11 Data Broadcast

JIANLIANG XU and DIK-LUN LEE

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{xujl,dlee}@cs.ust.hk

QINGLONG HU

IBM Silicon Valley Lab
San Jose, CA, USA
qhu@us.ibm.com

WANG-CHIEN LEE

Verizon Laboratories
Waltham, MA, USA
wang-chien.lee@verizon.com

11.1 INTRODUCTION

We have been witnessing in the past few years the rapid growth of wireless data applications in the commercial market thanks to the advent of wireless devices, wireless high-speed networks, and supporting software technologies. We envisage that in the near future, a large number of mobile users carrying portable devices (e.g., palmtops, laptops, PDAs, WAP phones, etc.) will be able to access a variety of information from anywhere and at any time. The types of information that may become accessible wirelessly are boundless, and include news, stock quotes, airline schedules, and weather and traffic information, to name but a few.

There are two fundamental information delivery methods for wireless data applications: point-to-point access and broadcast. In point-to-point access, a logical channel is established between the client and the server. Queries are submitted to the server and results returned to the client in much the same way as in a wired network. In broadcast, data are sent simultaneously to all users residing in the broadcast area. It is up to the client to select the data it wants. Later we will see that in a special kind of broadcast system, namely on-demand broadcast, the client can also submit queries to the server so that the data it wants are guaranteed to be broadcast.

Compared with point-to-point access, broadcast is a more attractive method for several reasons:

- A single broadcast of a data item can satisfy all the outstanding requests for that item simultaneously. As such, broadcast can scale up to an arbitrary number of users.
- 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.
- A wireless communication system essentially employs a broadcast component to deliver information. Thus, data broadcast can be implemented without introducing any additional cost.

While point-to-point and broadcast systems share many concerns, such as the need to improve response time while conserving power and bandwidth consumption, this chapter focuses on broadcast systems only.

Access efficiency and *power conservation* are two critical issues in any wireless data system. Access efficiency concerns how fast a request is satisfied, while power conservation concerns how to reduce a mobile client's power consumption when it is accessing the data it wants. The second issue is important because of the limited battery power on mobile clients, which ranges from only a few hours to about half a day under continuous use. Moreover, only a modest improvement in battery capacity of 20-30% can be expected over the next few years [30]. In the literature, two basic performance metrics, namely *access time* and *tune-in time*, are used to measure access efficiency and power conservation for a broadcast system, respectively:

- Access Time: The time elapsed between the moment when a query is issued and the moment when it is satisfied.
- Tune-in Time: The time a mobile client stays active to receive the requested data items.

Obviously, broadcasting irrelevant data items increases client access time, and hence, deteriorates the efficiency of a broadcast system. A broadcast *schedule*, which determines what is to be broadcast by the server and when, should be carefully designed. 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, 12], 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, 16, 21], push-based broadcast and on-demand data deliveries are combined to complement each other. Consequently, there are three kinds of data scheduling methods (i.e., *push-based scheduling*, *on-demand scheduling*, and *hybrid scheduling*) corresponding to these three data broadcast models.

In data broadcast, to retrieve a data item a mobile client has to continuously monitor the broadcast until the data item of interest arrives. This will consume a lot of battery power since the client has to remain active during its waiting time. A solution to this problem is *air indexing*. The basic idea is that by including auxiliary information about the arrival times of data items on the broadcast channel, mobile clients are able to predict the arrivals of their desired data. Thus, they can stay in the power saving mode and tune into the broadcast channel only when the data items of interest to them arrive. The drawback of this solution is that broadcast cycles are lengthened due to additional indexing information. As such, there is a trade-off between access time and tune-in time. Several indexing techniques for wireless data broadcast have been introduced to conserve battery power while maintaining short access latency. Among these techniques, index tree [18] and signature [22] are two representative methods for indexing broadcast channels.

The rest of this chapter is organized as follows. Various data scheduling techniques are discussed for push-based, on-demand, and hybrid broadcast models in Section 11.2. In Section 11.3, air indexing techniques are introduced for single-attribute and multi-attribute queries. Section 11.4 discusses some other issues of wireless data broadcast, such as semantic broadcast, fault-tolerant broadcast, and updates handling. Finally, this chapter is summarized in Section 11.5.

11.2 DATA SCHEDULING

11.2.1 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*.

11.2.1.1 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 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.

