BROADCAST SCHEDULING ALGORITHMS FOR WIRELESS DATA DISSEMINATION[*]

JIANLIANG XU[†] AND JIANGCHUAN LIU[‡]

Abstract. This chapter introduces broadcast scheduling algorithms for wireless data dissemination. We classify them into push-based, on-demand, and hybrid scheduling and provide a survey on the state-of-the-art solutions. Moreover, we describe a newly developed algorithm, called Slack Inverse Number of pending requests (SIN), for time-critical on-demand scheduling, where user requests are associated with time constraints. We show the experimental results that prove the superiority of the SIN algorithm over the existing algorithms. In addition, we briefly review some other issues of wireless data dissemination, including fault-tolerant broadcast, updates handling, air indexing, and client cache management.

Key words. Wireless data dissemination, scheduling, broadcast, on-demand, push, content delivery.

AMS subject classifications. ?

1. Introduction. Owing to the widespread deployment of wireless networks and fast-improving capabilities of mobile devices, we have been seeing increasing interest on wireless data services from both industrial and academic communities in recent years. There are two fundamental information delivery approaches for wireless data services: *point-to-point access* and *broadcast* [16, 40]. In point-to-point access, a logical channel is established between the client and the server. Queries are submitted to the server and results are returned to the client in much the same way as in a wired network. In broadcast, on the other hand, data are accessible simultaneously by all clients residing in the broadcast area. The server determines the data to broadcast based on explicit client requests or historical statistics.

Point-to-point access is particularly suitable for light-loaded systems when contention for wireless channels and server processing is not severe. However, as the number of users increases, the system performance deteriorates rapidly. Compared with point-to-point access, broadcast is a very attractive alternative for several reasons [1, 21]. First, it allows simultaneous access by an arbitrary number of mobile clients and thus allows efficient usage of the scarce wireless bandwidth. Second, mobile wireless environments are characterized by asymmetric communication, i.e., the *downlink* communication capacity is much greater than the *uplink* communication capacity. Data broadcast can take advantage of the large downlink capacity when delivering data to clients. Third, a wireless communication system essentially employs a broadcast mechanism to deliver information. Thus, data broadcast can be implemented without introducing any system overhead.

Wireless data broadcast services have been available as commercial products for many years (e.g., StarBand [47] and Hughes Network [48]). In particular, recent announcement of the *smart personal objects technology* (SPOT) by Microsoft [46], has further ascertained the industrial interest on and feasibility of utilizing broadcast for wireless data services. With a continuous broadcast network (called DirectBand

*The writing of this chapter was supported by grants from Hong Kong Baptist University (Grant FRG/02-03/II-34) and Hong Kong Research Grants Council (Grant CUHK C001/2050312).

†Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, KLN, Hong Kong

‡Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong

Network) using FM radio subcarrier frequencies, SPOT-based devices (e.g., PDAs and watches) can continuously receive timely information such as stock quotes, airline schedules, local news, weather, and traffic information. This chapter shall focus itself on wireless data dissemination with broadcast.

A key issue in data broadcast systems is the *scheduling algorithm*, which determines what is to be broadcast by the server and when. There are three kinds of broadcast models, namely *push-based* broadcast, *on-demand* (or *pull-based*) broadcast, and *hybrid* broadcast. In push-based broadcast [1, 15, 19], the server disseminates information using a periodic/aperiodic broadcast program (generally without any intervention of clients); in on-demand broadcast [5, 6], the server disseminates information based on the outstanding requests submitted by clients; in hybrid broadcast [4, 20, 23], push-based broadcast and on-demand data deliveries are combined to complement each other. Consequently, there are three kinds of data scheduling algorithms, (i.e., *push-based scheduling*, *on-demand scheduling*, and *hybrid scheduling*), corresponding to the three data broadcast models.

The rest of this chapter is organized as follows. Section 2 discusses various push-based, on-demand, and hybrid scheduling algorithms. In Section 3, a new scheduling algorithm for time-critical on-demand broadcast is introduced. Section 4 discusses some other issues of wireless data dissemination, such as fault-tolerant broadcast, updates handling, and air indexing. Finally, this chapter is summarized in Section 5.

2. Data Scheduling Algorithms.

2.1. Push-based Data Scheduling.

In push-based data broadcast, the server broadcasts data proactively to all clients according to the broadcast program generated by the data scheduling algorithm. The broadcast program essentially determines the order and frequencies that the data items are broadcast in. The scheduling algorithm may make use of precompiled access profiles in determining the broadcast program. In the following, four typical methods for push-based data scheduling are described, namely *flat broadcast*, *probabilistic-based broadcast*, *broadcast disks*, and *optimal scheduling.*

2.1.1. Flat Broadcast.

The simplest scheme for data scheduling is flat broadcast. With a flat broadcast program, all data items are broadcast in a round-robin manner. The average access time for every data item is the same, i.e., half of the broadcast cycle. This scheme is simple, but its performance is poor in terms of average access time when data access probabilities are skewed.

2.1.2. Probabilistic-based Broadcast.

To improve performance for skewed data access, the probabilistic-based broadcast [36] selects an item i for inclusion in the broadcast program with probability f_i, where f_i is determined by the access probabilities of the items. The best setting for f_i is given by the following formula [36]:

$$f_i = \frac{\sqrt{q_i}}{\sum_{j=1}^{N} \sqrt{q_j}}, \tag{2.1}$$

where q_j is the access probability for item j, and N is the number of items in the database.

A drawback of the probabilistic-based broadcast approach is that it may have an arbitrarily large access time for a data item. Furthermore, this scheme shows inferior performance to other algorithms for skewed broadcast [36].

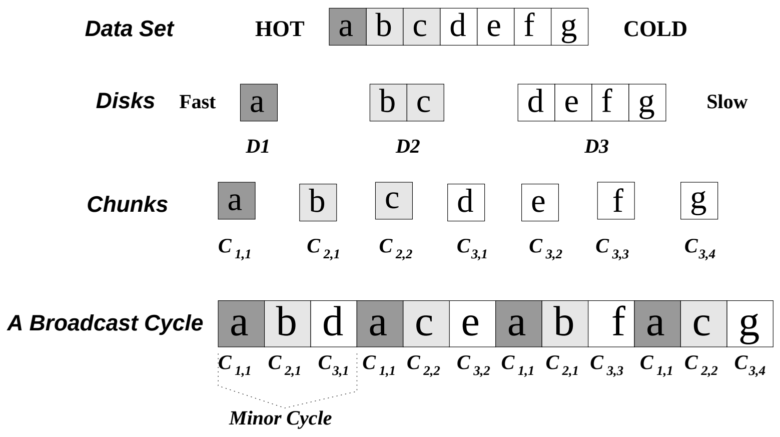
Data Set
HOT
a b c d e f g
COLD
Disks
Fast
a
b c
d e f g
Slow
D1
D2
D3
Chunks
a b c d e f g
$C_{1,1}$ $C_{2,1}$ $C_{2,2}$ $C_{3,1}$ $C_{3,2}$ $C_{3,3}$ $C_{3,4}$
A Broadcast Cycle
a b d a c e a b f a c g
$C_{1,1}$ $C_{2,1}$ $C_{3,1}$ $C_{1,1}$ $C_{2,2}$ $C_{3,2}$ $C_{1,1}$ $C_{2,1}$ $C_{3,3}$ $C_{1,1}$ $C_{2,2}$ $C_{3,4}$
Minor Cycle

FIG. 2.1. *An Example of a Seven-item, Three-disk Broadcast Program*

2.1.3. Broadcast Disks. A hierarchical dissemination architecture, called *Broadcast Disk* (Bdisk), was introduced in [1]. Data items are assigned to different *logical* disks so that data items in the same range of access probabilities are grouped on the same disk. Data items are then selected from the disks for broadcast according to the relative broadcast frequencies assigned to the disks. This is achieved by further dividing each disk into smaller, equal-size units called *chunks*, broadcasting a chunk from each disk each time, and cycling through all the chunks sequentially over all the disks. A *minor cycle* is defined as a sub-cycle consisting of one chunk from each disk. Consequently, data items in a minor cycle are repeated only once. The number of minor cycles in a broadcast cycle equals the Least Common Multiple (LCM) of the relative broadcast frequencies of the disks. Conceptually, the disks can be conceived as real physical disks spinning at different speeds, with the faster disks placing more instances of their data items on the broadcast channel. The algorithm that generates broadcast disks is given below.

