Chapter 12
Data Dissemination and Broadcast Disks

12.1 Introduction

Data management is one of the important challenges in mobile applications. Due to resource constraints, mobile devices are not equipped to run any interesting, non-trivial stand-alone applications. On the other hand, the enabling technologies such as cellular communication, wireless LAN, wireless data network and satellite services have equipped mobile terminals with capabilities to access data/information anywhere at any time. Researchers saw immense potentials in combining two forementioned aspects in complementary roles, for developing a powerful framework for innovative person centric services. The key idea is to organize a mobile application as a set of synchronized activities that requires very little computation at a mobile end host, but capitalizes on globally available resources accessible through networks. This would allow a mobile user to transact both official business, and financial transactions. The possibilities of creating innovative commercial and financial models for anytime, anywhere access of person centric services excited research in mobile data management.

The core problem of data delivery over mobile wireless network became a major challenge to the database community since the adaptation of the concept of object oriented databases. In a wireless network, connectivity is not only flaky, but bandwidth is also low. Furthermore, there is no control on the number of mobile users which may appear in an area of cellular coverage. The scale is simply overwhelming. The conventional data delivery model based on request and response (like HTTP) cannot just match up to the challenge of scale.

The push based data delivery is considered as the most attractive option that may allow a server to avoid being flooded by large number of client requests. Some early work in the area of data dissemination inspired the research for data dissemination in mobile distributed system. In 1734, broadcast emerged as a dominant theme for high volume information dissemination by utilizing print media. Around 1898, radio transmission was introduced, and in 1924 video streaming became a possible with

R.K. Ghosh, *Wireless Networking and Mobile Data Management*,
DOI 10.1007/978-981-10-3941-6_12

invention of Television. Electronic transmission media not only served as entertainment channels but also revolutionized dissemination of public information.

Around mid eighties two systems were implemented using broadcast delivery. The first one is known as BCIS (Boston Community Information System) [8]. It was a pilot project for dissemination of news and information over FM channel to about 200 clients having personal computers equipped with wireless interfaces. The system used both push and pull based data delivery methods [3]. The second one called Datacycle [10], was a high throughput oriented transaction system implemented over public telephone system. It exploited transmission bandwidth of optical systems to leverage database as a storage pump, and broadcast records repeatedly on a broadcast channel. A transaction on a host could request an operation on database using a well defined interface provided with the associated access manager. The operation was decomposed into a specification to process records as they appear on the broadcast channel. To complete a transaction, a host submits update/commit requests to the access manager. Using a network uplink, the access manager sends the update request to the update manager, which executes non conflicting updates on the database.

Theoretical results concerning performance of broadcast delivery models were provided in [14]. Wong's result, in particular, states that the lower bound of the bandwidth requirement for the best possible mean response time of a broadcast page is proportional to square root of its access frequency.

In this chapter our focus is primarily restricted to push based data delivery. However, it deals with classification of various options for data delivery in a client server system along with the factors influencing these options. The major part of the chapter is devoted to the idea behind treating an inherently sequential medium such as a wireless communication channel organized in the form of logical disks on the air. Furthermore, it shows that by treating the medium of air as part of a logical memory hierarchy, data delivery mechanism can be organized extending from an individual mobile client's memory to a server's disk.

12.2 Data Access Issues in Mobile Environment

At an end user's level, the major concerns are:

1. Energy efficient accesses of requisite data,
2. Management of range of disconnections, and
3. Efficient processing of queries.

Therefore, indexing, caching and replications are as much relevant in mobile environment as they are over the wired network with stationary hosts. However, due to the unique physical characteristics of mobile terminals and wireless communication medium these issues needed a revisit. In summary, the techniques for data management in mobile distributed environment should be able to handle following four main issues.

1. Mobility
2. Scaling
3. Disconnection
4. Access/delivery modes.

Data related to mobility may far exceed the complexity of any conventional large database. It comprises of locating a mobile object, addressing, routing queries and delivering responses. Some of these issues have been discussed under location management in the previous chapter. The issues arising out of scaling and disconnection are mainly handled through replication and caching strategies. Replication and caching are discussed in Chap. 14. Access and delivery mechanisms are intertwined. The role of delivery mechanism does not mean just putting loads of bits over communication channels, but organizing these bits through a structure that can be effectively exploited to access data. But two most important constraints which come on the way are:

- Availability of limited bandwidth, and
- Availability of limited battery power.

These constraints severely restrict transfer of large volume data. In a cellular based network, the number of active mobile users cannot be controlled. If every mobile uses a back channel to pull data for its own application from a fixed nodes, then the channel saturates quickly and suffers from congestion too often. Availability of limited energy in a mobile device is also a possible cause of planned disconnections, as the user may wish to prolong battery life. So there is a need for energy efficient access methods.

12.3 Pull and Push Based Data Delivery

A client application is the consumer of data and the server is the producer. In other words, data and information are generated at the servers. So, the servers are repositories of data/information that the client applications would require from time to time. A server has to deliver appropriate data which the clients require for the applications to run. There are broadly two data delivery models as shown in Fig. 12.1. A client initiated data delivery is essentially *pull based*. The client sends an explicit request to a server. The responsibility of fetching data rests exclusively on the client. On the other hand, when a server takes the responsibility of transferring data, then transfer takes place in anticipation of a future access. No explicit request are made from any client. The server estimates the data requirements of the client population in a global context. A client must be alert during the time when the required data for its application flows into a broadcast channel from a server. The data delivery model adhering to this protocol referred to as `push-based` delivery. The delivery model is analogous to TV transmission where the viewers tune to specific channel if they wish to view a TV program. The channel caters to an expected general viewing

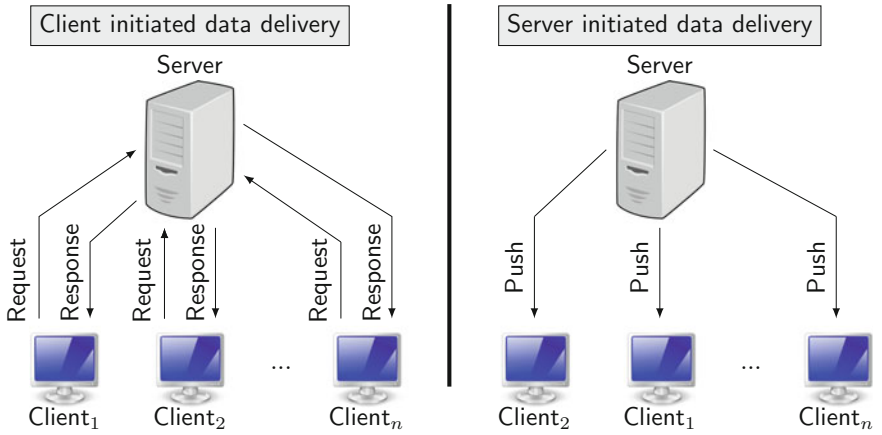

Fig. 12.1 Client and server initiated data delivery mechanisms

pattern on the basis of monthly ratings of the different TV shows. TV transmission, thus, may not necessarily cater to specific individual choices. A similar situation is witnessed in push-based data transfer. The information is not explicitly sought for, but broadcast by a server to all or a group of clients. This method of data transfer is referred to as *data dissemination*.

Figure 12.2 illustrates an implementation oriented view of data transfer mechanism as applicable to the two data delivery models. In the pull-based data transfer occurs in a *request-response cycle*. data transfer. In a push based transfer, data transfer occurs on a *broadcast channel*. The server must have an idea of data that should be pushed. Therefore, there may be a back channel through which profile informa-

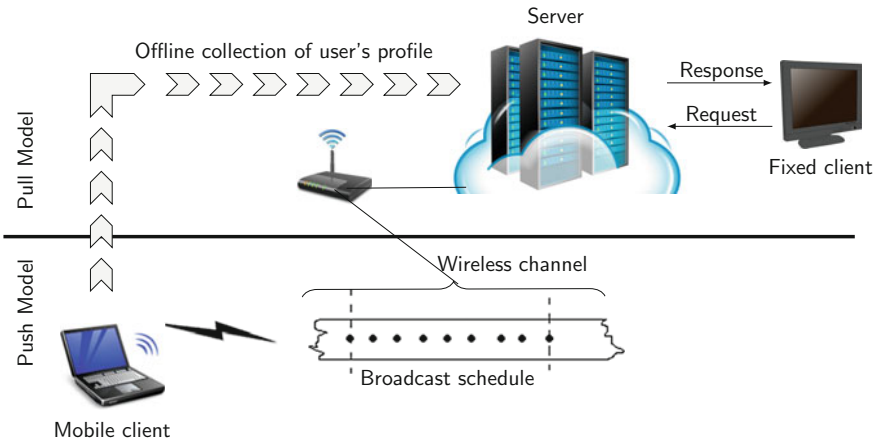

Fig. 12.2 Data delivery models

tion of clients can be collected as indicated in the Fig. 12.2. Otherwise, the data dissemination on broadcast channel can only be organized purely on the basis of a knowledgeable assessment of the data requirements of the clients.