11.2.1.2 Probabilistic-based Broadcast To improve performance for skewed data access, the probabilistic-based broadcast [38] selects an item i for inclusion in the broadcast program with probability f_i, where f_i is determined by the access proba-

bilities of the items. The best setting for f_i is given by the following formula [38]:

$$f_i = \frac{\sqrt{q_i}}{\sum_{j=1}^{N} \sqrt{q_j}}, \tag{11.1}$$

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 [38].

11.2.1.3 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.

Broadcast Disks Generation Algorithm {
  Order the items in decreasing order of access popularities;
  Allocate items in the same range of access probabilities on a different *disk*;
  Choose the relative broadcast frequency $rel\_freq(i)$ (in integer) for each disk i;
  Split each disk into a number of smaller, equal-size chunks:
    Calculate max_chunks as the LCM of the relative frequencies;
    Split each disk i into $num\_chunk(i) = max\_chunks/rel\_freq(i)$ chunks;
    let C_{ij} be the j^{th} chunk in disk i;
  Create the broadcast program by interleaving the chunks of each disk:
    for $i := 0$ to $max\_chunks - 1$
    {
      for $j := 0$ to num_disks
        broadcast chunk $C_{j,(i \bmod num\_chunks(j))}$;
    }
}

Fig. 11.1 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.

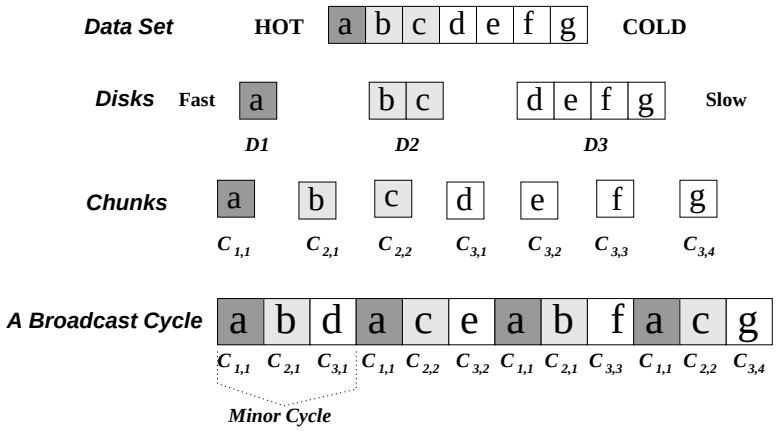


Fig. 11.1 An Example of a Seven-item, Three-disk Broadcast Program

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 the assigned data items of each disk.

11.2.1.4 Optimal Push Scheduling Optimal broadcast schedules have been studied in [12, 34, 37, 38]. [12] discovered a *square-root rule* for minimizing access latency (note that a similar rule was proposed in a previous work [38], 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.,

$$s_i \propto \sqrt{l_i/q_i} \tag{11.2}$$

or

$$s_i^2 \frac{q_i}{l_i} = constant. \tag{11.3}$$

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 [37]. 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.

Heuristic Algorithm for Optimal Push Scheduling {
Calculate optimal spacing s_i for each item i using Equation (11.2);
Initialize $T = 0, B_i = 0$, and $C_i = s_i, i = 1, 2, \ldots, N$;
While (the system is not terminated){
Determine a set of item $S = \{i | B_i \leq T, 1 \leq i \leq N\}$;
Select to broadcast the item i_{min} with the min C_i value in S (break ties arbitrarily);
$B_{i_{min}} = C_{i_{min}}$;
$C_{i_{min}} = B_{i_{min}} + s_{i_{min}}$;
Wait for the completion of transmission for item i_{min};
$T = T + l_{i_{min}}$;
}
}

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 [37].

In [12], 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 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. Note that the expression $(T - R(I_m))^2 q_m / l_m$ is similar to Equation (11.3). 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 [33], the broadcast system was formulated as a deterministic Markov Decision Process (*MDP*). [33] 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.

11.2.2 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, 13, 34]).

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

11.2.2.1 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 [11] (also described in [38]), 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 [11] 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 [42], *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 [11].

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 the *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*. 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.

11.2.2.2 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:

$$deadline = arrival\_time + service\_time \times S_{MAX}, \tag{11.4}$$

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.

11.2.2.3 Energy-efficient Scheduling Datta et al. [10] 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 [18] (refer to Section 11.3 for details) 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:

$$Priority = IF^{ASP} \times PF, \tag{11.5}$$

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 [10]. 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.