Algorithm 1 Broadcast Disks Generation Algorithm.

```
1: Order the items in decreasing order of access popularities
2: Allocate items in the same range of access probabilities on a different disk
3: Choose the relative broadcast frequency $rel\_freq(i)$ (in integer) for each disk $i$
4: Split each disk into a number of smaller, equal-size chunks:
5: Calculate $max\_chunks$ as the LCM of the relative frequencies
6: Split each disk $i$ into $num\_chunk(i) = max\_chunks/rel\_freq(i)$ chunks
7: let $C_{ij}$ be the $j^{th}$ chunk in disk $i$
8: Create the broadcast program by interleaving the chunks of each disk:
9: for $i := 0$ to $max\_chunks - 1$ do
10:    for $j := 0$ to $num\_disks$ do
11:       broadcast chunk $C_{j,(i \bmod num\_chunks(j))}$
12:    end for
13: end for
```

Fig. 2.1 illustrates an example in which seven data items are divided into three groups of similar access probabilities and assigned to three separate disks in the broadcast. These three disks are interleaved in a single broadcast cycle. The first disk rotates at a speed twice as fast as the second one and four times as fast as the slowest disk (the third disk). The resulting broadcast cycle consists of four minor cycles.

We can observe that the *Bdisk* method can be used to construct a fine-grained memory hierarchy such that items of higher popularities are broadcast more frequently by varying the number of the disks, the size, relative spinning speed, and assigned data items of each disk.

2.1.4. Optimal Push Scheduling. Optimal broadcast schedules have been studied in [15, 33, 35, 36]. [15] discovered a *square-root rule* for minimizing access latency (a similar rule was proposed in a previous work [36], which considered fixed-size data items only). The rule states that the minimum overall expected access latency is achieved when the following two conditions are met:

1. instances of each data item are equally spaced on the broadcast channel;
2. the spacing s_i of two consecutive instances of each item i is proportional to the square-root of its length l_i and inversely proportional to the square-root of its access probability q_i, i.e.,

$$s_i \propto \sqrt{l_i/q_i} \tag{2.2}$$

or

$$s_i^2 \frac{q_i}{l_i} = constant. \tag{2.3}$$

Since these two conditions are not always simultaneously achievable, the online scheduling algorithm can only approximate the theoretical results. An efficient heuristic scheme was introduced in [35]. This scheme maintains two variables, B_i and C_i, for each item i. B_i is the earliest time when the next instance of item i should begin transmission, and $C_i = B_i + s_i$. C_i could be interpreted as the "suggested worse-case completion time" for the next transmission of item i. Let N be the number of items in the database and T be the current time. The heuristic online scheduling algorithm is given below.

Algorithm 2 Heuristic Algorithm for Optimal Push Scheduling.

```
1: Calculate optimal spacing $s_i$ for each item $i$ using Equation (2.2)
2: Initialize $T = 0, B_i = 0$, and $C_i = s_i$, $i = 1, 2, \ldots, N$
3: while the system is not terminated do
4:    Determine a set of item $S = \{i | B_i \leq T, 1 \leq i \leq N\}$
5:    Select to broadcast the item $i_{min}$ with the min $C_i$ value in $S$ (break ties arbitrarily)
6:    $B_{i_{min}} = C_{i_{min}}$
7:    $C_{i_{min}} = B_{i_{min}} + s_{i_{min}}$
8:    Wait for the completion of transmission for item $i_{min}$
9:    $T = T + l_{i_{min}}$
10: end while
```

This algorithm has a complexity of $O(logN)$ for each scheduling decision. Simulation results show that this algorithm performs close to the analytical lower bounds [35].

In [15], a low-overhead bucket-based scheduling algorithm based on the *square-root rule* was also provided. In this strategy, the database is partitioned into several buckets which are kept as cyclical queues. The algorithm chooses to broadcast the

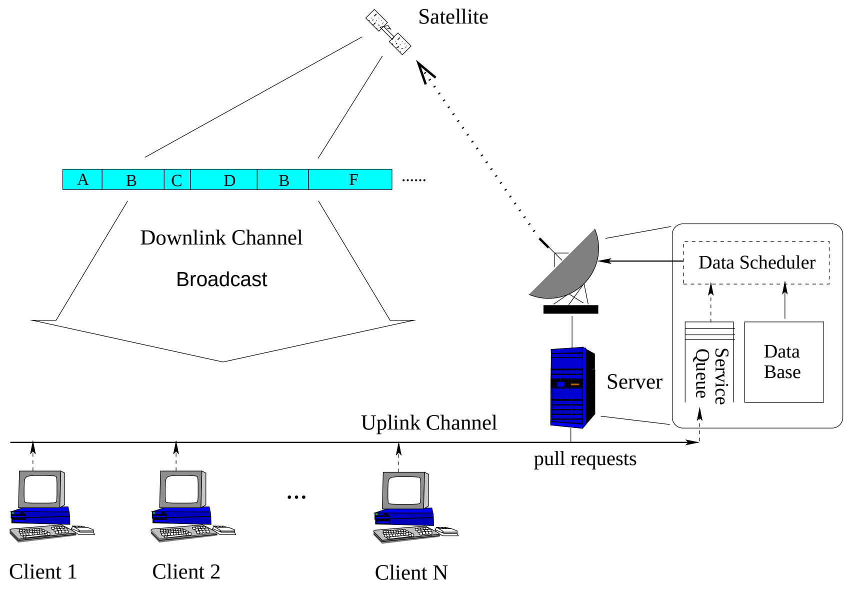

FIG. 2.2. *Architecture of On-Demand Broadcast*

first item in the bucket for which the expression $(T - R(I_m))^2 q_m / l_m$ evaluates to the largest value. In the expression, T is the current time, $R(i)$ is the time at which an instance of item i was most recently transmitted, I_m is the first item in bucket m, and q_m and l_m are average values of q_i's and l_i's for the items in bucket m. The bucket-based scheduling algorithm is similar to the *Bdisk* approach, but in contrast to the *Bdisk* approach, which has a fixed broadcast schedule, the bucket-based algorithm schedules the items online. As a result, they differ in the following aspects. First, a broadcast program generated using the *Bdisk* approach is periodic, whereas the bucket-based algorithm cannot guarantee that. Second, in the bucket-based algorithm, every broadcast instance is filled up with some data based on the scheduling decision, whereas the *Bdisk* approach may create "holes" in its broadcast program. Finally, the broadcast frequency for each disk is chosen manually in the *Bdisk* approach, while the broadcast frequency for each item is obtained analytically to achieve the optimal overall system performance in the bucket-based algorithm. Regrettably, no study has been carried out to compare their performance.

