Data Dissemination and Broadcast Disks

R. K. Ghosh

Jan-Feb, 2002

4.1 Introduction

Data management is the biggest challenge in developing truly mobile applications. Mobile devices are resource poor. This prevents a mobile terminal even 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 sattelite services provide a mobile terminal capabilities to access data/information anywhere at any time. So, an application may be organized as a set of synchronized activities that requires very little computation at mobile unit, but operates on globally available resources. This would allow a mobile user to transact both official and commercial transactions. The scenario of *mobile* or *nomadic* computing presented above poses the greatest challenge to database community since the adaptation of the concept of object model in database management. The problems the arise are how the mobility of user is going to affect the storing of data, delivery of data, processing of queries and processing of transactions. Apart from the user mobility there are many other characteristics of mobile computing which introduce sufficient complexities to data management for mobile computing, these include following, among others:

- How to securely broadcast queries over wireless data network?
- What is the influence of limited battery life on data access from a mobile device?
- What is the appropriate data delivery model for lossy, low bandwidth wireless links?

The techniques for data management in mobile distributed system are designed to address the following four main issues.

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

Mobility is not confined only to mobile networks. It also arises in the fixed network when computation is relocated from one node to another. A Remote

Procedure Call (RPC) is an example of control mobility in a fixed network. In a RPC, a thread started from from one node executes on another. From the point of view of database community, the interest on mobility is how to make the data highly available in spite of user mobility. Therefore, the issues those needs to be addressed are replication, and migration.

Wireless communication links are not only low bandwidth but depends on many sources of interferences. There is essentially no practical upper bound to communication delay over potentially lossy wireless channels. In order to treat the causes of fluctuation in connections in a uniform manner, disconnection is assumed to include various degrees of disconnections ranging from total disconnection to weak connection. However, disconnection may not mean *failure*. Failure is unanticipated whereas the disconnection is usually anticipated. So disconnection sometimes is considered as a planned failure. However, sometimes an *unplanned* disconnection may occur due to interferences in wireless channel. Mobiles may reconnect after a short unplanned disconnection. Therefore, it even unplanned disconnection can not be considered as a failure.

A straightforward way to retrieve data from the repositories or the servers is to send explicit queries over the uplink channel. Typically, the capacity of uplink channel is much lower than downlink channel. Since there is no control on the number of clients accessing a wireless channel, a large number of queries may get queued on the uplink even if each client generate only one or two queries. To avoid congestion build ups, instead of on-demand access, push based data delivery normally preferred.

Wireless medium can not exercise any control over the number of users. Furthermore, there is also no control on the number of active mobile users at a time in a specific cellular region. Thus, the number of active users may vary from a few hundreds in lean time to hundreds of thousand at peak time. Data service should not only be able to handle large variations in the number of active mobiles, but also should not cause any degradation in quality. This problem is referred to as scale. The requirements of data delivery and access modes are based on the challenges thrown by both:

- Scale factor, and
- Fluctuations in connectivity.

Mobility related data alone could far exceed the complexity of any conventional large database. It comprises of locating a mobile object, addressing, routing, replicating, etc. This aspect of data management have already been discussed under mobility management in chapter.So, it is not our concern here. The mobility related data of the above kind is referred to as *global* data management. Global data management is handled at the network level.

Local data management deals with the problem at the end user level or mobile terminal level. It includes energy efficient data access, managing range of disconnections, and query processing. Many of these problems are also relevant to data management in fixed network. But the scope of new research in the area is primarily due to the unique physical characteristics of mobile terminals. Important among these that we wish to address here are:

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

These constraints severely restrict transfer of large volume data. As noted eleswhere, in a cellular based network the number of active mobile users can not 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. Limited operating 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.

