C. Ravishankar, "Adaptive data broadcasting in asymmetric communication environments," in *Proc. 8th IDEAS*, 2004, pp. 27–36.

[14] H. Leung, J. Yuen, K. Lam, and E. Chan, "Enhanced multiversion data broadcast scheme for time-constrained mobile computing systems," in *Proc. 13th DEXA*, 2002, pp. 747–751.

[15] T. F. Bowen, G. Gopal, G. Herman, T. Hickey, K. C. Lee, W. H. Mansfield, J. Raitz, and A. Weinrib, "The Datacycle architecture," *Commun. ACM*, vol. 35, no. 12, pp. 71–81, 1992.

[16] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham, "Efficient concurrency control for broadcast environment," in *Proc. ACM SIGMOD*, 1999, pp. 85–96.

[17] E. Pitoura and R. Chrusanthis, "Multiversion data broadcast," *IEEE Trans. Comput.*, vol. 51, no. 10, pp. 1224–1230, Oct. 2002.

[18] G. Lee, S. Lo, and A. Chen, "Data allocation on wireless broadcast channels for efficient query processing," *IEEE Trans. Comput.*, vol. 51, no. 10, pp. 1237–1252, Oct. 2002.

[19] J. Huang and M. Chen, "Broadcasting dependent data for ordered queries without replication in a multi-channel mobile environment," in *Proc. 19th IEEE ICDE*, Mar. 2003, pp. 692–694.

[20] G. Lee, M. Yeh, S. Lo, and A. Chen, "A strategy for efficient access of multiple data items in mobile environments," in *Proc. 3rd Int. Conf. Mobile Data Manage.*, 2002, pp. 71–78.

[21] J. Huang and M. Chen, "Broadcast program generation for unordered queries with data replication," in *Proc. 18th ACM SAC*, Mar. 2003, pp. 866–870.

[22] F. Martinwz, U. Gonzalez, and I. Stojmenovic, "A parallel hill climbing algorithm for pushing dependent data in clients-providers-servers systems," in *Proc. 7th IEEE ISCC*, 2002, pp. 611–616.

[23] A. Bar-Noy, J. Naor, and B. Schiber, "Pushing dependent data in clients-providers-servers systems," in *Proc. ACM MobiCom*, 2000, pp. 222–230.

[24] C. Liaskos, S. Petridou, G. Papadimitriou, P. Nicopolitidis, M. Obaidat, and A. Pomportsis, "Clustering-driven wireless data broadcasting," *IEEE Wireless Commun. Mag.*, vol. 16, no. 6, pp. 80–87, Dec. 2009.

[25] C. Liaskos, S. Petridou, P. Nicopolitidis, and G. Papadimitriou, "On the analytical performance optimization of wireless data broadcasting," *IEEE Trans. Veh. Technol.*, vol. 59, no. 2, pp. 884–895, Feb. 2010.

[26] C. Liaskos, S. Petridou, and G. Papadimitriou, "Cost-aware wireless data broadcasting," *IEEE Trans. Broadcast.*, vol. 56, no. 1, pp. 66–76, Mar. 2010.

[27] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Using learning automata for adaptive push-based data broadcasting in asymmetric wireless environments," *IEEE Trans. Veh. Technol.*, vol. 51, no. 6, pp. 1652–1660, Nov. 2002.

[28] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Exploiting locality of demand to improve the performance of wireless data broadcasting," *IEEE Trans. Veh. Technol.*, vol. 55, no. 4, pp. 1347–1361, Jul. 2006.

[29] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Multiple-antenna data broadcasting for environments with locality of demand," *IEEE Trans. Veh. Technol.*, vol. 56, no. 5, pt. 1, pp. 2807–2816, Sep. 2007.

[30] P. Fishburn, "Intransitive indifference in preference theory: A survey," *Oper. Res.*, vol. 18, no. 2, pp. 207–228.

[31] A. Jain and R. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ: Prentice-Hall, 1988.

[32] S. J. N. Vaidya, "Scheduling algorithms for a data broadcast system: Minimizing variance of the response time," Comput. Sci. Dept., Texas A&M Univ., College Station, TX, TR 98-005, 1998.

[33] E. Alexandre, A. Pena, and M. Sobreira, "Low-complexity bit-allocation algorithm for MPEG AAC audio coders," *IEEE Signal Process. Lett.*, vol. 12, no. 12, pp. 824–826, Dec. 2005.

[34] R. Sukkar, S. Herman, A. Setlur, and C. Mitchell, "Reducing computational complexity and response latency through the detection of contentless frames," in *Proc. IEEE ICASSP*, Antwerp, Belgium, Jun. 2000, pp. 3751–3754.

Christos K. Liaskos (M'08) received the Diploma degree in electrical and computer engineering in 2004 and the M.S. degree in medical informatics in 2008 from the Aristotle University of Thessaloniki, Thessaloniki, Greece, where he is currently pursuing the Ph.D. degree in communication networks.

His research interests include wireless networks, neural networks, and fractal compression algorithms.

Sophia G. Petridou (M'08) received the Diploma and Ph.D. degrees in computer science from the Aristotle University of Thessaloniki, Thessaloniki, Greece, in 2000 and 2008, respectively.

[Current position at university?]Her research interests include clustering and optical and wireless networks.

Georgios I. Papadimitriou (M'89–SM'02) received the Diploma and Ph.D. degrees in computer engineering from the University of Patras, Patras, Greece, in 1989 and 1994, respectively.

In 1997, he joined the faculty of the Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece, where he is currently an Associate Professor. He is coauthor of three books published by Wiley. He is author or coauthor of 90 journal and 100 conference papers. His main research interests include optical networks and wireless networks.

Prof. Papadimitriou is Associate Editor of five IEEE journals.