In a separate study [32], the broadcast system was formulated as a deterministic Markov Decision Process (*MDP*). [32] proposed a class of algorithms *Priority Index Policies With Length* (*PIPWL-γ*) which broadcast the item with the largest $(p_i/l_i)^\gamma (T - R(i))$, where the parameters are defined as above. In the simulation experiments, *PIPWL-0.5* showed a better performance than other settings did.

2.2. On-Demand Data Scheduling. As can be seen, push-based wireless data broadcasts are not tailored to a particular user's needs but rather satisfy the needs of the majority. Further, push-based broadcasts are not scalable to a large database size and react slowly to workload changes. To alleviate these problems, many recent research studies on wireless data dissemination have proposed using on-demand data broadcast (e.g., [5, 6, 16, 33]).

A wireless on-demand broadcast system supports both broadcast and on-demand services through a broadcast channel and a low-bandwidth uplink channel (see Fig. 2.2

for an example). The uplink channel can be a wired or a wireless link. When a client needs a data item, it sends to the server an on-demand request for the item through the uplink. Client requests are queued up (if necessary) at the server upon arrival. The server repeatedly chooses an item from among the outstanding requests, broadcasts it over the broadcast channel, and removes the associated request(s) from the queue. The clients monitor the broadcast channel and retrieve the item(s) they require.

The data scheduling algorithm in on-demand broadcast determines which request to service from its queue of waiting requests at every broadcast instance. In the following, on-demand scheduling techniques for fixed-size items and variable-size items, and energy-efficient on-demand scheduling are described.

2.2.1. On-Demand Scheduling for Equal-Size Items. Early studies on on-demand scheduling considered only equal-size data items. The average access time performance was used as the optimization objective. In [13] (also described in [36]), three scheduling algorithms, namely *MRF*, *MRFL*, and *LWF*, were proposed and compared to the *FCFS* algorithm:

- **First-Come-First-Served (FCFS):** Data items are broadcast in the order of their requests. This scheme is simple, but it has a poor average access performance for skewed data requests.
- **Most Requests First (MRF):** The data item with the largest number of pending requests is broadcast first; ties are broken in an arbitrary manner.
- **MRF Low (MRFL):** *MRFL* is essentially the same as *MRF*, but it breaks ties in favor of the item with the lowest request probability.
- **Longest Wait First (LWF):** The data item with the largest total waiting time, i.e., the sum of the time that all pending requests for the item have been waiting, is chosen for broadcast.

Numerical results presented in [13] yield the following observations: When the load is light, the average access time is insensitive to the scheduling algorithm used. This is expected because few scheduling decisions are required in this case. As the load increases, *MRF* yields the best access time performance when request probabilities on the items are equal. When request probabilities follow the *Zipf* distribution [45], *LWF* has the best performance and *MRFL* is close to *LWF*. However, *LWF* is not a practical algorithm for a large system. This is because at each scheduling decision, it needs to recalculate the total accumulated waiting time for every item with pending requests in order to decide which one to broadcast. Thus, *MRFL* was suggested as a low-overhead replacement of *LWF* in [13].

However, it was observed in [6] that *MRFL* has a performance as poor as *MRF* for a large database system. This is because, for large databases, the opportunity for tie-breaking diminishes and thus *MRFL* degenerates to *MRF*. Consequently, [6] proposed a low-overhead and scalable approach called *RxW*. The *RxW* algorithm schedules for the next broadcast the item with the maximal $R \times W$ value, where R is the number of outstanding requests for that item and W is the amount of time that the oldest of those requests has been waiting for. Thus, *RxW* broadcasts an item either because it is very popular or because there is at least one request that has waited for a long time. The method could be implemented inexpensively by maintaining the outstanding requests in two sorted orders, one ordered by R values and the other ordered by W values. In order to avoid exhaustive search of the service queue, a pruning technique was proposed to find the maximal $R \times W$ value. Simulation results show that the performance of *RxW* is close to *LWF*, meaning that it is a good alternative for *LWF* when scheduling complexity is a major concern.

To further improve scheduling overheads, a parameterized algorithm was developed based on *RxW* [6]. The parameterized *RxW* algorithm selects the first item it encounters in the searching process whose $R \times W$ value is greater than or equal to $\alpha \times$ *threshold*, where α is a system parameter and *threshold* is the running average of the $R \times W$ values of the requests that have been serviced. Varying the α parameter can adjust the performance trade-off between access time and scheduling overhead. For example, in the extreme case where $\alpha = 0$, this scheme selects the top item either in the R list or in the W list; this has the least scheduling complexity, but its access time performance may not be very good. With larger α values, the access time performance can be improved, but the scheduling complexity is increased as well.

2.2.2. On-Demand Scheduling for Variable-Size Items. On-demand scheduling for applications with variable data item sizes was studied in [5]. To evaluate the performance for items of different sizes, a new performance metric called *stretch* was used:

- **Stretch:** the ratio of the access time of a request to its service time, where the service time is the time needed to complete the request if it were the only job in the system.

Compared with access time, stretch is believed to be a more reasonable metric for items of variable sizes since it takes into consideration the size (i.e., service time) of a requested data item. Based on the stretch metric, four different algorithms have been investigated [5]. All of the four algorithms considered are preemptive in the sense that the scheduling decision is re-evaluated after broadcasting *any page* of a data item (it is assumed that a data item consists of one or more pages that have a fixed size and are broadcast together in a single data transmission).

- **Preemptive Longest Wait First (PLWF):** This is the preemptive version of the *LWF* algorithm. The *LWF* criterion is applied to select the subsequent data item to be broadcast.
- **Shortest Remaining Time First (SRTF):** The data item with the shortest remaining time is selected.
- **Longest Total Stretch First (LTSF):** The data item which has the largest total *current stretch* is chosen for broadcast. Here, the current stretch of a pending request is the ratio of the time the request has been in the system thus far to its service time.
- **MAX algorithm:** A deadline is assigned to each arriving request, and it schedules for the next broadcast the item with the earliest deadline. In computing the deadline for a request, the following formula is used:

$$deadline = arrival\_time + service\_time \times S_{MAX}, \tag{2.4}$$

  where S_{MAX} is the maximum stretch value of the individual requests for the last satisfied requests in a history window. To reduce computational complexity, once a deadline is set for a request, this value does not change even if S_{MAX} is updated before the request is serviced.

The trace-based performance study carried out in [5] indicates that none of these schemes is superior to the others in all cases. Their performance really depends on the system settings. Overall, the *MAX* scheme, with a simple implementation, performs quite well in both the worst and average cases in access time and stretch measures.

2.2.3. Energy-Efficient Scheduling. Datta et al. [12] took into consideration the energy saving issue in on-demand broadcasts. The proposed algorithms broadcast the requested data items in batches, using an existing indexing technique [21] to index the data items in the current broadcast cycle. This way, a mobile client may tune into a small portion of the broadcast instead of monitoring the broadcast channel until the desired data arrives. Thus, the proposed method is energy efficient. The data scheduling is based on a priority formula:

$$Priority = IF^{ASP} \times PF, \tag{2.5}$$

where IF (Ignore Factor) denotes the number of times that the particular item has not been included in a broadcast cycle, PF (Popularity Factor) is the number of requests for this item, and ASP (Adaptive Scaling Factor) is a factor that weights the significance of IF and PF. Two sets of broadcast protocols, namely Constant Broadcast Size (*CBS*) and Variable Broadcast Size (*VBS*), were investigated in [12]. The *CBS* strategy broadcasts data items in decreasing order of the priority values until the fixed broadcast size is exhausted. The *VBS* strategy broadcasts all data items with positive priority values. Simulation results show that the *VBS* protocol outperforms the *CBS* protocol at light loads, while at heavy loads the *CBS* protocol predominates.