4.2 Data Delivery Models

The applications over the Internet such as e-entertainments, e-commerce, e-business, etc., lead to simultaneous accesses by millions of clients. Consequently, even with a large bandwidth, the information services are deteriorates instead of improving. This leads to a situation where "*plenty is not so plenty*". The main reason is that almost all applications use HTTP, a fundamentally unsuitable protocol, for data transfer. HTTP is a request–response type protocol. So, when large number of simultaneous requests is generated by the users it chokes the available bandwidth both at both at the server and the client ends. The situation is compounded by the users who continue to generate (poll) more requests for same data when they do not receive the response within expected time. These repeated requests cost not only in terms of network bandwidth but require additional the server cycles. Thus, request-response type protocol can not scale up, as a server gets quickly saturated; and the bandwidth is eaten away by unnecessary repetition of client requests. In contrast, suppose a data dissemination protocol is employed for transfer of data from the server to the clients wherein a server initiates the transfer of data as well as manages the propagation of updates to previously circulated data. The clients listen to the broadcast by the server and capture data needed for their respective applications. This approach to data transfer has a clear advantage over request-response protocol. It can easily scale up easily and independent of the number of clients.

4.2.1 Types of Data Delivery Models

The application running at client is the consumer of data and the server is the producer. In other words, data/information are generated at the servers. So, the servers are repositories of data/information that the client applications would require from time to time. Therefore, the server has to deliver appropriate data which the clients require for the applications to run. There are broadly two data delivery models:

- Client initiated data delivery, and
- Server initiated data delivery.

Client initiated data deliver is essentially *pull based*. Client sends an explicit request to a server. The responsibility of fetching data rests on the client.

When a server takes the responsibility of transferring data, then transfer takes place in anticipation of a future access. No explicit request exists from any client. However, data requirements of client population assesed in a global context. A client must be alert during the time the needed data for its application gets transferred over a channel by 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 want to view a TV program. The channel caters to an expected general viewing pattern by considering monthly ratings of the serials and other shows. TV transmission, thus, may not necessarily satisfy specific individual tastes. A similar situation is witnessed in push-based data transfer. Since the information is not explicitly sought for, but broadcast by a server to a all or a group of clients. This method of data transfer is referred to as data dissemination.

Figure 4.1 illustrates the data transfers as applicable to the two data delivery models discussed above. The top half of the figure represents pull

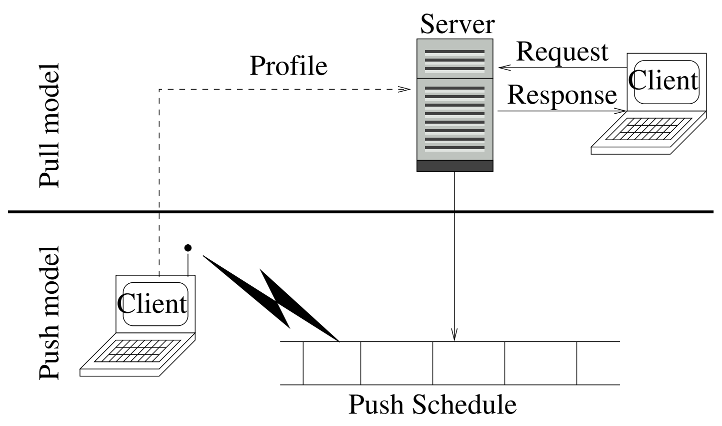


Figure 4.1: Data delivery models

based data transfer. In the pull-based data transfer occurs in a *request-response cycle*. The bottom half of the figure 4.1 represents push based data transfer. In a push based transfer, data transfer occurs on a *broadcast channel*. The server must have and idea of data that should be pushed. So, there should be a back channel through which profile information of clients can be collected as indicated in the figure 4.1. 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.

Before proceeding further, let us examine the difficulties associated with pull-based model in the context of an environment where the server has to serve a huge number of clients which are at times unidentified.