12.4 Dissemination in Mobile Environment

In mobile environment, there is a built-in asymmetry as mobile devices are resource poor. Compute-intensive tasks are executed on the servers. Databases and other storage repositories are also located on the servers. So, data dissemination techniques in mobile environment are built around push based delivery mechanism, and rely on broadcast or point to multipoint transmissions.

Pure push and pure pull represent two extreme data delivery methods. Between the two extremities many delivery methods have been used. These methods were determined by the varying requirements of data from the applications. On the basis of design issues faced by applications, data delivery methods can be broadly classified into four categories.

1. Data oriented,
2. Mechanism oriented,
3. Organization oriented,
4. Bandwidth oriented.

In the data oriented design, a client's requirement of data can be of three types, namely,

1. Publish only,
2. Demand driven,
3. Hybrid.

Certain data objects generated by a server assist a client learn about the occurrences of events. The client does not require to process such data, but needs to know as and when such data objects get generated. A server's obligation is limited to publish such data objects on downlink channel as and when these get generated. A client accesses the desired data by passive listening and local filtering. Some applications may require the clients to explicitly fetch certain data from a server and process them. This requires a pull based delivery, wherein a clients places a demand for its data requirement explicitly in form of queries over its uplink channel. The server sends responses that resolve the queries. A hybrid delivery mechanism represents a combination mechanism, where some data requirements are met by data objects belonging to publish group and others are met by demand driven group of objects. Normally, in a hybrid model there will be only a few objects belonging to demand driven group.

Franklin and Zodnik [6, 7] observed application design spaces can be partitioned according to mechanisms of data delivery, namely,

1. Delivery initiation,
2. Scheduling strategy,
3. Communication type.

Delivery of data can either be initiated by a client or by a server. When initiated by a client it represents essentially a pull model. On the other hand, when delivery is initiated by a server, it represents push model. Scheduling basically deals with the temporal mechanism of data delivery. The data is delivered on some pre-determined schedule. Communication type can be: (i) unicast or point to point, (ii) multicast or point to many points, i.e., one-to-many (iii) broadcast or one-to-all. There is a difference between multicast and broadcast though both represent the communication type that transmits data from one source to many destinations. In multicast the number of recipients are known, whereas in broadcast the recipient can be any one. For multicast, communication, a list of recipients should be maintained.

The organization of data delivery refers to the way data organized for delivery to client. Two possible ways data can be organized, namely,

1. Broadcast program,
2. Selective tuning.

Broadcast program requires a server to anticipate the data requirements of its clients. On the basis of the requirements, the server organizes a push schedule for data on downlink channel. The data objects may be organized on a push schedule based on client access priorities or bandwidth minimization or combination of both. We plan to discuss more about broadcast program subsequently in this chapter as it represents an important and interesting data dissemination technique specially in the context of mobile distributed environments. Selective tuning requires a client to create its own filtering and caching strategies for accessing data from the downlink channel. The server embeds meta data and/or index along with information when pushing data on downlink channel. Air indexing schemes are discussed in the next chapter.

Bandwidth oriented data delivery mechanism is concerned with apportioning of bandwidth for pull and push. Normally, in a mobile environment much of the server data is pushed on a downlink channel. Only a small fraction of bandwidth is allocated for uplink communication. The uplink serves as a back channel for the demand driven data delivery. However, in theory, bandwidth partitioning can be done either (i) statically, or (ii) dynamically. In static allocation, the bandwidth is split in advance between downlink and uplink. But this type of allocation may lead to under utilization when access pattern is dynamic. Dynamic allocation is complex. It requires additional runtime information about the changes in access patterns.

12.5 Comparison of Pull and Push Models

In a wired network, all the computers share the same physical medium (wire) for data transfer. Therefore, data transmission capacities in both uplink and downlink directions are the same. In wireless network, there is an asymmetry in capacity of

Table 12.1 Asymmetry in link capacities of wireless networks

Network type	Technology	Downlink	Uplink
Satellite	DirecPC	400 kbps	56.6 kbps (thru Tel.)
Cable TV	Cable modem	10–30 Mbps	128 kbps (Shared)
Telephone	ADSL, VDSL Modem	2 Mbps	9.6–640 kbps
Wireless	802.11	1–10 Mbps	9.6–19.2 kbps

links in two directions. The asymmetry in link capacities in the case of a few recent examples N/W technologies have been provided in Table 12.1. Even, otherwise, the asymmetry can occur even in a symmetric client-server setup. The anomalies in bandwidths occur due to the fact that:

- Either physically different media are used for the uplink and the downlink connections, or
- The same medium is split asymmetrically for uplink and downlink connections.

As the table indicates, depending on the N/W technology, the variations between the downlink and the uplink capacities can range from 1:8 to 1:500. The asymmetry can arise not only due to the limitation of N/W technology, but also due to *load asymmetry* or *data volume asymmetry*. The load asymmetry is introduced when a few machines in the N/W handle most of the messages. In particular, the service load asymmetry may happen due to following reasons:

- The client-to-server ratio is high, so, the average load on a server is high.
- The updates are too frequent, so, the clients continually poll the servers for new data.

In some application environments, the asymmetry in volume of data arises due to mismatch in the volumes of data transmitted in each direction. For example, in information retrieval type application, a few URLs (a mouse key click) may result in a huge document to be downloaded. So, the requirements for uplink capacity is much smaller than downlink. Typically wireless connectivity may offer only unidirectional connections. So, the clients may be unable to connect by design rather than just due to technical problems.

In a pull-based system, the clients must have a priori knowledge about what to ask for. Pull-based data retrieval is typically similar to the RPC protocol. Each data transfer from the server is initiated explicitly by a request from a client. In response to the request form a client, the server transfers the requested data. The biggest drawback to implement a pull-based data transfer system is that a back channel should be made available to the clients for fetching desired data. The back channel eats away bandwidth available for data transfer. In a typical environment, where the clients are mobile, the congestion in back channel can be real bottleneck for fetching data.

The availability, as well as the use of the back channel for clients is restricted either because of the security reasons or due to the power problem at the clients, or

both. The saturation of a server due to huge number of client requests may be another severe problem. Indeed, if the *request rate* is higher than the *service rate*, the server will eventually saturate. However, in traditional applications, it can be controlled.

In summary the scale of mobile computing environment seems to be the most important factor contributing to the drawbacks of pull based data delivery model [2, 11, 12].

The major advantages of a push-based data delivery model can be summarized as follows:

- *Efficiency*: The transfer of data is needed only when the updates arrive at a server, or when the new data gets created. Obviously, a client is not expected to know when the new data gets generated at the server.
- *Scalability*: The push-based data delivery model translates into greater scalability. A client need not poll the server in periodic intervals to check for the delivery of data items it may need.
- *Low bandwidth requirement*: There is no need for deploying a back channel. The transfer of data is initiated by the server when some updates are made or some new data get created. So the utilization of entire bandwidth can be made possible only via downstream link, and more data can flow on the channel. Furthermore, the utilization of available bandwidth can be done optimally as it is coordinated at the server end.

12.6 Classification of Data Delivery Models

In the context client-server paradigm the basic difference between data delivery models lies in the fact whether the data transfer is initiated by a client or by the server. Both pull and push based data delivery can be either periodic or aperiodic.

Aperiodic pull is found in traditional systems or request/response kind of data delivery model. Periodic pull arises when a client uses polling to obtain the data it is interested in. It uses a regular schedule to request for the data. Polling is useful in applications such as remote sensing where a client can wait while sending repeated probes, in periodic intervals, for the arrival of the data. The data by itself, is not critical for immediate running of application. In summary, if the transfer is initiated by the client then it can be one of the following types:

(i) *Request/response*: This is found in the traditional schemes such as RPC. A client uses aperiodic pull over point-to-point physical link between the server. Of course, the server may choose to use an one-to-many link for transfer of data. For the client requesting the data, it appears as point-to-point, while the other clients can snoop on the link and get data they have not explicitly requested.

(ii) *Polling*: In some applications such as remote sensing or control applications, a system may periodically send request to a number of sites to obtain changed values. If the information is sent over a point-to-point link, it is a pull based

approach and known as *polling*. But if the data transfer is over an one-to-many physical link then other clients can snoop.

The periodic push can be for the transmission of a set of data updates or newly created data in a regular interval, such as updates in Stock prices. This data delivery model is useful in situations where clients may not always be available. Since the delivery is for unidentified clients, availability of some specific client or a set of specific clients does not matter. The push based model is preferable for various reasons such as:

- High request processing load or the load asymmetry generated by the clients, or
- A large number of clients are interested for the data being pushed, i.e., high client to server ratio.

Aperiodic push is viewed as publish and subscribe type of data dissemination. There is no periodic interval for sending out the data. The biggest problem in aperiodic push is the assumption that the clients are always listening. In a mobile computing environment the clients may like to remain disconnected till the exact time of arrival of the required data for the reason of extending battery life. The clients may also move to the cells different from where they initially wanted to get the data. The push-based model is exposed to snooping, and a client may also miss out the required data, if it is not listening at the opportune time when the data flows on the downlink channel. So, a mechanism is needed to be in place for a client to be able estimate the time for tuning to the downstream channel for the required data. Therefore, such a data delivery model could considered as more appropriate to the situation where what is sent out is not critical to immediate running of the application. On the positive side, the push based transfer uses downstream channel more effectively. In summary, push based data delivery can categorized as follows.