2.3. Hybrid Data Scheduling. Push-based data broadcast cannot adapt well to a large database and a dynamic environment. On-demand data broadcast can overcome these problems. However, it has two main disadvantages: i) more uplink messages are issued by mobile clients, thereby adding demand on the scarce uplink bandwidth and consuming more battery power on mobile clients; ii) if the uplink channel is congested, the access latency will become extremely high. A promising approach, called hybrid broadcast, is to combine push-based and on-demand techniques so that they can complement each other. In the design of a hybrid system, three issues need to be considered:

1. access method from a client's point of view, i.e., where to obtain the requested data and how;
2. bandwidth/channel allocation between the push-based and on-demand deliveries;
3. assignment of a data item to either push-based broadcast, on-demand broadcast or both.

Concerning these three issues, there are different proposals for hybrid broadcast in the literature. In the following, we introduce the techniques for balancing push and pull and adaptive hybrid broadcast.

2.3.1. Balancing Push and Pull. A hybrid architecture was first investigated in [36, 37]. The model is shown in Fig. 2.3. In that model, items are classified as either frequently requested (f-request) or infrequently requested (i-request). It is assumed that clients know which items are f-requests and which are i-requests. The model services f-requests using a broadcast cycle and i-requests on demand. In the downlink scheduling, the server makes K consecutive transmissions of f-requested items (according to a broadcast program), followed by the transmission of the first item in the i-request queue (if at least one such request is waiting). Analytical results for the average access time were derived in [37].

In [4], the push-based *Bdisk* model was extended to integrate with a pull-based approach. The proposed hybrid solution, called *Interleaved Push and Pull* (*IPP*), consists of an uplink for clients to send to the server pull requests for the items that are not on the push-based broadcast. The server interleaves the *Bdisk* broadcast with

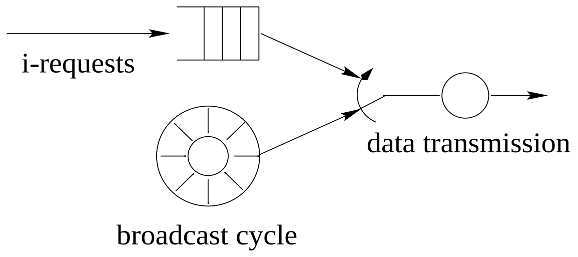

FIG. 2.3. *Architecture of Hybrid Broadcast*

the responses to pull requests on the broadcast channel. To improve the scalability of *IPP*, three different techniques were proposed:

1. Adjust the assignment of bandwidth to push and pull. This introduces a trade-off between how fast the push-based delivery is executed and how fast the queue of pull requests is served.
2. Provide a pull threshold T. Before a request is sent to the server, the client first monitors the broadcast channel for T time. If the requested data does not appear in the broadcast channel, the client sends a pull request to the server. This technique avoids overloading the pull service because a client will only pull an item that would otherwise have a very high push latency.
3. Successively chop off the pushed items from the slowest part of the broadcast schedule. This has the effect of increasing the available bandwidth for pulls. The disadvantage of this approach is that if there is not enough bandwidth for pulls, the performance might degrade severely since the pull latencies for non-broadcast items will be extremely high.

2.3.2. Adaptive Hybrid Broadcast. Adaptive broadcast strategies were studied for dynamic systems [25, 31]. These studies are based on the hybrid model in which the most frequently accessed items are delivered to clients based on flat broadcast, while the least frequently accessed items are provided point-to-point on a separate channel. In [31], a technique that continuously adjusts the broadcast content to match the hot-spot of the database was proposed. To do this, each item is associated with a *temperature* that corresponds to its request rate. Thus, each item can be in one of three possible states, namely *vapor*, *liquid*, and *frigid*. Vapor data items are those heavily requested and currently broadcast; liquid data items are those having recently received a moderate number of requests which is still not large enough for immediate broadcast; frigid data items refer to the cold items. The access frequency, and hence the state, of a data item can be dynamically estimated from the number of on-demand requests received through the uplink channel. For example, liquid data can be heated to vapor data if more requests are received. Simulation results show that this technique adapts very well to rapidly changing workloads.

Another adaptive broadcast scheme was discussed in [25], which assumes fixed channel allocation for data broadcast and point-to-point communication. The idea for adaptive broadcast is to maximize (but not overload) the use of available point-to-point channels so that a better overall system performance can be achieved.

3. Time-Critical On-Demand Broadcast. In many situations, user requests are associated with time constraints. Consider a driver who queries a traffic information server to select one of several alternative routes at some point ahead [14]. Clearly, it is necessary for the server to provide the driver with the traffic information (e.g., which route is less congested) *before* he reaches that point; otherwise, the information is of no value to the driver. In this case, it is necessary for users to specify for each request a *deadline* beyond which she is no longer interested (or less interested) in the requested information. On-demand broadcast with time constraints is referred to as *time-critical on-demand broadcast* [43]. This section introduces a newly developed scheduling algorithm for time-critical on-demand broadcast.

We start by illustrating the factors affecting the performance of time-critical broadcast scheduling. The broadcast duration of an item is referred to as a *broadcast tick*. We compare EDF (earliest deadline first) and MRF (most requests first), two typical scheduling algorithms in unicast and broadcast respectively. At each broadcast tick, EDF broadcasts the item with the shortest remaining lifetime to cater for the *urgency* of requests. MRF, on the other hand, broadcasts the item that has the largest number of pending requests to account for the *productivity* of broadcasting. As shall be shown in Section 3.2, EDF and MRF respectively achieve good performance for certain workloads only. This suggests an integration of the *urgency* and *productivity* factors to improve scheduling performance. Intuitively,

- For two items with the same number of pending requests, the one with closer deadline should be broadcast first.
- For two items with the same deadline, the one with more pending requests should be broadcast first.

Motivated by the above observations, a new scheduling algorithm, called SIN-α (*S*lack time *I*nverse *N*umber of pending requests), has been proposed [43]. Specifically, the $sin.\alpha$ value of each item that has at least one pending request is given by

$$sin.\alpha = \frac{slack}{num^{\alpha}} = \frac{1stDeadline - clock}{num^{\alpha}},$$

where *slack* is the duration from the current time (i.e., *clock*) to the deadline of the most urgent pending request for the item (i.e., $1stDeadline$), *num* is the number of pending requests for the item, and $\alpha \geq 0$ is a relative weight of productivity to urgency. At each broadcast tick, the item with the minimum $sin.\alpha$ value is broadcast on the downlink channel. It is easy to see that the larger the value of α, the more influential the number of pending requests.

3.1. An Efficient Implementation. A straightforward implementation of SIN-α is to compute, at each broadcast tick, the $sin.\alpha$ values of all items that have pending requests and to broadcast the one with the minimum $sin.\alpha$ value. Such an implementation has a scheduling complexity of at least $\mathcal{O}(m)$, where m is the number of items with pending requests. A more efficient implementation of SIN-α is presented in the section.