4.3 Drawbacks of Pull-Based Model

In a wired network all the computers share same physical medium (wire) for data transfer. Therefore, data transmission capacities in both uplink and downlink directions are the same. As far as wireless network is concerned there is an asymmetry in capacity of links in two direction. The variation in capacities between uplink (client to server) and downlink (server to client). The asymmetry in link capacities in case of a few recent examples N/W technologies have been been cited in the table 4.1 below. Even, otherwise, asymmetry can occur even in a symmetric client server setup. The anomalies

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

Table 4.1: Asymmetry in link capacities of wireless networks.

in bandwidths occurs as

- Either physically different media are used for uplink and 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 downlink and 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. The cause of service load asymmetry can be due to

- Either due to client to server ratio being high,
- Or due to the environment where updates are too frequent and clients continually poll for new data.

In some application environments data volume asymmetry arise due to volume of data transmitted in each direction. For example, in information retrieval type application just a few URLs (a mouse key click) results in a huge document to be downloaded. So, the requirements for uplink capacity is much smaller than downlink. Typically wireless connectivity may offer only uni-directional connections. So, the clients may be unable to connect also 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 similiar to 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 systm is that the client must have a back channel to request for the data. The back channel will eat away bandwdith available for data transfer. In a typical environment where clients are mobile back channel congestion can be real bottleneck for data transfer.

The availability as well as the use of the back channel for clients is restricted because of security reasons or the clients' battery problem. The saturation of server by number of requests from clients can be another sever problem. If the *request rate* is greater than the *service rate* eventually the server saturation will occur. However, in traditional applications, it can be controlled.

In summary the scale of mobile computing environment seems to be the single most factor causing all the drawbacks of pull based data delivery model.

4.4 Advantages of Push Based Model

The drawbacks of pull model can be considered as advantageous to argue a case for push-based model. Specially, when the clients are mobile and number of clients can not be controlled. So, only a server centric approach to transfer of data looks feasible. The situation is similar to case of information flows from the executives to the sub-ordinates in an business or government organization. The notices and office orders can only be generated by the executives; and these are brought to knowledge of the sub-ordinates by placing them on a bulletin board. Individually a sub-ordinate may not get a copy of the order. Technically push-based data transfer is more of a *data dissemination* model than a data delivery model.

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

- *Efficiency*: Data transfer is needed only when updates arrive at the server, or when new data are created. Obviously, a client is not expected to know when 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 data transfer is initiated by the server when some updates are made or some new data get created. So the utilization of bandwidth is exclusively for downstream link. So, requirement for bandwidth is low. Furthermore, the utilization of available bandwidth can be done optimally by the server.

The sequence of data transmitted by a push-base model is called a *push program* or a *schedule*. If the transmission is over a broadcast channel, then the sequence is called a *broadcast program*. The broadcast schedule can be classified as follow [5, 4].

- *Periodic*: In periodic transmission, the schedule is repeated in fixed intervals of time. The schedule of a periodic transmission is designed on the basis of anticipated polling patterns of the clients. So the interval of time between two consecutive instances of the arrivals of same data item is fixed.

- *Aperiodic*: The transmission in aperiodic schedule takes place when data updates are received or new data are created. There is no fixed interval of time between two consecutive arrivals of same data item.

- *Point-to-point*: The data delivered on downlink channel to a single client in the point-to-point data transfer model. It is equivalent to unicast. However, the term *unicast* refers to a communication pattern rather than actual physical communication link. In a non point-to-point broadcast medium *unicast* refers to the case where message is meant for a single client but visible to all.

- *One-to-many*: The transmission can be either a broadcast or a multicast. Broadcast means any unidentified client which can tune to the broadcast channel also receives the transmission, whereas in multicast, a group of identified clients receive the transmission.

4.5 A Taxonomy of Data Delivery Models

In the context client-server paradigm the basic difference between data delivery models lies with 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 system or request/response kind of data delivery model. Periodic pull arises when a client uses polling to obtain data it is interested in. It uses a regular schedule to request for data. Polling is useful in application such as remote sensing where client can spend sometime to repeatedly probe for the arrival of data in periodic interval of time. The data is not critical for immediate running of application. Thus, when the transfer is initiated by the client then it can be one of the following:

(i) *Request/response*: This is found in traditional scheme 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. But for the client who requested the data it appears as point-to-point, while other clients can snoop on the link and get data they have not explicitly requested.