(i) *Publish/Subscribe* [13]: In this model of data delivery, the flow of data is initiated by a server and is aperiodic. Typically, one-to-many link is used to transfer data.
(ii) *Broadcast disks* [1]: Basically, it is a periodic push mechanism. The clients wait until the data item appears on broadcast. In a sense, it is like accessing of a storage device whose average latency is half the interval at which the requested item is repeated on broadcast. The periodic push can use either point-to-point or one-to-many link; though, one-to-many is more likely.

Figure 12.3 provides a broad classification of data transfer mechanisms based on the pull and push based delivery models and physical link characteristics as discussed above.

The characteristics of a link have a role in deciding how the data delivery model can scale up when there is no control in population of clients. In a point-to-point communication, data sent from a source to a single client. Clearly, p-to-p (point-to-point) transfers cannot scale up easily when the number of clients becomes large. In one-to-many communication data is sent to a number of clients. One-to-many communication can be either multicast or broadcast. Usually, multicasting is implemented by sending data to a router which maintains the list of the recipients and

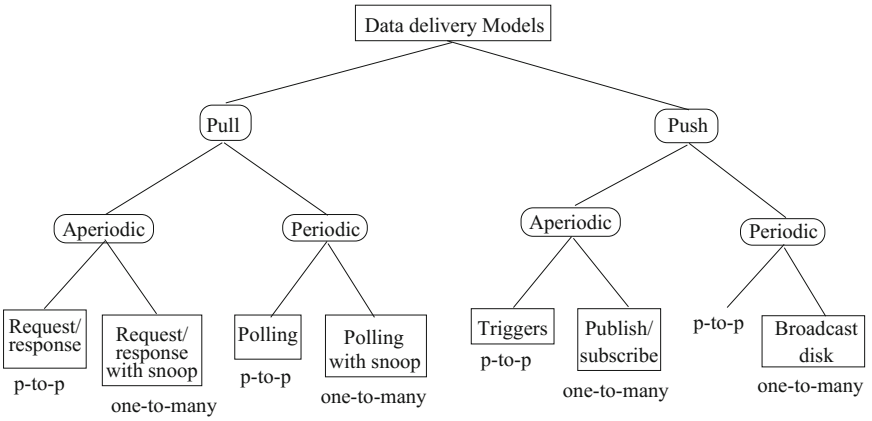

Fig. 12.3 Taxonomy of data transfer

forwards the data to those recipients. So the interests of clients should be known a priori as opposed to broadcast where clients are unidentified. In one-to-many broadcast clients receive data for which they may not be interested at all.

12.7 Broadcast Disk

From the client's perspective, periodic push is similar to accessing a secondary storage. Therefore, the data delivery should be organized to give the best performance to the clients as a local disk would. Suppose in a broadcast schedule, every item appears only once. Then in the worst case, a client may have to wait for one broadcast period for fetching the data item it requires. In the best case, the access can be immediate, and it happens if the client tunes in to listen exactly at time when its item appears in the broadcast channel. It is rarely, the case that all items are accessed equally frequently. Generally, the access pattern tends to be skewed to a few *hot spots*. So it makes sense to capture the pattern of accesses in a broadcast program. *Broadcast disk* is a paradigm for organizing the structure of a periodic broadcast program.

12.7.1 Flat Periodic Broadcast Model

A generic broadcast disk model should be able to capture various access patterns. A flat program is a degenerate case of the generic broadcast disk model. A flat broadcast program shown in Fig. 12.4, can be considered as a logical disk spinning at a speed of one spin in a broadcast period.

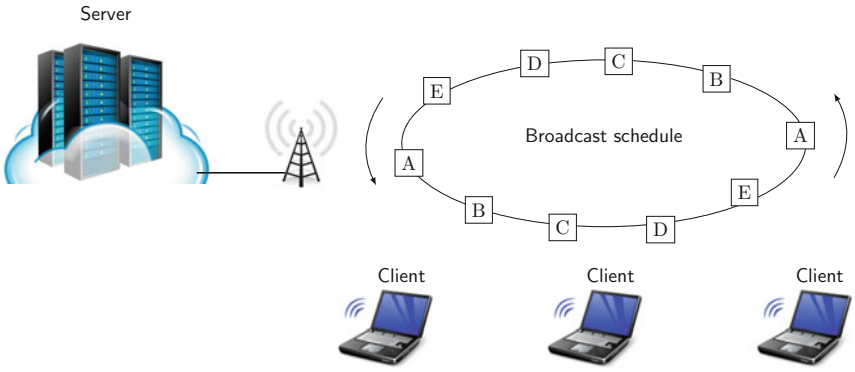

Fig. 12.4 A flat broadcast disk

12.7.2 Skewed Periodic Broadcast

Consider the case when the broadcast data is organized into a multiple number of disks, and each disk spinning with a different speed. The data items are placed into the fastest to the slowest spinning disks in descending order of frequencies of accesses. That is most frequently accessed data is placed on the fastest disk. The least frequently accessed data is placed on the slowest disk.

Depending on whether the broadcast data get updated or not, a periodic broadcast can be considered *static* or *dynamic*. If the sequence of broadcast data remains the same for every broadcast period, then the broadcast is static, otherwise it is dynamic. In the case of dynamic periodic broadcast, the period of broadcast may also vary.

Another way to classify periodic broadcasts would be on the basis of the inter arrival times of two consecutive instances of the same data item. The broadcast is called *regular*, if the inter arrival time of two consecutive instances of the same data item is fixed. The broadcast is *irregular*, if there is a variance in inter arrival times.

12.7.3 Properties of Broadcast Programs

From an abstract point of view, a broadcast programs visualized as an attempt to generate a bandwidth allocation scheme. Given the access probabilities of each client, the job of a broadcast program is to find the optimal fraction of the bandwidth which may be allocated for an item. Assuming no caching at the clients, it is known that the optimal bandwidth that can be allocated for an item is proportional to square root of its access probability [4]. The simplest possible idea would be to generate a random broadcast schedule according to the square root allocation formula; and then hope that this broadcast schedule matches the average inter arrival time between any two instances of same item as expected by the clients. However, the probability is almost

negligible that a random broadcast program may minimize the expected delay due to the variance in the inter arrival times.

Let us consider an example mentioned in [1]. It illustrates following three different broadcast programs involving three pages A, B, and C shown in Fig. 12.5. The performance characteristic of the last program (Fig. 12.5c) is identical to the case where item A is stored on a single-page disk spinning two times faster than a two-page disk storing items B and C. In this case, the waiting time for accessing page A is either 0 page or 1 pages, assuming that the requirement coincides with the broadcast of a page boundary. Therefore, average wait is 0.5 page for A. Whereas the average wait for page B or C is 1.5 pages. Assuming accesses for each page is equally likely, the total wait $(1/3)(0.5 + 1.5 + 1.5) = 7/6$ page. In reality the requirement for a page coinciding with the page boundary of a broadcast is low. So adding 1/2 page to the total wait, we have the delay as $5/3 = 1.67$ pages.

Let us derive the exact expression for the expected delay for an item I from the point when the request is made. Suppose, a request for the value of I arises at some point of time t falling in the interval j as shown in Fig. 12.6. If I is scheduled to arrive during an interval in future, then the maximum waiting time is the interval of time from the beginning of interval j to the starting of the transmission of I as indicated in the figure. The time of t may appear any where within interval j. Assuming each interval to be of unit time, $t \in [0, 1)$, the expected delay is given by

$$\sum_{1}^{N} \int_{0}^{1} \left(t^{j}_{max}(I) - t\right) dt = \sum_{1}^{N} \left(t^{j}_{max}(I) - \frac{1}{2}\right),$$

where N is the number of intervals. Each interval is the time required to transmit one data item. Using the above formula, the expected delays for the arrival of different items on the broadcast channel can be computed as illustrated by the Table 12.2.