11.2.3 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.

11.2.3.1 Balancing Push and Pull A hybrid architecture was first investigated in [38, 39]. The model is shown in Fig. 11.2. 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 [39].

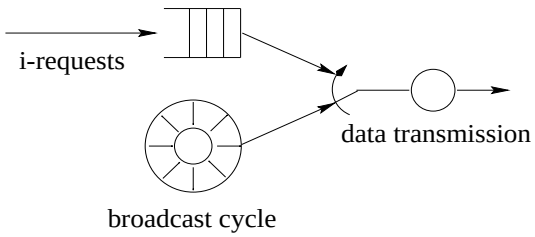


Fig. 11.2 Architecture of Hybrid Broadcast

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 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.

11.2.3.2 Adaptive Hybrid Broadcast Adaptive broadcast strategies were studied for dynamic systems [24, 32]. 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 [32], 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 [24], 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.

11.3 AIR INDEXING

11.3.1 Power Conserving Indexing

Power conservation is a key issue for battery-powered mobile computers. Air indexing techniques can be employed to predict the arrival time of a requested data item so that a client can slip into *doze mode* and switch back to *active mode* only when the data of interest arrives, thus substantially reducing battery consumption.

In the following, various indexing techniques will be described. The general access protocol for retrieving indexed *data frame*s involves the following steps:

- Initial probe: The client tunes into the broadcast channel and determines when the next index is broadcast.

- Search: The client accesses the index to find out when to tune into the broadcast channel to get the required frames.
- Retrieve: The client downloads all the requested information frames.

When no index is used, a broadcast cycle consists of data frames only (called *non-index*). As such, the length of the broadcast cycle and hence the access time are minimum. However, in this case, since every arriving frame must be checked against the condition specified in the query, the tune-in time is very long and is equal to the access time.

11.3.1.1 The Hashing Technique As mentioned previously, there is a trade-off between the access time and the tune-in time. Thus, we need different data organization methods to accommodate different applications. The *hashing-based scheme* and the *flexible indexing method* were proposed in [17].

In the hashing-based scheme, instead of broadcasting a separate directory frame with each broadcast cycle, each frame carries the control information together with the data that it holds. The control information guides a search to the frame containing the desired data in order to improve the tune-in time. It consists of a hash function and a shift function. The hash function hashes a key attribute to the address of the frame holding the desired data. In the case of collision, the shift function is used to compute the address of the overflow area, which consists of a sequential set of frames starting at a position behind the frame address generated by the hash function.

The flexible indexing method first sorts the data items in ascending (or descending) order and then divides them into p segments numbered 1 through p. The first frame in each of the data segments contains a control index, which is a binary index mapping a given key value to the frame containing that key. In this way, we can reduce the tune-in time. The parameter p makes the indexing method flexible since, depending on its value, we can either get a very good tune-in time or a very good access time.

In selecting between the hashing-based scheme and the flexible indexing method, the former should be used when the tune-in time requirement is not rigid and the key size is relatively large compared to the record size. Otherwise, the latter should be used.

11.3.1.2 The Index Tree Technique As with a traditional disk-based environment, the index tree technique [18] has been applied to data broadcasts on wireless channels. Instead of storing the locations of disk records, an index tree stores the arrival times of information frames.

Fig. 11.3 depicts an example of an index tree for a broadcast cycle which consists of 81 information frames. The lowest level consists of square boxes which represent a collection of three information frames. Each index node has three pointers (for simplicity, the three pointers pointing out from each leave node of the index tree are represented by just one arrow).

To reduce tune-in time while maintaining a good access time for clients, the index tree can be replicated and interleaved with the information frames. In *distributed indexing*, the index tree is divided into replicated and non-replicated parts. The