(ii) *Polling*: In some application such as remote sensing or control applications, a system may periodically send request to other 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 Stock prices. This data delivery model is useful in situation where clients may not be available always. Since the delivery is for unidentified clients, the 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 – load asymmetry – generated by the clients or a large number of clients – high client to server ratio – are interested for what is being pushed. Aperiodic push is essentially publish and subscribe type 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 other cell. The push-based model is exposed to snooping and also the client can miss it out if it is not listening or can not determine the exact time for tuning to downstream channel when required data is being broadcast. Therefore, such a data delivery model could considered as more appropriate to the situation where what is sent out

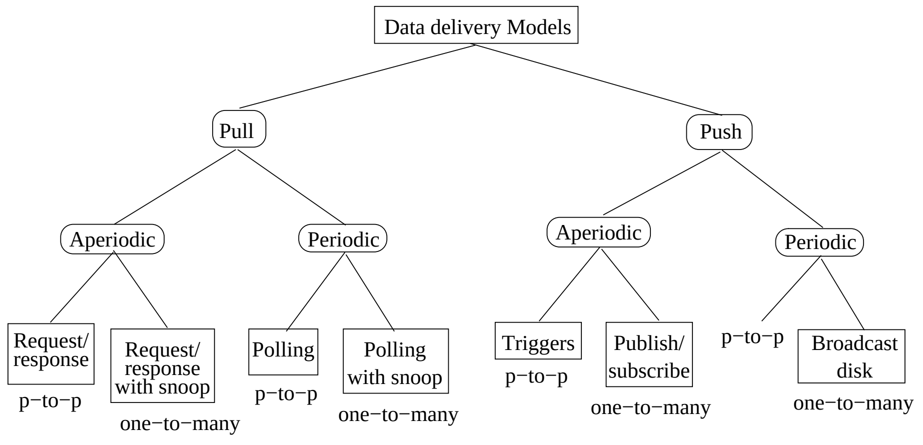


Figure 4.2: Taxonomy of Data Transfer

is not critical to immediate running of the application. But the push based transfer uses downstream channel more effectively. In summary, push based data delivery can categorized as follows.

(i) *Publish/Subscribe [6]* type data dissemination technique. Data flow is initiated by the server and is aperiodic. Typically one-to-many link is used to transfer data.

(ii) *Broadcast disks [1]*: It is a periodic push. 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 searched 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 4.2 provides a broad but 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 can not scale up easily when the number of clients becomes large. In one-to-many communication data is sent to a number of clients. The 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 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.

4.6 Broadcast Disk

From the clients perspective, periodic push is similar to accessing a secondary storage. Therefore, the data delivery should be organized to give best performance to the clients as a local disk would. If every item in a broadcast schedule appears only once then in the worst case, the client has to wait for one broadcast period from the time its starts listening to broadcast channel. It is rarely the case that all items are accessed equally frequently. Generally, the access tends to be skewed to a few *hot spots*. So it makes sense to capture the pattern of access in a broadcast program. *Broadcast disk* is a paradigm for organizing the structure of a periodic broadcast program.

4.6.1 Flat Periodic Broadcast Model

A flat broadcast program is shown in figure 4.3 can be considered as a logical disk spinning at a speed of one spin per broadcast period. The flat program is a degenerate case of generic broadcast disk model, where a single large disk spins at a speed of once per broadcast period.

4.6.2 Skewed Periodic Broadcast

Consider the case wherein 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 depending on the highest to the lowest frequencies of respective accesses. That is most frequently accessed data is placed on the fastest disk. The least frequently data accessed is placed on the slowest disk.