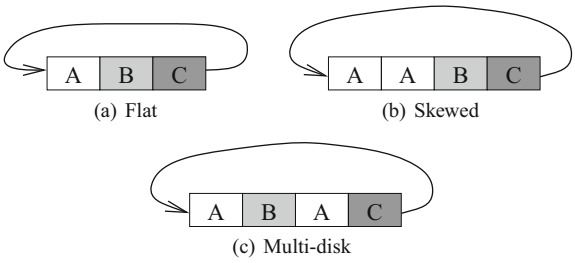

Fig. 12.5 Broadcast programs [1]

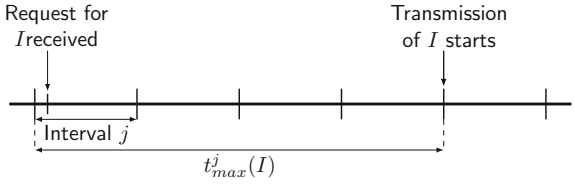

Fig. 12.6 Formulation of maximum waiting time

Table 12.2 Expected delays for arrival of items on channel

Arrival of requests	Expected delay					
	$t^j_{max}(A) - (1/2)$		$t^j_{max}(B) - (1/2)$		$t^j_{max}(C) - (1/2)$	
	Skewed	Multidisk	Skewed	Multidisk	Skewed	Multidisk
Interval 1	0.5	1.5	1.5	0.5	2.5	2.5
Interval 2	2.5	0.5	0.5	3.5	1.5	1.5
Interval 3	1.5	1.5	3.5	2.5	0.5	0.5
Interval 4	0.5	0.5	2.5	1.5	3.5	3.5
Average	1.25	1	2	2	2	2

Table 12.3 Expected delays for arrival of items on channel

Access probability			Expected delay		
A	B	C	Flat	Skewed	Multi-disk
0.333	0.333	0.333	1.50	1.75	1.67
0.5	0.25	0.25	1.50	1.63	1.50
0.75	0.125	0.125	1.50	1.44	1.25
0.9	0.05	0.05	1.50	1.33	1.10
1.0	0.0	0.0	1.50	1.25	1.00

The expected delays for the arrival of items A, B and C respectively are 1.25, 2, 2, in the case of clustered skewed broadcast, and 1, 2, 2 for the case of multi-disk broadcast. If the probability of access for each item is equally likely, then the expected delays for an item in two broadcast methods are:

$$\text{Skewed broadcast} : \tfrac{1}{3}(1.25 + 2 + 2) = 1.75$$
$$\text{Multi-disk broadcast} : \tfrac{1}{3}(1 + 2 + 2) = 1.67$$

Table 12.3 gives the computed delays (in terms of page broadcast period) for page requests corresponding to distribution of access probabilities of the clients. In the case of uniform page access probabilities, flat disk is the best. Non-flat programs are better for skewed access probabilities. Obviously, higher the access probability more is the bandwidth requirement. When page A is accessed with probability 0.5, according to square root formula [4] the optimal bandwidth allocation works out to be $\sqrt{0.5}/(\sqrt{0.5} + \sqrt{0.25} + \sqrt{0.25})$ which is 41%. The flat program allocates only 33% (8% less in) bandwidth. Whereas the multi-disk program (in which page A appears twice in the schedule) allocates 50% (9% excess in) bandwidth.

12.7.4 Advantages of Multi-Disk Program

A multi-disk program performs better (results in less delay) than the corresponding skewed program. This problem can be explained by what is known as "*bus stop paradox*". The paradox is explained as follows. Suppose there are buses plying to different destinations D_1 and D_2, D_3, etc., from a station S. The number of buses to destination D_1 is more than the available other destinations. But all the buses for destination D_1 are clustered in a small window of time t_w in a day, whereas buses to other destination are staggered in more or less in equal intervals throughout the day. Under this scenario, if a person is unable to be reach S within the interval t_w, cannot get to the destination D_1 for the entire day, although the probability of getting a bus to destinations D_1 is higher than the probabilities for other destinations. This happens due the fact that the buses to other destinations are not clustered.

The probability of arrival of a request during a time interval is directly proportional to the length of the interval. This implies that if the variance in broadcast rate (inter arrival rate) of a page is high, then the expected delay increases. If the inter arrival rate of a page is fixed then the expected delay to satisfy any requests for that page at any random time is half of the interval between two successive broadcasts of the same page. The randomness in inter arrival rate can also reduce the effectiveness of a pre-fetching techniques. The client also cannot go into *doze* mode to reduce the power consumption, because the time of arrival of the requested page cannot be estimated. On the other hand, if interval of arrival is fixed, the update dissemination can be planned by the server. This predictability helps in understanding of the update semantics at the client. Therefore, the desirable features of a broadcast program are:

- The inter arrival time of the consecutive copies of a data item should be fixed.
- The length of a broadcast schedule should be pre-defined, after which it should repeat. Equivalently, the periodicity of a broadcast should be fixed.
- It should use as much bandwidth as possible subject to the above two constraints.

12.7.5 Algorithm for Broadcast Program

Algorithm 26 generates a bandwidth allocation for a periodic push based broadcast when information related to page usage is available. This algorithm was originally proposed in [1].

Example
Consider a three disk program in which pages of D_1 to be broadcast 2 times as frequently as D_2 and 3 times as frequently as D_3. That is, the frequencies are: $f_1 = 6, f_2 = 3$ and $f_3 = 2$ and $T_{max} = LCM(f_1, f_2, f_3) = 6$. This implies splitting disks results in following chunks:

Algorithm 26: Generation of broadcast program

begin
  order the pages to be broadcast from *hottest* to *coldest*;
  group the pages into multiple ranges;
  `// Each range represents a logical broadcast disk.`
  `// Assume` N_{disks} `disks denoted by` $D_1, \ldots, D_{N_{disk}}$.
  foreach *(disk* D_i*)* **do**
    `// ` f_i`, must be an integral value.`
    choose relative frequency f_i;
  end
  split each disk into smaller units called *chunk*s;
  $T_{max} = \text{LCM}\{f_i | 1 \leq i \leq N_{disk}\}$;
  foreach *(disk* D_i*)* **do**
    split D_i into equal sized chunk $C_{ij}, j = 1, \ldots, T_{max}/f_i$;
  end
  `// Create broadcast program interleaving disk chunks.`
  for $(i = 1; i \leq T_{max}, i++)$ **do**
    for $(j = 1; j \leq N_{disks}, j++)$ **do**
      broadcast chunk $C_{j,((i-1) \bmod T_j+1)}$;
    end
  end
end

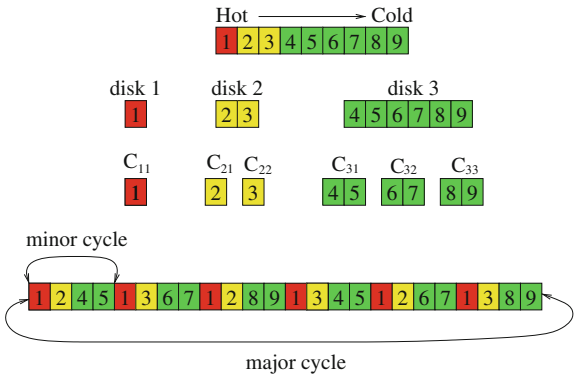

Fig. 12.7 Bandwidth allocation by broadcast algorithm

1. Disk D_1: splitting results in one chunk C_{11}.
2. Disk D_2: splitting results in two chunks C_{21}, C_{22}, and
3. Disk D_3: splitting results in three chunks C_{31}, C_{32}, C_{33}.

Though chunk size of a disk fixed, across the disks the size may be variable. Figure 12.7 illustrates the broadcast program generated by the applying the above algorithm. As shown in the Fig. 12.7, disk D_1 consists of a single page and only one chunk. Disk D_2 consists of two pages and each chunk has one page. Disk D_3 has six pages divided into three chunks, where each chunk has two pages. A minor broadcast cycle consists of three chunks, one from each disks D_1, D_2, D_3. Therefore, each minor cycle consists of four pages, one page each from D_1 and D_2 and two

pages from D_3. One major broadcast cycle consist of $T_{max} = 6$ minor cycles. So, one major cycle consists of 24 pages. The allocation schedule produces three logical levels of memory hierarchy.

- The first being the smallest and the fastest disk D_1.
- The second being the disk D_2 which is slower than D_1.
- The last being the disk D_3 which is slower but larger than both D_1 and D_2.

12.7.6 Parameters for Tuning Disk Model

Three main parameters to be tuned to particular access probability distribution are:

- *The number of disks*. It determines the *different* frequency ranges with which set of all pages should be broadcast.
- *The number of pages per disk*. It determines set of pages of identical frequency range on broadcast.
- *The relative frequency of broadcast*. It determines the size of disks, and hence the arrival time.

The thumb rule is to configure the fastest disk to have only few pages. This because, adding an extra page to the fastest disk adds significantly to the delay of arrival time for the pages in the slower disks. The constraint requiring frequencies to be positive integers leads to a broadcast schedule with fixed inter-arrival time for the pages. A regularity in inter-arrival time substantially eases the maintenance of update semantics, and the predictability helps the client side caching strategy. All pages in the same disk get same amount of bandwidth as they are broadcast with the same frequency.

12.7.7 Dynamic Broadcast Program

The algorithm discussed above generates a static broadcast program. It means there is no change in the broadcast period, the amount of broadcast data or the values of data. In other words, there is no dynamicity in broadcast data. But it is possible to introduce dynamicity as follows.

- *Item placement*: Data may be moved around in between disks. It means items of data traverse the levels of disk hierarchy. In other words, some item may lose importance while other items may gain importance. The movement of items influences the client side caching strategy for pre-fetching (hoarding).
- *Disk structure*: The hierarchy of disks itself may be changed. For example, the ratios of the disk speeds can be modified. An entire disk can be removed or a new disk may be added.

- *Disk contents*: The contents of disk may change. Extra pages may be added to a disk or some pages may be removed from a disk.
- *Disk values*: Some pages on the broadcast may change in value.

Read-only and update type of broadcasts get affected by the dynamicity involving *Item placement*, *Disk structure* and *Disk contents*. However, *Disk value* introduces dynamicity that is applicable to the update scenario only. The modifications involving *Item placement* or *Disk structure* influences the relative frequencies as well as the order of the appearances of data items already on broadcast. Whereas the update of the value of a data item does not alter the relative frequencies. So, in absence of the updates, the first two types of modifications primarily affect the performance. The performance of client side caching is affected as caching algorithms need information about the latency of data items. This results in client side storage to be sub-optimally used. However, an advance notification may mitigate the problem.