The pending requests in the service queue are grouped by the requested items. Two data structures, an *S-list* and an *N-list*, are used to index the requested items in the service queue. Each item has one entry in the S-list and N-list respectively. As shown in Figure 3.1, the S-list is a bidirectional linked list where the items are sorted in ascending order of the associated earliest deadline (i.e., in ascending order of *slack*). In the N-list, the items having the same number of pending requests are first structured into a min-heap built on the key of the earliest deadline. The roots of the heaps are then organized into a bidirectional linked list in descending order of

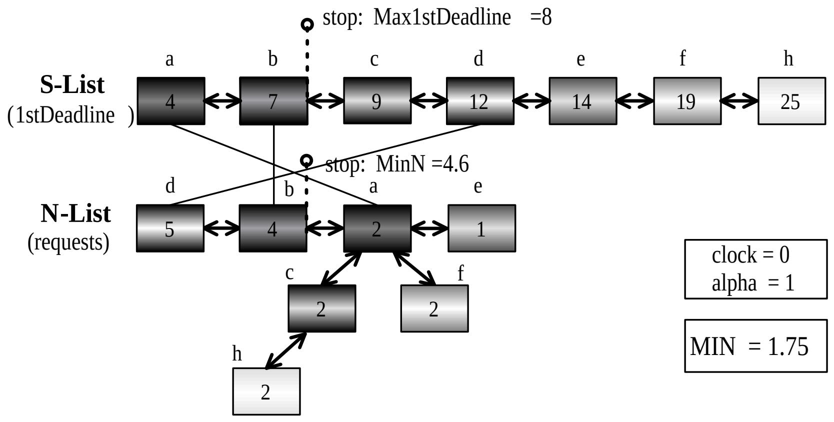

FIG. 3.1. *Indexing Structures of the Service Queue*

the number of pending requests (i.e., *num*). The heap that indexes the items with n pending requests is referred to as *heap-n*.

At each broadcast tick, the server broadcasts the item with the minimum $sin.\alpha$ value. The proposed data structures reduce the search space of candidate items in two aspects. First, since the requested items in each min-heap of the N-list have the same number of requests and the min-heap is constructed based on the key of the earliest deadline, the root item of each heap has the minimum $sin.\alpha$ value among all the items in the heap. Thus, non-root items in the N-list can be excluded from the search space. Moreover, the search space can be further pruned by searching the S-list and N-list in an alternate fashion. Starting from the heads of the two lists, we sequentially examine the entries therein. Two values, $MinN$ and $Max1stDeadline$, are maintained to cut off the search space in the N-list and S-list respectively. Let MIN denote the minimum $sin.\alpha$ value found so far. Since the S-list is sorted in ascending order of *slack*, an unexamined item has a $sin.\alpha$ value less than MIN only if its *num* value exceeds

$$MinN = \left(\frac{NextS}{MIN}\right)^{\frac{1}{\alpha}},$$

where $NextS$ is the *slack* value of the next item in the S-list. Similarly, since the N-list is sorted in descending order of *num*, an unexamined item has a smaller $sin.\alpha$ value than MIN only if its *slack* value is less than

$$MaxS = (NextN)^{\alpha} \cdot MIN,$$

i.e., the corresponding $1stDeadline$ value is less than

$$Max1stDeadline = (NextN)^{\alpha} \cdot MIN + clock,$$

where $NextN$ is the *num* value of the next item in the N-list. MIN, $MinN$, and $Max1stDeadline$ are updated after examining each item. The search process continues until the list tails are reached or the next items in the lists violate the necessary conditions indicated by $MinN$ and $Max1stDeadline$. The pseudo-code of the scheduling algorithm is presented in Algorithm 3, where *pn* and *ps* point to the next items in the N-list and S-list respectively.

Algorithm 3 Efficient Search Algorithm for SIN-α.

```
1: $MIN := \infty$
2: $pn$ := the head of the N-list
3: $ps$ := the head of the S-list
4: while ($pn \neq nil$ or $ps \neq nil$) do
5:   if $pn \neq nil$ then
6:     calculate $sin.\alpha_{pn}$, the $sin.\alpha$ value of the item pointed by $pn$
7:     if $sin.\alpha_{pn} < MIN$ then $MIN := sin.\alpha_{pn}$
8:     if $ps$ and $pn$ refer the same item then advance $ps$ to the next unexamined
       item in the S-list whose entry in the N-list is a heap root
9:     advance $pn$ to the next unexamined item in the N-list
10:    if $ps \neq nil$ then $MinN := (\frac{ps \rightarrow 1stDeadline - clock}{MIN})^{\frac{1}{\alpha}}$
11:    if $pn \neq nil$ and $pn \rightarrow num < MinN$ then $pn$ := nil
12:    if $pn \neq nil$ then $Max1stDeadline := (pn \rightarrow num)^{\alpha} \cdot MIN + clock$
13:    if $ps \neq nil$ and $ps \rightarrow 1stDeadline > Max1stDeadline$ then $ps$ := nil
14:  end if
15:  if $ps \neq nil$ then
16:    calculate $sin.\alpha_{ps}$, the $sin.\alpha$ value of the item pointed by $ps$
17:    if $sin.\alpha_{ps} < MIN$ then $MIN := sin.\alpha_{ps}$
18:    if $ps$ and $pn$ refer the same item then advance $pn$ to the next unexamined
       item in the N-list
19:    advance $ps$ to the next unexamined item in the S-list whose entry in the N-list
       is a heap root
20:    repeat lines 10-13
21:  end if
22: end while
```

Figure 3.1 shows an example. Suppose that the current clock is 0 and α is set to 1. First, we examine the first item d in the N-list and get $MIN = 2.4$, $MinN = 1.7$, and $Max1stDeadline = 9.6$. Then, we go ahead to examine the first item a in the S-list and obtain a smaller $sin.\alpha$ value 2.0 for a. Therefore, we update $MIN = 2.0$, $MinN = 3.5$, and $Max1stDeadline = 8.0$. Next, we go to the second item b in the N-list, whose $sin.\alpha$ value is 1.75, and we update $MIN = 1.75$ and $MinN = 8.0$ ($Max1stDeadline$ remains at 8.0 since pn becomes nil). The searching ceases here since the remaining items do not have a $1stDeadline$ value greater than $Max1stDeadline$ and an N value less than $MinN$ and, hence, cannot have a $sin.\alpha$ value less than MIN. In total, we only need to examine three item entries to find the item b to broadcast.

When a new request arrives, the request is inserted into the service queue and the corresponding request group is updated. If the request group is empty, a new item entry is created for the requested item and two index entries are inserted into the S-list and *heap-1* of the N-list respectively. Otherwise, the earliest deadline of the requested item is updated if necessary, and the S-list is adjusted accordingly. Moreover, the entry of the requested item in the N-list is moved from *heap-x* to *heap-*$(x+1)$, if there were x pending requests for the item before the new request arrival. After selecting an item to broadcast, the scheduler removes from the service queue the requests for the item as well as those whose lifetimes will expire in the next broadcast tick.

Description	**Default**	**Range**
Number of Pages	4923	-
Request Rate Scaling Factor	1	[0.25-128]
Service Rate (pages/second)	10	-
Overall Mean Relative Deadline (seconds)	60	-

TABLE 3.1
Workload Parameters and Settings

3.2. Performance Evaluation. A trace-driven simulator has been developed to evaluate the performance of the SIN algorithm. Real trace collected from the World Cup '98 website [49] (i.e., the day-38 trace) was used to simulate the requests made by clients. The day-38 trace contains over 7 million requests for 4923 distinct web pages.[1] The average request rate is 83 per second. The access counts of different pages sorted in descending order are shown in Figure 3.2. It can be seen that the access pattern follows a Zipf-like distribution, which is consistent with the observation made in the literature [10]. To simulate different levels of workloads, the time scale of the trace was changed by introducing a *request rate scaling factor f*. The inter-arrival time between two consecutive requests was set to the actual time logged in the trace divided by f. It is obvious that the higher the value of f, the heavier the workload. The service rate (i.e., capacity) of the broadcast channel is described in the number of pages that can be transmitted per second. The default service rate was set at 10 pages/sec. To model the time constraints of requests, each request was assigned a relative deadline d randomly generated based on a specified distribution (i.e., exponential, uniform, or fixed distributions) with the assigned mean value of the requested page. In the default setting, we differentiate the time requirements for different pages and assume the mean relative deadlines of requests for different pages are uniformly distributed between 0 and 120 seconds. Under this setting, the overall mean relative deadline of requests is 60 seconds. The workload parameters used in the experiments are summarized in Table 3.1. Each simulation run started with an empty service queue. The first 1,000,000 requests were considered the start-up period, and performance statistics were collected for the subsequent 2,000,000 requests.