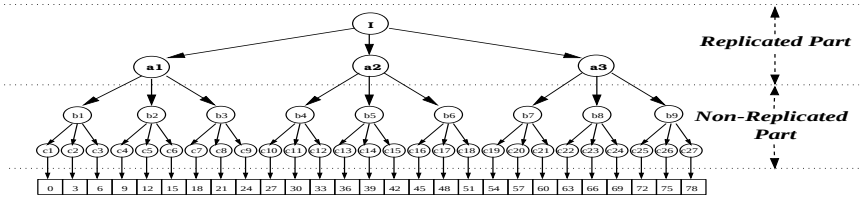


Fig. 11.3 A Full Index Tree

replicated part consists of the upper levels of the index tree, while the non-replicated part consists of the lower levels. The index tree is broadcast every $\frac{1}{d}$ of a broadcast cycle. However, instead of replicating the entire index tree d times, each broadcast only consists of the replicated part and the non-replicated part that indexes the data frames immediately following it. As such, each node in the non-replicated part appears only once in a broadcast cycle. Since the lower levels of an index tree take up much more space than the upper part (i.e., the replicated part of the index tree), the index overheads can be greatly reduced if the lower levels of the index tree are not replicated. In this way, tune-in time can be improved significantly without causing much deterioration in access time.

To support distributed indexing, every frame has an *offset* to the beginning of the root of the next index tree. The first node of each distributed index tree contains a tuple, with the first field containing the primary key of the data frame that is broadcast last and the second field containing the offset to the beginning of the next broadcast cycle. This is to guide the clients that have missed the required data in the current cycle to tune to the next broadcast cycle. There is a *control index* at the beginning of every replicated index to direct clients to a proper branch in the index tree. This additional index information for navigation together with the sparse index tree provides the same function as the complete index tree.

11.3.1.3 The Signature Technique The signature technique has been widely used for information retrieval. A signature of an information frame is basically a bit vector generated by first hashing the values in the information frame into bit strings and then superimposing one on top of another [22]. Signatures are broadcast together with the information frames. A query signature is generated in a similar way based on the query specified by the user. To answer a query, a mobile client can simply retrieve information signatures from the broadcast channel and then match the signatures with the query signature by performing a bitwise *AND* operation. If the result is not the same as the query signature, the corresponding information frame can be ignored. Otherwise, the information frame is further checked against the query. This step is to eliminate records with different values but that have the same signature due to the superimposition process.

The signature technique interleaves signatures with their corresponding information frames. By checking a signature, a mobile client can decide whether an information frame contains the desired information. If it does not, the client turns

into doze mode and wakes up again for the next signature. The primary issue with different signature methods is the size and the number of levels of the signatures to be used.

In [22], three signature algorithms, namely *simple signature*, *integrated signature*, and *multi-level signature*, were proposed and their cost models for access time and tune-in time were given. For simple signatures, the signature frame is broadcast before the corresponding information frame. Therefore, the number of signatures is equal to the number of information frames in a broadcast cycle. An integrated signature is constructed for a group of consecutive frames, called a *frame group*. The multi-level signature is a combination of the simple signature and the integrated signature methods, in which the upper level signatures are integrated signatures and the lowest level signatures are simple signatures.

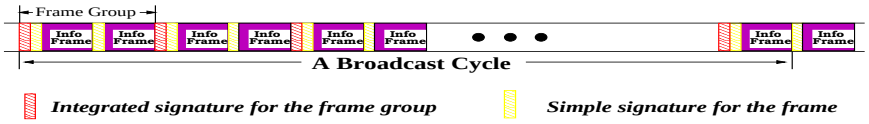


Fig. 11.4 The Multi-level Signature Technique

Fig. 11.4 illustrates a two-level signature scheme. The black signatures in the figure are integrated signatures. An integrated signature indexes all data frames between itself and the next integrated signature (i.e., two data frames). The white signatures are simple signatures for the corresponding data frames. In the case of non-clustered data frames, the number of data frames indexed by an integrated signature is usually small in order to maintain the filtering capability of the integrated signatures. On the other hand, if similar data frames are grouped together, the number of frames indexed by an integrated signature can be large.

11.3.1.4 The Hybrid Index Approach Both the signature and the index tree techniques have some advantages and disadvantages. For example, the index tree method is good for random data access, while the signature method is good for sequentially structured media such as broadcast channels. The index tree technique is very efficient for a clustered broadcast cycle, and the signature method is not affected much by the clustering factor. While the signature method is particularly good for multi-attribute retrieval, the index tree provides a more accurate and complete global view of the data frames. Since clients can quickly search the index tree to find out the arrival time of the desired data, the tune-in time is normally very short for the index tree method. However, a signature does not contain global information about the data frames; thus it can only help clients to make a quick decision regarding whether the current frame (or a group of frames) is relevant to the query or not. For the signature method, the filtering efficiency depends heavily on the false drop probability of the signatures. As a result, the tune-in time is normally long and is proportional to the length of a broadcast cycle.