Dynamicity introduced due to *Disk Contents* does not influence the items that appear on the broadcast. However, some items which appeared previously may disappear and some new items may appear. It can be viewed an extreme case of the *Item Placement*, i.e., the items removed may be assumed have infinite latencies and appear in another disk. The clients can cache an item before it disappears if an advance warning is provided. Or even a client can evict the item from the cache to make room for a new item. *Data Value* updates introduces the problem of data consistency. To take care of this situation cached copies at client should be updated or invalidated.

12.7.8 Unused or Empty Slots in Broadcast Disk

The broadcast schedule may consists of many unused slots or pages [5]. It happens specially when the number of pages in a disk is not an exact multiple of the number chunks into which the disk is partitioned. These unused slots may be used opportunistically to enhance the quality of broadcast by including indexes, updates, invalidations, or even the information pertaining to dynamically emerging situation. The guiding principle of broadcast based dissemination is that the number of disks should as small as possible. Normally, in the order of 2–5. However, the number of pages to be broadcast is often substantially large. Therefore, the number of unused slots, if any, is expected to be quite small in number. In other words, it may be possible to tweak the relative frequencies slightly to reduce the number of unused slots.

If unused slots are not used sensibly, it may lead to substantial wastage of bandwidth. To understand how it may occur, let us modify the previous example as follows. Suppose a list of 12 pages are to be placed on the broadcast channel. We divide these pages into 3 disks:

D_1: has 1 page,
D_2: has 2 pages, and
D_3: has now 9 instead of 6 pages.

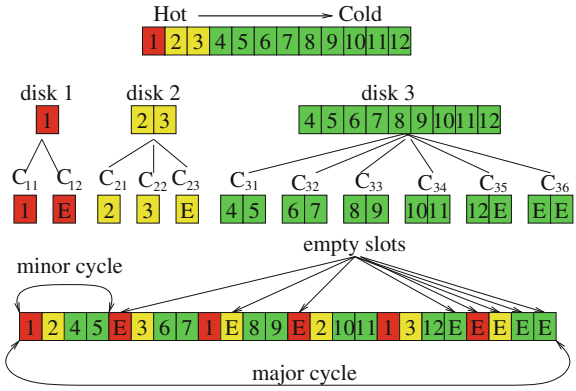

Fig. 12.8 Many unused slots may be generated by broadcast program

D_1 is smallest and fastest, D_3 is largest and slowest. Let the frequencies of D_1, D_2 and D_3 be 3, 2, and 1 respectively. Since, $T_{max} = LCM(3, 2, 1) = 6$, the chunking algorithm divides pages of D_1 into 6/3 = 2 chunks, the pages of D_2 into 6/2 = 3 chunks and the pages of D_3 into 6/1 = 6 chunks. The number of pages per chunk in disk D_1, D_2 and D_3 are $\lceil 1/2 \rceil = 1$, $\lceil 2/3 \rceil = 1$ and $\lceil 9/6 \rceil = 2$ respectively. So, the chunking requires one empty page each to be padded to each chunk of disks D_1 and D_2. In the case of D_3, a padding of 3 empty pages is needed. So, at total of 5 empty pages are inserted as padding to generate integer number of pages for the chunking of broadcast data. Figure 12.8 illustrates the process of chunking and depicts that 8 empty pages appear in broadcast cycle on a single major cycle consisting of 24 pages. It leads to a wastage of 33% of the broadcast bandwidth.

12.7.9 Eliminating Unused Slot

Unused slots in the chunks in a broadcast schedule are like holes in disk fragmentation. A broadcast schedule consists of three types of chunks:

1. Fully used chunks: every slots in such a chunk have useful data.
2. Partially wasted chunks: some slots in such chunks have data and the other slots are free.
3. Fully wasted chunks: all slots in such a chunk are free.

The simple strategy to eliminate unused slots is to compact the holes in the broadcast schedule. However, the compacting technique applies only under some stringent assumptions [5]. These assumptions are:

1. The client population do not change,
2. No update is allowed,
3. The clients do not employ pre-fetching or caching,
4. The clients do not use the uplink channel,

5. When a client switches to a public channel, it can retrieve data pages without wait,
6. Each query result is represented by one page, and the length of each page is the same.
7. A server uses only one channel for broadcast, and broadcast is reliable.

These assumptions essentially imply that access patterns of the mobile clients remain unchanged over time.

The compaction algorithm first needs to compute the indices of unused slots in each wasted chunk of a disk. The computation of the indices turns out to be simple due to the assumptions presented above. But to concretize the computation formula a few notations become handy.

NP_i : number of pages in disk D_i.
NC_i : number of chunks in disk D_i.
NS_i : number of slots in a chunk in disk D_i.

The basic idea behind the algorithm is to first determine the schedule by Algorithm 26 of Sect. 12.7.5. According to this algorithm, the number of slots are:

$$NS_i = \left\lceil \frac{NP_i}{NC_i} \right\rceil = \left\lceil \frac{NP_i}{T_{max}/f_i} \right\rceil = \left\lceil \frac{NP_i \times f_i}{T_{max}} \right\rceil,$$

where T_{max} is LCM of the chosen frequencies for the disks.

Consider the example shown in Fig. 12.8. The number of slots in a chunk in three different disks as determined by the algorithm are:

$$NS_1 = \left\lceil \frac{1 \times 3}{6} \right\rceil = 1, NS_2 = \left\lceil \frac{2 \times 2}{6} \right\rceil = 1, \text{ and } NS_3 = \left\lceil \frac{9 \times 1}{6} \right\rceil = 2.$$

The execution of the original algorithm organizes the broadcast schedule (major cycle) into chunks consisting of 24 slots. Out of these 8 slots are empty, which implies only $24 - 8 = 16$ slots have useful data. Modified algorithm places a logical cut line at the end of the slot 16. To the right of this cut line there are 8 slots, out of which only 3 have useful data. Furthermore, there are exactly 3 wasted (empty) slots to the left the cut line. The modified algorithm moves the data from the right to the left of the cut line to empty slots preserving their relative order of occurrences. The movement of data is illustrated by Fig. 12.9. So the strategy of the algorithm is compact the holes present in partially wasted chunks by moving data appearing to the right of the cut-line. This action pushes all empty slots to the right of the cut-line

Before formally presenting the modified algorithm, we observe that empty slots are located either in a fully wasted chunk (all free slots) or a partially wasted chunk (with some free slots). The indices of fully wasted chunks in D_i is given by

$$FW_i = NC_i - \left\lceil \frac{NP_i}{NS_i} \right\rceil$$

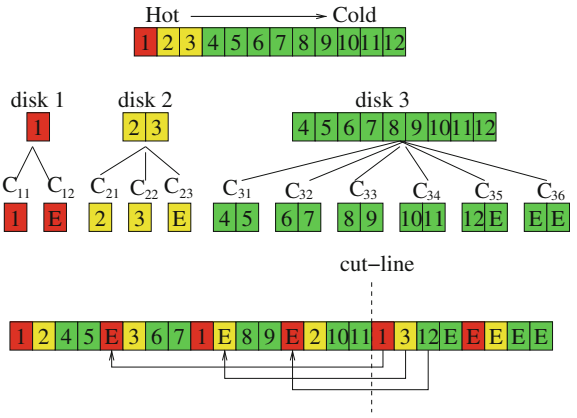

Fig. 12.9 Eliminating unused slots

Therefore, fully wasted chunks in disk D_i are C_{ij} where, $NC_i - FW_i + 1 \leq j \leq NC_i$. So, the number of fully wasted chunks in D_3 of our running example is:

$$\begin{aligned} FW_1 &= NC_1 - \left\lceil \frac{NP_1}{NS_1} \right\rceil = 2 - \left\lceil \frac{1}{1} \right\rceil = 1 \\ FW_2 &= NC_2 - \left\lceil \frac{NP_2}{NS_2} \right\rceil = 3 - \left\lceil \frac{1}{1} \right\rceil = 2 \\ FW_3 &= NC_3 - \left\lceil \frac{NP_3}{NS_3} \right\rceil = 6 - \left\lceil \frac{9}{2} \right\rceil = 1, \end{aligned}$$

Let w be the number of wasted slots in a partially wasted chunk in D_i. The value of w can vary between 1 and $NS_i - 1$, i.e., $1 \leq w \leq NS_i - 1$. There can be only one partially wasted chunk and it occurs only when $NP_i \neq NC_i \times NS_i$, where all the quantities have integral values. Therefore, the condition for checking partially wasted chunk is simply:

$$\left\lceil \frac{NP_i}{NS_i} \right\rceil - \left\lfloor \frac{NP_i}{NS_i} \right\rfloor = 1.$$

The index of the partially wasted chunk C_{ij} is $j = NC_i - FW_i$. The total number of wasted slots is $NS_i - NP_i$, and the number of fully wasted chunks is FW_i. Therefore, the number of wasted slots in a partially wasted chunk is given by

$$w = NS_i \times (NC_i - NP_i) - FW_i \times NS_i.$$

So, in chunk C_{ij}, the empty slots are E_{ijk}, where $NS_i - w \leq k \leq NS_i$.