Recall that in SIN-α, $\alpha \geq 0$ is a factor rating the relative importance of productivity and urgency for candidate data items (i.e., web pages). It was observed that an α value between 1 and 4 gave the best overall performance, but no α value consistently dominates the other values. The performance difference between the α values from 1 to 4 is not very significant. Therefore, in the rest of this section, SIN-1 is used as the representative of SIN-α. Note that the SIN-α algorithm would obtain even better performance if the value of α can be fine tuned for specific workloads.

SIN-1 is compared with the existing algorithms EDF, MRF, and the recently proposed *RxW* [6]. With *RxW*, the server broadcasts the page that has either a large number of pending requests or a long waiting time. The objective of *RxW* is to reduce the response time of requests. Figures 3.3 through 3.5 shows the request drop rates as a function of the scaling factor f.

Comparing different scheduling algorithms, the SIN-1 algorithm performs the best throughout the tested range of scaling factor. The improvement of SIN-1 relative to EDF is up to 13.1%, 15.3%, and 38.0% for exponential, uniform, and fixed deadline

[1]Interested readers are referred to [7] for more details of the WorldCup98 web server traces.

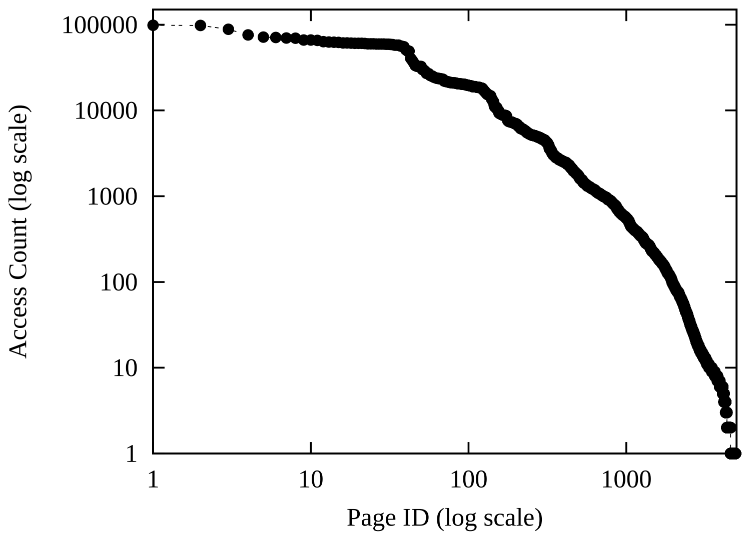

FIG. 3.2. *Page Access Counts for the Day-38 Trace (4923 Pages)*

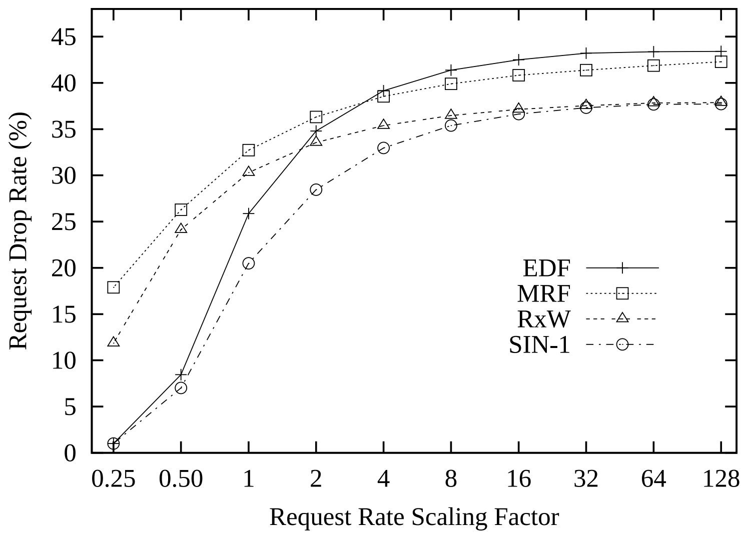

FIG. 3.3. *Request Drop Rates for Exponentially Distributed Deadlines*

distributions respectively and the improvement relative to MRF is at least 10.8%, 16.8%, and 49.8% respectively. Since MRF and *RxW* ignore the request deadlines, their drop rates are high even when the system is lightly loaded (see the left parts of Figures 3.3 through 3.5). In contrast, no requests are dropped by SIN-1 and EDF at low system loads. When the system is heavily loaded (see the right parts of Figures 3.3 through 3.5), SIN-1 performs substantially better than both MRF and EDF.

It is interesting to note that, among the three deadline distributions, the relative performance of MRF and *RxW* against EDF and SIN-1 is the worst when the relative deadline is fixed. This can be explained as follows. If the relative deadline spans

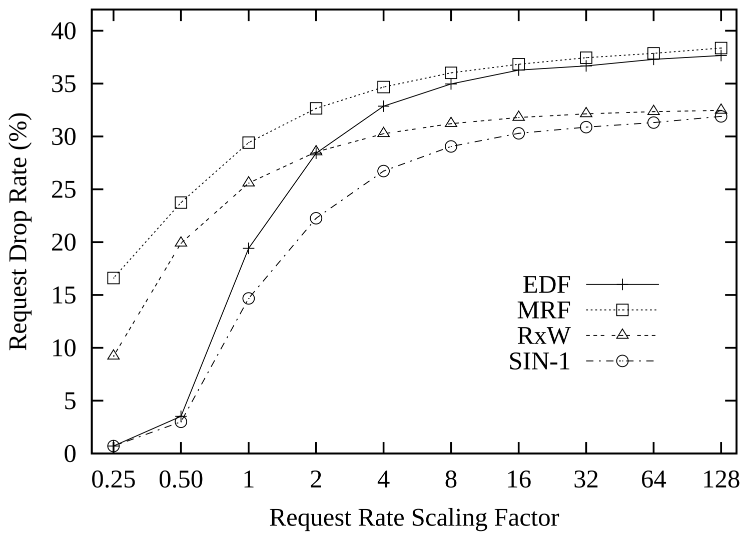

FIG. 3.4. *Request Drop Rates for Uniform Deadlines*

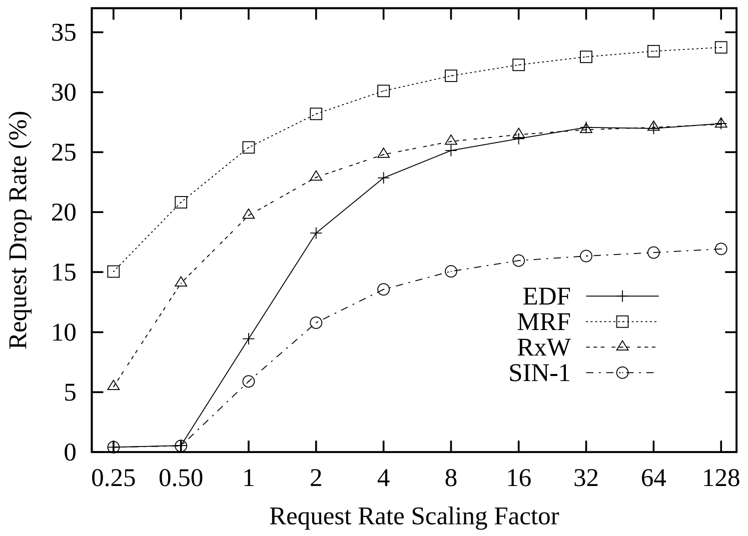

FIG. 3.5. *Request Drop Rates for Fixed Deadlines*

over a range, a newly arrived request has a chance of overwriting the slack time of the requested item if there exist pending requests for the item already. Therefore, MRF and *RxW*, to some extent, take the urgency factor into consideration by first broadcasting the item with the largest number of pending requests and/or the longest waiting time. However, a new request never overwrites the slack time of the requested item under fixed deadline distribution. In this case, MRF and *RxW* completely ignore the urgency factor and performs much worse than SIN-1.