A new index method, called the *hybrid index*, builds index information on top of the signatures and a *sparse* index tree to provide a global view for the data frames

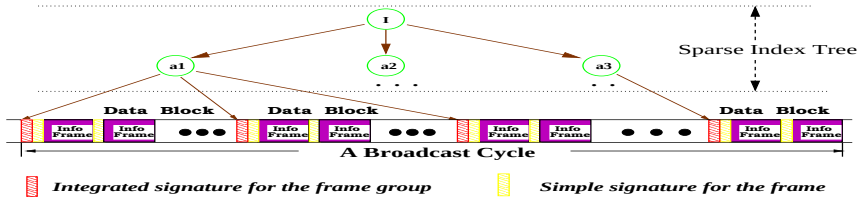


Fig. 11.5 The Hybrid Index Technique

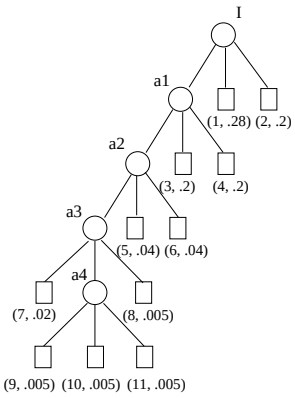
and their corresponding signatures. The index tree is called sparse because only the upper t levels of the index tree (the replicated part in the distributed indexing) are constructed. A *key-search pointer* node in the t-th level points to a *data block*, which is a group of consecutive frames following their corresponding signatures. Since the size of the upper t levels of an index tree is usually small, the overheads for such additional indexes are very small. Fig. 11.5 illustrates a hybrid index. To retrieve a data frame, a mobile client first searches the sparse index tree to obtain the approximate location information about the desired data frame and then tunes into the broadcast to find out the desired frame.

Since the hybrid index technique is built on top of the signature method, it retains all of the advantages of a signature method. Meanwhile, the global information provided by the sparse index tree considerably improves tune-in time.

11.3.1.5 The Unbalanced Index Tree Technique To achieve better performance with skewed queries, the unbalanced index tree technique was investigated [9, 31]. Unbalanced indexing minimizes the average index search cost by reducing the number of index searches for hot data at the expense of spending more on cold data.

For fixed index fanouts, a *Huffman-based* algorithm can be used to construct an optimal unbalanced index tree. Let N be the number of total data items and d the fanout of the index tree. The Huffman-based algorithm first creates a forest of N subtrees, each of which is a single node labeled with the corresponding access frequency. Then, the d subtrees with the smallest labels are attached to a new node, and the resulting subtree is labeled with the sum of all the labels from its d child subtrees. This procedure is repeated until there is only one subtree. Fig. 11.6 demonstrates an index tree with a fixed fanout of three. In the figure, each data item i is given in the form of (i, q_i), where q_i is the access probability for item i.

Given the data access patterns, an optimal unbalanced index tree with a fixed fanout is easy to construct. However, its performance may not be optimal. Thus, [9] discussed a more sophisticated case for variable fanouts. In this case, the problem of optimally constructing an index tree is NP-hard [9]. In [9], a greedy algorithm *Variant Fanout (VF)* was proposed. Basically, the *VF* scheme builds the index tree in a top-down manner. *VF* starts by attaching all data items to the root node. Then, it groups the nodes with small access probabilities and moves them to one level lower so as to minimize the average index search cost. Fig. 11.7 shows an index tree built using the *VF* method, where the access probability for each data is the same as in



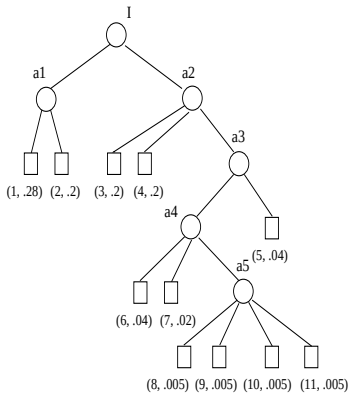


Fig. 11.6 Index Tree of a Fixed Fanout of 3 **Fig. 11.7** Index Tree of Variable Fanouts

the example for fixed fanouts. The index tree with variable fanouts in Fig. 11.7 has a better average index search performance than the index tree with fixed fanouts in Fig. 11.6 [9].