Once we have identified the indices of empty slots, the compaction algorithm becomes straightforward. The algorithm creates an array `Broadcast[.]` which records the pages to be selected in sequence to appear on the broadcast channel. The modified algorithm [5] consists of two phases.

1. In the first phase, the required number of empty pages are added to the disk chunks as needed for the allocation of same integral number of slots in a minor cycle. So this phase is identical to Algorithm 26.
2. In the second phase, the cut-line is determined and the contents of the occupied slots after the cut-line are swapped with the corresponding empty slots before the cut-line. So, the basic compaction step is to find the corresponding sequence numbers of slots which need to be swapped.

Phase 2 of the algorithm is essentially a compaction step. It does not really make any additional contribution to the original algorithm for creating a broadcast cycle. Algorithm 27 gives the details of this phase and included here for the sake of completeness.

Algorithm 27: Using empty slots.

begin
`// Broadcast cycle created by Algorithm 26`
calculate total slots TS in a major cycle;
calculated total wasted slots TWS in a major cycle;
determine cut-line $CL = TS - TWS$;
find the nonempty slots after the CL;
record these slots in the array *Moved*.
`// Use a sequence number SN to sequence of empty slots in a major cycle as 1,2,…, TS`
find out SN of an E_{ijk} before the cut-line and replace it with a record in *Moved* array in sequence.
`// Broadcast the contents of the Broadcast array in sequence.`
for $(i = 1; i \leq CL; i++)$ **do**
  broadcast *Broadcast*[i];
end
end

The first step of algorithm computes total number of slots (TS) which is equal to

$$TS = T_{max} \times \sum_{i=1}^{S} NS_i = T_{max} \times \sum_{i=1}^{S} \left\lceil \frac{NP_i \times f_i}{T_{max}} \right\rceil$$

The next step determines the total number of wasted slots (TWS) in one major cycle. Then the cut-line is identified. The computation performed is as follows.

$$
\begin{aligned}
TWS &= \sum_{i=1}^{S}((NS_i \times NC_i - NP_i) \times f_i) \\
&= \sum_{i=1}^{S}\left(\left(NS_i \times \frac{T_{max}}{f_i} - NP_i\right) \times f_i\right) \\
&= \sum_{i=1}^{S}(NS_i \times T_{max} - NP_i \times f_i)
\end{aligned}
$$

The core of the compaction of data which is performed next. The data slots to right of cut-line are moved into a separate array *Moved*. The slots from which data have been placed in *Moved* array can now be declared as empty. Then data is moved from *Moved* array to the empty slots just before the cut-line. Although compaction algorithm does not disturb chunk orders from disks in minor cycle, it re-orders the pages that make up a chunk. Hence, using compaction requires the client to change the order of servicing queries.

12.8 Probabilistic Model of Broadcast

Wong [14] proposed a probabilistic approach for generating cyclic broadcast schedule. The approach appears impractical though it improves the performance of skewed data access by selecting data items according to their relative access frequencies. Wong's result tells that for fixed-sized data objects, access time is minimized if $\frac{p_i}{p_j} = \frac{\sqrt{q_i}}{\sqrt{q_j}}$ for all i, j. We can simplify the result by summing the denominator over all j, and get $p_i = \frac{q_i}{\sum_j \sqrt{q_j}}$.

The difficulty with the method is that it generates an acyclic broadcast schedule. The access time of a data item could also be arbitrarily large. It performs poorly compared to other skewed broadcast methods discussed later in the text. But Wong [14] proposed a heuristic to work around. The heuristic assumes that items are of equal size. The underlying idea is to convert acyclic broadcast to cyclic one which can give near optimal results. This method is further refined for variable sized data items and presented in a neat understandable format in [9, 15]. A few important parameters of the algorithm are:

1. N items to be broadcast.
2. The length of item i is l_i.
3. Page access probabilities are q_i, for $i = 1, 2, \ldots, N$.

The algorithm states that optimal latency is achieved when following two conditions are met.

C1: Copies of each data item are equally spaced, i.e., inter appearance gap between two copies of same item is constant.

C2: The spacing s_i of two successive copies of same item is proportional to square root of its length l_i, i.e., satisfying Eq. 12.1.

$$s_i \propto \sqrt{\frac{l_i}{q_i}}, \text{ or } s_i^2 \frac{q_i}{l_i} = c, \text{ where } c \text{ is a constant} \tag{12.1}$$

Both conditions cannot be met simultaneously. So, the heuristic only approximates the optimal results obtained from theoretical analysis. Hameed and Vaidya [9] introduced a pair of additional variables B_i and C_i for each item i:

1. B_i: is the earliest time when next instance of item i should be transmitted, and
2. $C_i = B_i + s_i$.

This essentially means C_i is the worst case completion time of the next instance of item i. Using the above idea, Hameed and Viadya proposed a technique for generating cyclic broadcast as provided by Algorithm 28 [9]. The simulation results presented in [9] shows that this method of broadcast performs close to the analytically obtained optimal result. For more details on this algorithm the reader can refer to the original paper.

Algorithm 28: Generating broadcast program.

begin
  `// Initializations begin.`
  $T = 0$; `// Represents time parameter.`
  foreach *($i \in N$)* **do**
    $B_i = 0$; `// Earliest time for next instance of item` i
    $C_i = s_i$. `// Spacing between two successive copies of` i
  end
  `// Initialiations end.`
  for *($i = 1, i < N, i = i + 1$)* **do**
    compute the optimal spacing s_i using Eq. 12.1;
  end
  repeat
    determine a set of items $S = \{i | B_i \leq T, 1 \leq i \leq N\}$;
    choose i_{min} such that $C_{i_{min}} = \min\{C_i | 1 \leq i \leq N\}$.
    set $B_{i_{min}} = C_{i_{min}}$;
    $C_{i_{min}} = B_{i_{min}} + s_{i_{min}}$;
    wait for transmission of item i_{min} to complete;
    $T = T + l_{i_{min}}$; `//` l_i `is length of item` i
  until *(system is live)*;
end

12.9 Memory Hierarchy

In a typical client server model, the memory hierarchy consists of

- Client side cache and disk
- Server side cache and disk.

In a push based dissemination system, broadcast introduces an intermediate level of memory hierarchy between the client and the server [1]. If the client's cache or disk does not have an item then broadcast is used to satisfy the request. Otherwise, if a back-channel is provided then the client puts an explicit request for the item. In such a case, the client waits for the item and tunes in at the exact time to access it.

Multilevel broadcast disk places a sub-hierarchy within the broadcast.

- The fastest disk is at the top level and the slowest disk is at the bottom level.
- We can view this combination hierarchy in broadcast system as shown in the Fig. 12.10.

As opposed to traditional memory hierarchy, the sub-hierarchy introduced by broadcast disks has some distinctive features, namely:

- *Tunable access latency*: By choosing number of disks it is possible to control access latency of data items. In fact, it is possible to create arbitrary fine-grained

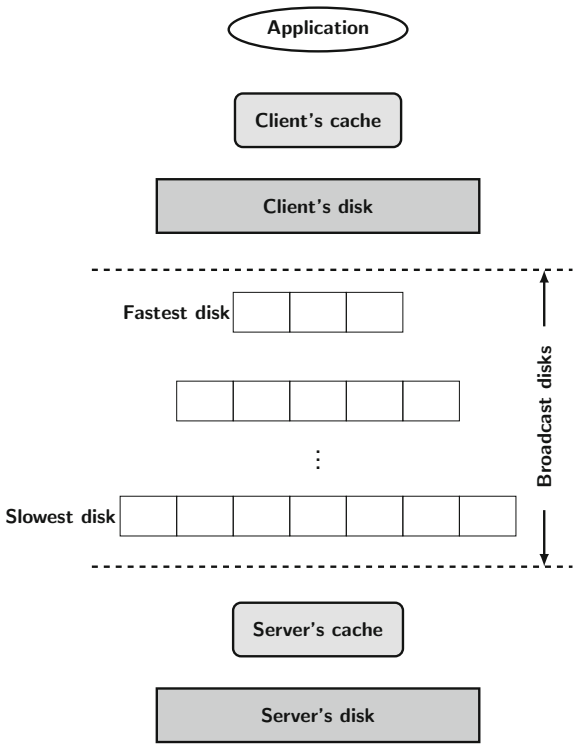

Fig. 12.10 Memory hierarchy of broadcast disks [1]

memory hierarchy with more number of disks. The biggest advantage is that the increasing the number of *disks in air* does not involve any extra h/w cost.

- *Cost variation*: Normally the access cost is proportional to the level of the memory hierarchy. In broadcast system this is true only for the average cost. The instantaneous cost can vary from zero to the broadcast period. It may be cheaper to use a back-channel for fetching the required data than to wait.

12.10 Client Cache Management

In a push based data dissemination, the server uses its best knowledge and the requirements of the clients. However, in a mobile environment, the number of clients which a server may have to serve is very large. So, it is difficult for a server either to gather knowledge or collate the data requirements of all the clients. Many clients may even be unidentified. Furthermore, back channels may not available to the clients to upload their profiles. Thus, the server can possibly determine an expected requirement pattern of an average client which serves a large cross section of clients. Consequently, many clients may have to wait long time for some data which they want quickly, while a few others may observer data flowing into the broadcast channel much before they may actually need it.