4. Other Related Issues.

4.1. Fault-tolerant Broadcast. Wireless transmission is error-prone. Data might be corrupted or lost due to many factors like signal interference, etc. When errors occur, mobile clients have to wait for the next copy of the data if no special precaution is taken. This will increase both access time and tune-in time. To deal with unreliable wireless communication, the basic idea is to introduce controlled redundancy in the broadcast program. Such redundancy allows mobile clients to obtain their data items from the current broadcast cycle even in the presence of errors. This eliminates the need to wait for the next broadcast of the data whenever any error occurs. Studies on fault-tolerant broadcast disks have been performed in [9] and [34].

4.2. Data Scheduling over Multiple Broadcast Channels. It is argued in [29] that multiple physical channels cannot be coalesced into a single high-bandwidth channel. Hence, recent studies have been working on data scheduling over multiple broadcast channels [26, 27, 29]. In [27], to minimize the average access delay for data items in the broadcast program, a heuristic algorithm VF^k was developed to allocate data over a number of channels. While previous studies addressed data scheduling and indexing separately, [26, 29] considered the scheduling problem of both data and index over multiple channels. Various server broadcast and client access protocols were investigated in [29]. In [26], the allocation problem aimed at minimizing both the average access time and the average tune-in time. It was mapped into the *personnel assignment problem* from which the optimization techniques were derived to solve the problem.

4.3. Handling Updates for Data Broadcast. In reality, many applications that can best profit from a broadcast-based approach are required to update their data frequently over time (e.g., stock quotation systems and traffic reports). Data consistency issues for transactional operations in push-based broadcast were explored in [28, 30]. For a wireless broadcast environment, the correctness criteria of *ACID* transactions might be too restrictive. Thus, these studies relaxed some of the requirements and new algorithms have been developed. In [28], the correctness criterion for read-only transactions is that each transaction reads consistent data, i.e., the read set of each read-only transaction must form a subset of a consistent database state. The proposed schemes maintain multiple versions of items either on air or in a client cache to increase the concurrency of client read-only transactions. In [30], the correctness criterion employed is *update consistency*, which ensures 1) the *mutual consistency* of data maintained by the server and read by clients; and 2) the *currency* of data read by clients. Two practical schemes, *F-Matrix* and *R-Matrix*, were proposed to efficiently detect update consistent histories by broadcasting some control information along with data.

4.4. Air Indexing. Energy conservation is a scarce resource on mobile clients, which ranges from only a few hours to about half a day under continuous use. To retrieve a data item in wireless data broadcast, if without any index information, a client has to continuously monitor the broadcast until the data arrives. This will consume a lot of energy since the client has to remain *active* during its waiting time. A solution to this problem is *air indexing*. The basic idea is to include index information about the arrival times of data items on the broadcast channel. By accessing the index, mobile clients are able to predict the arrivals of their desired data. Thus, they can stay in *power saving mode* during waiting time and tune into the broadcast channel only when the data items of their interests arrive. Several traditional disk-based indexing techniques such as B$^+$-tree have been extended for air indexing [11, 17, 18, 21, 22,

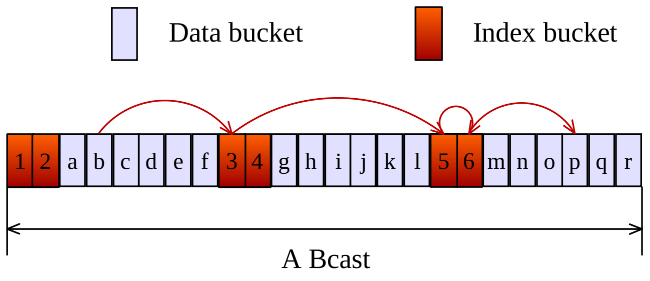

FIG. 4.1. *Data Organization on Wireless Broadcast Channels*

24]. A new exponential index has also been proposed in [39]. To facilitate search of data items via air index, each data bucket includes an offset to the beginning of the next index bucket. Taking Figure 4.1 as an example, the general access protocol for retrieving data involves the following steps:

- Initial probe: The client tunes into the broadcast channel at bucket b and determines when the next index bucket is broadcast. The client goes to the power saving mode.
- Index search: The client tunes into the broadcast channel again at index bucket 3 and selectively accesses a number of index buckets (i.e., index buckets 3, 5, and 6) to find out when to get the desired data held in bucket p. Again, it goes to the power saving mode.
- Data retrieval: When bucket p arrives, the client downloads it and retrieves the desired data.

4.5. Client Cache Management. An important issue relating to wireless data dissemination is client data caching. Client data caching is a common technique for improving access latency and data availability [41]. In the framework of a mobile wireless environment, this is much more desirable due to constraints such as limited bandwidth and frequent disconnections. However, frequent client disconnections and movements between different cells make the design of cache management strategies a challenge. [2] discussed methods for keeping clients' caches consistent with the updated data values at the server for the *Bdisk* systems. The techniques of invalidating or updating cached copies were investigated. The issues of cache consistency, cache replacement, and cache prefetching have also been investigated in [3, 8, 38, 42, 44].

5. Summary. This chapter has presented various broadcast scheduling techniques for wireless data dissemination. We surveyed push-based, on-demand, and hybrid scheduling algorithms. Push-based broadcast is attractive when access patterns are known *a priori*, while on-demand broadcast is desirable for dynamic access patterns. Hybrid data broadcast offers more flexibility by combining push-based and on-demand broadcasts. In addition, we have introduced a newly developed scheduling algorithm called SIN for time-critical on-demand broadcast. Finally, some other issues of wireless data dissemination, such as fault-tolerant broadcast, updates handling, air indexing, and client cache management, were briefly reviewed.

REFERENCES

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for

asymmetric communications environments. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 199–210, San Jose, CA, USA, May 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik. Disseminating updates on broadcast disks. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 354–365, Mumbai (Bombay), India, September 1996.

[3] S. Acharya, M. Franklin, and S. Zdonik. Prefetching from a broadcast disk. In *Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, pages 276–285, New Orleans, LA, USA, February 1996.

[4] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 183–194, Tucson, AZ, USA, May 1997.

[5] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 43–54, Dallas, TX, USA, October 1998.

[6] D. Aksoy and M. Franklin. R x W: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Transactions on Networking*, 7(6):846–860, December 1999.

[7] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May/June 2000.

[8] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching strategies for mobile environments. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 1–12, Minneapolis, MN, USA, May 1994.