11.3.2 Multi-attribute Air Indexing

So far, the index techniques considered are based on one attribute and can only handle single-attribute queries. In real world applications, data frames usually contain multiple attributes. Multi-attribute queries are desirable because they can provide more precise information to the users.

Since broadcast channels are a linear medium, when compared to single-attribute indexing and querying, data management and query protocols for multiple attributes appear much more complicated. *Data clustering* is an important technique used in single-attribute air indexing. It places data items with the same value under a specific attribute consecutively in a broadcast cycle [14, 17, 18]. Once the first data item with the desired attribute value arrives, all data items with the same attribute value can be successively retrieved from the broadcast. For multi-attribute indexing, a broadcast cycle is clustered based on the most frequently accessed attribute. Although the other attributes are non-clustered in the cycle, a second attribute can be chosen to cluster the data items within a data cluster of the first attribute. Likewise, a third attribute can be chosen to cluster the data items within a data cluster of the second attribute. We call the first attribute the *clustered attribute* and the other attributes the *non-clustered attributes*.

For each non-clustered attribute, a broadcast cycle can be partitioned into a number of segments called *meta-segments* [18], each of which holds a sequence of frames with non-decreasing (or non-increasing) values of that attribute. Thus, when we look at each individual meta-segment, the data frames are clustered on that attribute and the indexing techniques discussed in the last subsection can still be applied to a meta-segment. The number of meta-segments in the broadcast cycle for an attribute

is called the *scattering factor* of the attribute. The scattering factor of an attribute increases as the importance of the attribute decreases.

The index tree, signature, and hybrid methods are applicable to indexing multi-attribute data frames [15]. For multi-attribute indexing, an index tree is built for each index attribute, while multiple attribute values are superimposed to generate signatures.

When two special types of queries, i.e., queries with all conjunction operators and queries with all disjunction operators, are considered, empirical comparisons show that the index tree method, while performing well for single-attribute queries results in poor access time performance [15]. This is due to its large overheads for building a distributed index tree for each attribute indexed. Moreover, the index tree method has an update constraint, i.e., updates of a data frame are not reflected until the next broadcast cycle. The comparisons revealed that the hybrid technique is the best choice for multi-attribute queries due to its good access time and tune-in time. The signature method performs close to the hybrid method for disjunction queries. The index tree method has a similar tune-in time performance as the hybrid method for conjunction queries, whereas it is poor in terms of access time for any type of multi-attribute queries.

11.4 OTHER ISSUES

11.4.1 Semantic Broadcast

The indexing techniques discussed in Section 11.3 can help mobile clients filter information and improve tune-in time for data broadcast. This type of broadcast is called *item-based*. One major limitation of such item-based schemes is their lack of semantics associated with a broadcast. Thus, it is hard for mobile clients to determine if their queries could be answered from the broadcast entirely, forcing them to contact the server for possibly additional items. To remedy this, a semantic-based broadcast approach was suggested [20]. This approach attaches a semantic description to each broadcast unit, called a *chunk*, which is a cluster of data items. This allows clients to determine if a query can be answered based solely on the broadcast and to define precisely the remaining items in the form of a "supplementary" query.

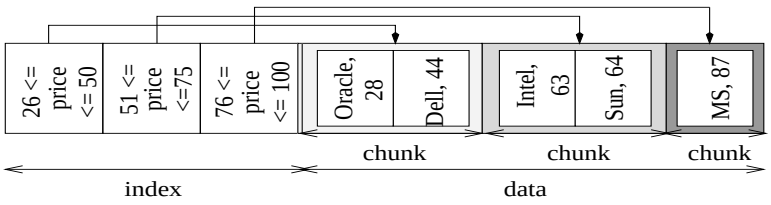


Fig. 11.8 An Example of Semantic Broadcast

Consider a stock information system containing stock pricing information. The server broadcasts some data, along with their indexes. A mobile client looking for an investment opportunity issues a query for the list of companies whose stock prices are between \$30 and \$70 (i.e., $30 \leq price \leq 70$). Employing a semantic-based broadcast scheme as shown in Fig. 11.8, data items are grouped into chunks, each of which is indexed by a semantic descriptor. A client locates the required data in the first two chunks since they together can satisfy the query predicate completely. For the first chunk, it drops the item (Oracle, 28) and keeps the item (Dell, 44). For the second chunk, both items (Intel, 63) and (Sun, 64) are retained. In case that the server decides not to broadcast the first chunk (i.e., the stocks whose prices are between \$26 and \$50), a client could assert that the missing data can be loaded from the server using a query with predicate $30 \leq price \leq 50$.