In the extreme cases, the average requirement pattern may completely mismatch the requirements of some clients while it may perfectly match the requirements of the other clients. In order to optimize an application's performance, it is important to filter out mismatches of a client's profile from that of the average client. The technique is to employ a client side cache management and use a pre-fetching strategy to store anticipated requirements of data in the client's cache.

In general there are two cache management techniques.

1. Demand-driven.
2. Pre-fetching.

In demand driven caching, a cache miss occurs when the data is accessed for the first time. The data is brought into cache when it is accessed for the first time. On the other hand, pre-fetching represents an opportunistic use of cache resources. The data is stored in the cache in anticipation of future use. Therefore, there is no delay even when the data accessed for the first time. But pre-fetching leads to the wastage of cache resources if the data is not eventually accessed.

The major issue that needs to be addressed before employing any caching strategy is *victim selection*. Replacing a cached page by a new page is the primary concern in a caching strategy. It is critical to performance. For example, sometimes an immediate references to the victim page may occur after it has been replaced, while there is a possibility that no reference is made to a new page which has just been brought into the cache. This situation may occur due to the locality associated with address references. So, the replacement policy should be such that a page having the lowest utility at that moment gets evicted.

In case of a dissemination based information system, the burden of data transfer rests on the server. It introduces certain differences that change the trade-offs associated with traditional caching. So for the sake of completeness, and for a better appraisal of the issues involved, a close examination of the problems associated with of client side caching with relative to push based system is needed.

In a pull-based system, when client faults on a page it can explicitly issue a request for the required page. In contrast, the absence of back channel in a push-based system forces a client experiencing a page fault to keep listening until the required page arrives on the broadcast channel. It essentially represents the fact that the nature of the communication medium (air) in a wireless communication system is sequential. Therefore, no random access is possible. Consequently, a client must wait to access the data in the order it is sent by the server. The access cost, therefore, is non-uniform. The data is not equidistant from the client's cache. It happens due to the multi-disk framework. This is in contrast with a traditional system where the cost is uniform.

12.10.1 Role of Client Side Caching

The role of any caching strategy is to choose an appropriate cache size that gives the best response to a client's applications. A large cache size can provide the best performance, as it is possible to cache most of the data requirements of the applications running at the client by liberally caching data from the broadcast channel. But normally the size of a client's cache is significantly smaller than the size of database at the server. The reasons are two-fold: (i) the cost of installing a large cache is very high, and (ii) a large cache makes the client heavy and thus non-portable. The smallness of cache size puts constraint on the decision as to what should be cached and what data should be evicted from the cache.

In a pull-based system, the performance of caching is optimal if it based on access probabilities. The access time remains uniform for pulling out different items from a server's database. It implies that all the cache misses have same performance penalty. However, this not true in a broadcast system. The server generates a schedule for the broadcast taking into account the access needs of all clients. This way unlimited scalability can be achieved by broadcast for data services. However, the attempt to improve performance for one of the access probability distributions lead to degradation of performance for another access probability distribution.

12.10.2 An Abstract Formulation

Let us consider an abstract formulation of the problem of client side caching. Suppose, a server S disseminating D pages for n clients $C_1, C_2, \ldots, C_n$. Let A_i be the access profile (data requirement for applications) of C_i. The first step is to gener-

ate a global access profile by aggregation of the access requirements of the clients. Let $A_s = \sigma(A_1, \ldots, A_n)$, where σ is an appropriately chosen aggregation function. Assume that the server has the ability to generate an optimal broadcast schedule for a given access profile. Let $\beta(A_s)$ be the generated broadcast program. Note that $\beta(A_s)$ is optimal iff $A_s \equiv A_1$. The factors causing broadcast to be sub-optimal are:

- The server averages the broadcast over a large population of clients.
- The access distributions that the clients provide are wrong.
- The access distributions of the clients change over time.
- The server gives higher priority (biased) to the needs of clients with different access distributions.

From the point of view of an individual client the cache acts as a filter as depicted in the picture of Fig. 12.11. The client side cache filters accesses by storing certain pages (not necessarily hot pages) such that filtered access distribution, namely $F_i = A_i - c_i$ matches A_s, where $c_i \subseteq D$ is the set of cached pages. If $c_i = \phi$ then $F_i = A_i$. If the server pattern A_s is different from client's access pattern $\beta(A_s)$ will be different from $\beta(A_i)$. In other words, for the cache-less clients the performance is sensitive to how close is the access pattern to the *average* client's access profile. In general, client cache should filter accesses to close the gap between A_s and F_i.

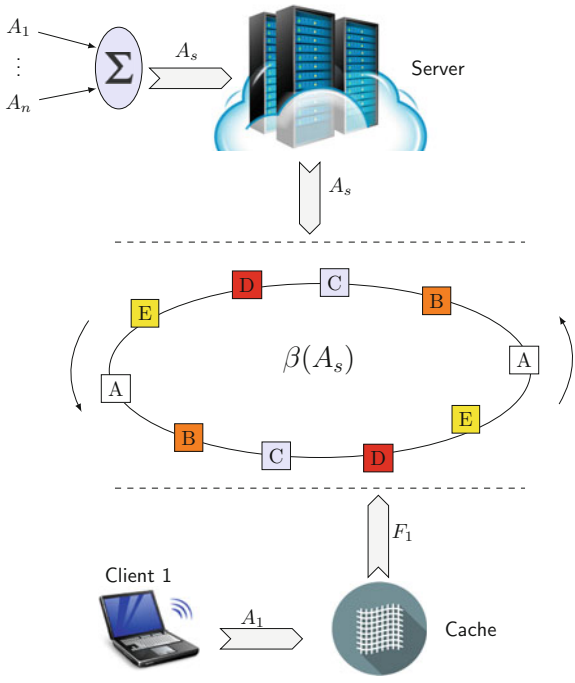

Fig. 12.11 A view of client side cache

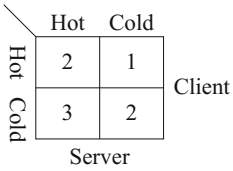

Fig. 12.12 Cost of prefetching

12.10.3 Consideration for Caching Cost

The traditional probability based caching at the client is not suitable due to the sequential nature of the retrieval from the broadcast channel. The cache should act as a filter selectively permitting those requirements for which the local access probability and the global access probability are the same. The server allocates bandwidth to each page based on its global likelihood of access. A client should cache only those pages which have significantly higher local probability of access. It is possible that either these probabilities match or they mismatch. So the cost consideration should take into account the *hot* and *cold* categorization of the pages both with respect to the server and the with respect to the client. What is cold at server may be hot at client and vice versa. Therefore, from prospective of a client caching those pages for which the cost of acquisition is significantly higher is an automatic choice. The table in Fig. 12.12 shows the priority for caching. However, in general, it is not possible to have a binary categorization of the priorities as suggested in the figure. In a broadcast disk environment, a client faulting on a page has to wait for the data to appear on broadcast channel. Therefore, in a broadcast disk environment, an appropriate cost based caching should be developed.

12.10.4 Cost-Based Caching Scheme: PIX and LIX

The idea of a cost-based caching scheme for broadcast disk is to increase cache hits of the pages for which the client has high access probabilities. Suppose p_i is the access probability of a page that is broadcast at a frequency x_i. Then the ratio p_i/x_i known as PIX (Probability Inverse frequency X) can be used for selecting victim for eviction of items from a client's cache. If p_i is high but x_i is low, then the item is hot for client but not so hot for server, then PIX will be high. On the other hand if p_i is low but x_i high, then the item is not hot for client but it is hot for server. So, by using least PIX as the criterion for the selection of victim (for cache eviction), we can ensure that a less frequently accessed page which occurs more frequently in broadcast schedule will be evicted from client cache. For example, consider two page a and b with access probabilities 0.01 and 0.05 respectively. Suppose the respective frequencies of occurrence of a and b on broadcast cycle are 2 and 100. Then PIX(a) = 0.005 and PIX(b) = 0.0005. On the basis of PIX values, b will be the victim page. Although the probability of access for b at the client is more than that of a, from the server

prospective b is hotter than a. The server broadcasts page b more frequently than it does the page a. Therefore, a is preferred over b for caching. Unfortunately PIX is not an implementable policy, as it requires advance knowledge of access probabilities. So, we need to invent a policy which is an implementable approximation to PIX.

Though access probabilities of a page cannot be measured or known beforehand, we may find an approximate measure of the same. The idea is to find a running average of the number of times each page is accessed. The running average assigns importance to the client's recent access pattern. LIX, as it is known, is an adaptation of LRU and PIX.

It maintains the cache as a singly linked list. For each broadcast disk a separate linked list is maintained at the client. When a page is accessed it is moved to the top of the list. This ensure that the pages accessed recently are not the candidates for eviction. When a new page enters the cache, LIX evaluates the *lix* value for the bottom of each chain. The page with smallest *lix* value is evicted to allow the new page to be stored. Although page is evicted with lowest *lix* value the new page joins the linked list corresponding to broadcast disk it belongs. The *lix* value is evaluated by the ratio of the estimated running average of the access probability and the frequency of broadcast for the page. For estimating of the running probability of access p_i each page i, client maintains 2 data items, namely, p_i value and the time t_i of the recent most access of the pages. It re-estimates p_i, when the page i is accessed again. The probability estimate is carried out according to the following rules.