[9] S. K. Baruah and A. Bestavros. Pinwheel scheduling for fault-tolerant broadcast disks in real-time database systems. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 543–551, Birmingham, UK, April 1997.

[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. IEEE INFOCOM'99*, pages 126–134, March 1999.

[11] M.-S. Chen, K.-L. Wu, and P. S. Yu. Optimizing index allocation for sequential data broadcasting in wireless mobile computing. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 15(1):161–173, Jan./Feb. 2003.

[12] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast protocols to support efficient retrieval from databases by mobile users. *ACM Transactions on Database Systems (TODS)*, 24(1):1–79, March 1999.

[13] H. D. Dykeman, M. Ammar, and J. W. Wong. Scheduling algorithms for videotex systems under broadcast delivery. In *Proceedings of IEEE International Conference on Communications (ICC'86)*, pages 1847–1851, Toronto, Canada, June 1986.

[14] J. Fernandez and K. Ramamritham. Adaptive dissemination of data in time-critical asymmetric communication environments. In *Proc. Euromicro Real-Time Systems Symp.*, pages 195–203, 1999.

[15] S. Hameed and N. H. Vaidya. Efficient algorithms for scheduling data broadcast. *ACM/Baltzer Journal of Wireless Networks (WINET)*, 5(3):183–193, 1999.

[16] Q. L. Hu, D. L. Lee, and W.-C. Lee. Performance evaluation of a wireless hierarchical data dissemination system. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 163–173, Seattle, WA, USA, August 1999.

[17] Q. L. Hu, W.-C. Lee, and D. L. Lee. A hybrid index technique for power efficient data broadcast. *Journal of Distributed and Parallel Databases (DPDB)*, 9(2):151–177, March 2000.

[18] Q. L. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'2000)*, pages 157–166, San Diego, CA, USA, February 2000.

[19] V. Liberatore. Multicast scheduling for list requests. In *Proc. IEEE INFOCOM'02*, New York, NY, June 2002.

[20] T. Imielinski and S. Viswanathan. Adaptive wireless information systems. In *Proceedings of the Special Interest Group in DataBase Systems (SIGDBS) Conference*, pages 19-41, Tokyo, Japan, October 1994.

[21] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air - organization and access. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(3):353–372, May-June 1997.

[22] K. C. K. Lee, H. V. Leong, and A. Si. A semantic broadcast scheme for a mobile environment based on dynamic chunking. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS'2000)*, pages 522–529, Taipei, Taiwan, April 2000.

[23] W.-C. Lee, Q. L. Hu, and D. L. Lee. A study of channel allocation methods for data dissemi-

nation in mobile computing environments. *ACM/Baltzer Journal of Mobile Networks and Applications (MONET)*, 4(2):117–129, 1999.

[24] W.-C. Lee and D. L. Lee. Using signature techniques for information filtering in wireless and mobile environments. *Journal of Distributed and Parallel Databases (DPDB)*, 4(3):205–227, July 1996.

[25] C. W. Lin and D. L. Lee. Adaptive data delivery in wireless communication environments. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS'2000)*, pages 444–452, Taipei, Taiwan, April 2000.

[26] S.-C. Lo and A. L. P. Chen. Optimal index and data allocation in multiple broadcast channels. In *Proceedings of the 16th IEEE International Conference on Data Engineering (ICDE'2000)*, pages 293–302, San Diego, CA, USA, February 2000.

[27] W.-C. Peng and M.-S. Chen. Dynamic generation of data broadcasting programs for a broadcast disk array in a mobile computing environment. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM'2000)*, pages 38–45, McLean, VA, USA, November 2000.

[28] E. Pitoura and P. K. Chrysanthis. Exploiting versions for handling updates in broadcast disks. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 114–125, Edinburgh, Scotland, UK, September 1999.

[29] K. Prabhakara, K. A. Hua, and J. Oh. Multi-level multi-channel air cache designs for broadcasting in a mobile environment. In *Proceedings of the 16th IEEE International Conference on Data Engineering (ICDE'2000)*, pages 167–176, San Diego, CA, USA, February 2000.

[30] J. Shanmugasundaram, A. Nithrakashyap, R. M. Sivasankaran, and K. Ramamritham. Efficient concurrency control for broadcast environments. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 85–96, Philadelphia, PA, USA, June 1999.

[31] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 326–335, Athens, Greece, August 1997.

[32] C. J. Su and L. Tassiulas. Broadcast scheduling for the distribution of information items with unequal length. In *Proceedings of the 31st Conference on Information Science and Systems (CISS'97)*, March 1997.

[33] C. J. Su, L. Tassiulas, and V. J. Tsotras. Broadcast scheduling for information distribution. *ACM/Baltzer Journal of Wireless Networks (WINET)*, 5(2):137–147, 1999.

[34] K. L. Tan and J. X. Yu. On selective tuning in unreliable wireless channels. *Journal of Data and Knowledge Engineering (DKE)*, 28(2):209–231, November 1998.

[35] N. H. Vaidya and S. Hameed. Scheduling data broadcast in asymmetric communication environments. *ACM/Baltzer Journal of Wireless Networks (WINET)*, 5(3):171–182, 1999.

[36] J. W. Wong. Broadcast delivery. *Proceedings of the IEEE*, 76(12):1566–1577, December 1988.

[37] J. W. Wong and H. D. Dykeman. Architecture and performance of large scale information delivery networks. In *Proceedings of the 12th International Teletraffic Congress*, pages 440–446, Torino, Italy, June 1988.

[38] J. Xu, Q. L. Hu, W.-C. Lee, and D. L. Lee. Performance evaluation of an optimal cache replacement policy for wireless data dissemination. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(1): 125-139, Jan. 2004.

[39] J. Xu, W.-C. Lee, and X. Tang. Exponential index: A parameterized distributed indexing scheme for data on air. In *Proceedings of the 2nd ACM SIGMOBILE/USENIX Int. Conf. on Mobile Systems, Applications, and Services (MobiSys'04)*, Boston, MA, June 2004.

[40] J. Xu, B. Li, and D. Lee. On bandwidth allocation for data dissemination in cellular mobile networks. *ACM/Kluwer Journal of Wireless Networks (WINET)*, 9(2): 103-116, March 2003.

[41] J. Xu, J. Liu, B. Li, and X. Jia. Caching and Prefetching for Web Content Distribution. *IEEE Computing in Science and Engineering (CiSE), Special Issue on Web Engineering*, to appear, 2004.

[42] J. Xu, X. Tang, and D. L. Lee. Performance analysis of location-dependent cache invalidation schemes for mobile environments. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(2): 474-488, March/April 2003.

[43] J. Xu, X. Tang, and W.-C. Lee. Time-critical on-demand broadcast: Algorithms, analysis, and performance evaluation. Technical Report COMP-03-015, Department of Computer Science, Hong Kong Baptist Univ., June 2003. Available at http://www.comp.hkbu.edu.hk/tech-report/.

[44] J. Xu, Y. Zhu, J. Xu, B. Li, and L. Ni. A cooperative caching algorithm for multi-cell data access. In *Proceedings of IEEE International Conference on Communications (ICC '04)*,

Paris, France, June 2004.

[45] G. K. Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, MA, USA, 1949.

[46] DirectBand Network. Microsoft Smart Personal Objects Technology (SPOT). [Online]. Available: http://www.microsoft.com/resources/spot/.

[47] StarBand. [Online]. Available: http://www.starband.com/.

[48] Hughes Network Systems. DIRECWAY homepage. [Online]. Available: http://www.direcway.com/.

[49] WorldCup98 web site access logs, 1998. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/WorldCup.html.