11.4.2 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 and air indexing have been performed in [8] and [36], respectively.

11.4.3 Data and Index Allocation over Multiple Broadcast Channels

It is argued in [28] that multiple physical channels cannot be coalesced into a single high-bandwidth channel. Hence, recent studies have been working on data and index allocation over multiple broadcast channels [25, 26, 28]. In [26], 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, [25, 28] considered the allocation problem of both data and index over multiple channels. Various server broadcast and client access protocols were investigated in [28]. In [25], 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.

11.4.4 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). [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.

Data consistency issues for transactional operations in push-based broadcast were explored in [27, 29]. 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 [27], 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 [29], 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.

11.4.5 Client Cache Management

An important issue relating to data broadcast is client data caching. Client data caching is a common technique for improving access latency and data availability. 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. The issues of cache consistency, cache replacement, and cache prefetching have been explored in [3, 7, 19, 40, 41].

11.5 SUMMARY

This chapter has presented various techniques for wireless data broadcast. Data scheduling and air indexing were investigated with respect to their performance in access efficiency and power consumption. For data scheduling, push-based, on-demand, and hybrid scheduling were discussed. 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. For air indexing, several basic indexing techniques, such as the hashing method, the index tree method, the signature method, and the hybrid method, were described. Air indexing techniques for multi-attribute queries were also discussed. Finally, some other issues of wireless data broadcast, such as semantic broadcast, fault-tolerant broadcast, updates handling, and client cache management, were briefly reviewed.

11.6 ACKNOWLEDGMENT

The writing of this chapter was supported by Research Grants Council of Hong Kong, China (Project numbers HKUST-6077/97E and HKUST-6241/00E).

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. 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.

8. 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.

9. M.-S. Chen, P. S. Yu, and K.-L. Wu. Indexed sequential data broadcasting in wireless mobile computing. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)*, pages 124–131, Baltimore, MD, USA, May 1997.

10. 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.

11. 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.

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

13. 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.

14. 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.

15. 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.

16. 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.

17. T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficient filtering of data on air. In *Proceedings of the 4th International Conference on Extending Database Technology (EDBT'94)*, pages 245–258, Cambridge, UK, March 1994.

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

19. J. Jing, A. K. Elmagarmid, A. Helal, and R. Alonso. Bit-sequences: A new cache invalidation method in mobile environments. *ACM/Baltzer Journal of Mobile Networks and Applications (MONET)*, 2(2):115–127, 1997.

20. 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.

21. W.-C. Lee, Q. L. Hu, and D. L. Lee. A study of channel allocation methods for data dissemination in mobile computing environments. *ACM/Baltzer Journal of Mobile Networks and Applications (MONET)*, 4(2):117–129, 1999.

22. 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.

23. W.-C. Lee and D. L. Lee. Signature caching techniques for information broadcast and filtering in mobile environments. *ACM/Baltzer Journal of Wireless Networks (WINET)*, 5(1):57–67, 1999.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. S. Sheng, A. Chandrasekaran, and R. W. Broderson. A portable multimedia terminal. *IEEE Communications Magazine*, 30(12):64–75, December 1992.

31. N. Shivakumar and S. Venkatasubramanian. Energy-efficient indexing for information dissemination in wireless systems. *ACM/Baltzer Journal of Mobile Networks and Applications (MONET)*, 1(4):433–446, 1996.

32. 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.

33. 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.

34. 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.

35. K. L. Tan and J. X. Yu. Energy efficient filtering of nonuniform broadcast. In *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS'96)*, pages 520–527, Hong Kong, May 1996.

36. 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.

37. 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.

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

39. 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.

40. J. Xu, Q. L. Hu, D. L. Lee, and W.-C. Lee. SAIU: An efficient cache replacement policy for wireless on-demand broadcasts. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM'2000)*, pages 46–53, McLean, VA, USA, November 2000.

41. J. Xu, X. Tang, D. L. Lee, and Q. L. Hu. Cache coherency in location-dependent information services for mobile environment. In *Proceedings of the 1st International Conference on Mobile Data Management*, pages 182–193, Hong Kong, December 1999.

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