1. Initialize $p_i = 0$ when page i enters the cache.
2. The new probability estimate of p_i is done when page is accessed next using the following formula
$$p_i^{new} = \frac{\lambda}{currTime - t_i} + (1 - \lambda)p_i,$$
where λ, $0 < \lambda < 1$, is an adjustable parameter.
3. Set $t_i = currTime$ and $p_i = p_i^{new}$.

As p_i is known for each page and so are their frequencies, *lix* values can be calculated easily. Essentially LIX is a simple approximation of PIX, but it was found to work well [1].

12.10.5 Pre-fetching Cost

The goal of pre-fetching is to improve the response time for a client application. Pre-fetching is an optimistic caching strategy, where the pages are brought in anticipation of future use. The dissemination oriented nature of broadcast system is ideal for pre-fetching, but it is slightly different in approach compared to a traditional pre-fetching scheme. In a traditional system, the client makes an explicit request for pre-fetching. Therefore, it is anadditional burden on resources. But in a broadcast environment

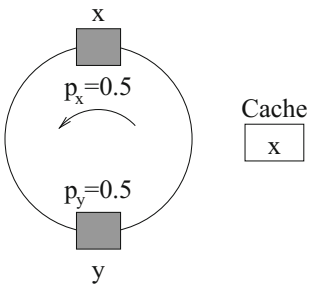

Fig. 12.13 Illustrating pre-fetch cost

the pages anyway flow past. Therefore, the client can pre-cache the selected pages if these are important. On the flop side, an eager pre-caching results in wastage of cache slots in respect of pages cached *too early*. In a traditional system, an improvement response time at a client can be achieved by

- minimizing the cache miss rate, and
- reducing the cost of a cache miss.

Though the improvement in broadcast environment also depend on the same parameters, here the considerable gains can be achieved by reducing the latency when a cache miss occurs.

Example
Suppose a client is interested in two pages x, y. The server broadcast them on a flat disk with 180° apart. There is a single cache slot at the client. Figure 12.13, illustrates the scenario. Under a demand-driven strategy, the client caches a page, say x, as a result of the requirement for the page. In this case, all the subsequent requests for x can be satisfied without delay, but if there is a requirement for y, then it has to wait till the page comes by on the broadcast. Then x is replaced y, and it remains resident till a request for x results in a cache miss when x again is brought into the cache. In this strategy the expected delay on a cache miss could be one half of the disk rotation. The cost for accessing a page is given by

$$C_i = p_i * m_i * d_i$$

where p_i is probability of access, m_i expected probability of a cache miss, d_i is the expected delay before the page i arrives on broadcast. The total expected cost for access over all pages in demand-driven strategy is:

$$\sum_{i \in \{x,y\}} C_i = 0.5 * 0.5 * 0.5 + 0.5 * 0.5 * 0.5 = 0.25$$

one quarter of a disk spin time.

Suppose we cache the page x when it arrives on broadcast and replace it with y when the latter arrives, the strategy is called *tag team*. The cost will be

$$\sum_{i \in \{x,y\}} C_i = 0.5 * 0.5 * 0.25 + 0.5 * 0.5 * 0.25 = 0.125$$

Therefore, tag team caching can double the performance over demand-driven caching. The improvement comes from the fact that a miss in tag team strategy can only be due to some requirement in half the broadcast cycle. In contrast the cache miss in demand-driven strategy can occur at any time.

12.11 Update Dissemination

In a client-server system update dissemination is a key issue concerning the performance versus the correctness tradeoff. A client accesses from its local cache, whereas the updates are collected at the server. Therefore, keeping the consistency of client's cache with the updates is a big problem. Any consistency preserving—by notification or invalidation—mechanism must be initiated by the server. However, it is possible that different applications require varying degree of consistency in data. Some can tolerate some degree of inconsistency. In general, the environment must guarantee consistency requirement that is *no weaker* than an application's requirement. Obviously, the stronger is the consistency guarantee the better it is for the application. However, the requirement for a stronger consistency hurts the performance as the communication and the processing overheads are more.

12.11.1 Advantages of Broadcast Updates

Specially, if there is no back channel, maintaining the consistency client's cache becomes entirely the responsibility of the server. On the other hand, there are some added advantages of a broadcast system, namely,

- The communication overheads are significantly cut-down as the notification of the updates are initiated by the server. However, in absence of any notification the client has to poll. Polling is not possible unless there is a back channel.
- A single notification will do, as it is available on broadcast to all the clients.
- The client's cache is automatically refreshed at least once in each broadcast period.

12.11.2 Data Consistency Models

The notion of data consistency depends on applications. For example, in a database application, the consistency of data normally tied with transaction serializability. But many applications may not require full serializability. So we need to arrive at

some weaker forms of the correctness. In a dissemination based system the notion of consistency is not fully understood till date. Therefore, the focus of our discussion is on a broadcast disk environment where all updates are performed at the server. The type access is allowed to the clients is read-only. The examples of such applications could be stock notification, weather report, etc. The models of data consistency normally employed are:

- *Latest value*: The client is interested for accessing the recent most value. The clients perform no caching and the server always broadcast the updated value. There is no serializability, i.e., mutual consistency among data items is not important.
- *Quasi-caching*: Defined per-client basis using a constraint specifying the tolerance of slackness with respect to the *Latest value*. A client can use cached data items and the server may disseminate updates more lazily.
- *Periodic*: Data values change only at some pre-specified intervals. In a broadcast environment, such intervals can be either be a minor or a major cycle of a broadcast. If a client caches the values of the broadcast data, then the validity of this data is guaranteed for the remainder of the period in which the data is read.
- *Serializability*: The notion serialiability is important for the context of transaction processing. But serializability can be implemented in the broadcast dissemination model, using optimistic concurrency control at the clients and having the server broadcast the update logs.
- *Opportunistic*: For some applications, it may be acceptable to use any version of data. Such a notion of consistency allows a client to use any cached value. It is also quite advantageous for long disconnection.

In *Latest value* model a client has to monitor broadcast continually to either invalidate or update the cached data. In contrast *Periodic* model fits well with the behaviour of a broadcast disk model. It does not require the client to monitor broadcast, since data changes only in certain intervals. *Quasi-caching* and *Opportunistic* models depend heavily on data access semantics of specific applications. *Serializability* model is applicable in transaction scenarios.

References

1. S. Acharya, M. Franklin, S. Zdonik, Dissemination-based data delivery using broadcast disks. IEEE Pers. Commun. **2**(6), 50–60 (2001)
2. S. Acharya, R. Alonso, M. Franklin, S. Zdonik, Broadcast disks: data management for asymmetric communication environments. ACM SIGMOD Rec. **24**(2), 199–210 (1995)
3. D. Aksoy, M.S.F. Leung, Pull versus push: a quantitative comparison for data broadcast, *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 3 (2004), pp. 1464–1468
4. M. Ammar, J. Wong, The design of teletext broadcast cycles. Perform. Eval. **5**(4), 235–242 (1985)
5. Y.I. Chang, C.N. Yang, A complementary approach to data broadcasting in mobile information systems. Data Knowl. Eng. **40**(2), 181–194 (2002)

6. M. Franklin, S. Zdonik, Dissemination-based information systems. Data Eng. **19**(3), 19–28 (1996)
7. M. Franklin, S. Zdonik, Data in your face: Push technology in perspective. ACM SIGMOD Rec. **27**(2), 516–519 (1998)
8. D.K. Gifford, R.W. Baldwin, S.T. Berlin, J.M. Lucassen, An architecture for large scale information systems. SIGOPS Oper. Syst. Rev. **19**(5), 161–170 (1985)
9. S. Hameed, N.H. Vaidya, Efficient algorithms for scheduling broadcast. ACM/Baltzer J. Wireless Netw. **5**(3), 183–193 (1999)
10. G. Herman, K.C. Lee, A. Weinrib, The datacycle architecture for very high throughput database systems. SIGMOD Rec. **16**(3), 97–103 (1987)
11. C.-L. Hu, M.-S. Chen, Adaptive balanced hybrid data delivery for multi-channel data broadcast, *IEEE International Conference on Communications, 2002. ICC 2002*, vol. 2 (IEEE, 2002), pp. 960–964
12. Q. Hu, D.L. Lee, W.-C. Lee, Performance evaluation of a wireless hierarchical data dissemination system, *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking* (ACM, 1999), pp. 163–173
13. Y. Huang, H. Garcia-Molina, Publish/subscribe tree construction in wireless ad-hoc networks, *Mobile Data Management (MDM'03)* (2003), pp. 122–140
14. J.W. Wong, Broadcast delivery. Proc. IEEE **76**(12), 1566–1577 (1988)
15. J. Xu, J. Liu, Broadcast Scheduling Algorithms for Wireless Data Dissemination, ed. by Y. Pan, Y. Xiao. *Design and Analysis of Wireless Networks: Wireless Network and Mobile Computing*, vol. 1 (Nova Science Publisher, 2005)