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

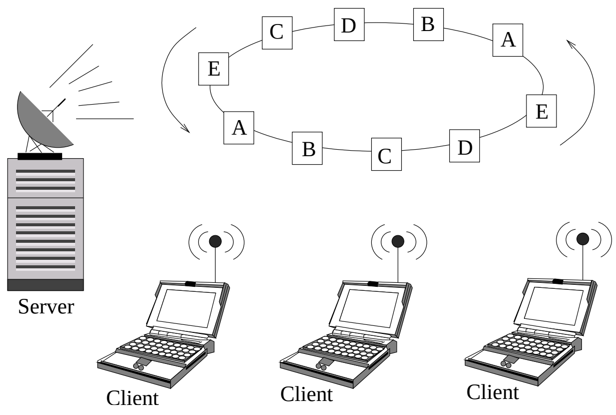


Figure 4.3: A flat broadcast disk

Otherwise it is dynamic. In the case of dynamic periodic broadcast, the period of broadcast may also vary.

Another way to classify the periodic broadcast could be on the basis of the inter arrival time of two consecutive instances of the same data item on broadcast channel. The broadcast is called *regular* if the inter arrival time of any two consecutive instances of the same data item is fixed. If there is variance in inter-arrival times, then the broadcast is *irregular*.

4.6.3 Properties of Broadcast Programs

From a more abstract point of view, a broadcast program visualized as an attempt to generate a bandwidth allocation scheme. Given the access probability of each client, a broadcast program determines the optimal percentage of bandwidth that should be allocated for each item. In absence of cache at the clients, the optimal bandwidth for an item is known to be proportional to square root of its access probability [2]. The broadcast program can then be generated randomly according to this bandwidth allocation principle such that average inter-arrival time between two instances of same item matches the needs of the client population. But the random broadcast will not be optimal in terms of minimizing the expected delay due to the variance in the inter-arrival times.

For example [1], consider the following three different broadcast programs involving three pages A, B, and C shown in figure 4.4. The performance

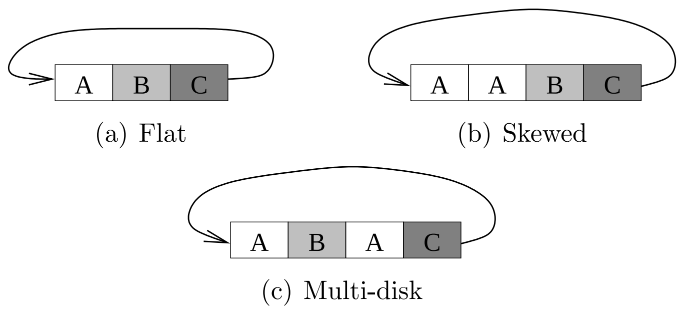


Figure 4.4: Broadcast programs [1].

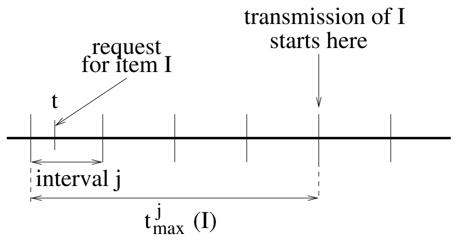


Figure 4.5: Formulation of maximum waiting time.

characteristic of third program shown in figure 4.4(c) is identical to the case that A is stored on a single-page disk spinning twice as fast as the two-page disk storing B and C. In case of the third program, the waiting time for accessing page, say A, is either 0 page or 1 pages, assuming that the requirement coincides with 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 the probability of access for each to be 1/3, the total wait (1/3)(0.5+1.5+1.5)=7/6 page. In reality the requirement for a page coinciding with page boundary broadcast is low. So adding 1/2 page to total wait, we have the delay as $5/3 = 1.67$ pages.

Indeed, it is possible to derive exact expression for the expected delay an item I from the point when the request is made. Let the need for I arise at some point of time t falling in the interval j as shown in figure 4.5. If I is scheduled to arrive in a future interval, then the maximum waiting time will the interval of time from the start 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, $t \in [0, 1)$. Therefore, 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, expected delays for item A in arrival skewed and regular broadcast types can be computed as follows:

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

Therefore, the expected delays for 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:

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

The table below, shows the computed delays for page requests corresponding to the client access probability distribution and the three different broadcast programs.

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

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 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.

4.6.4 Advantages of Multi-Disk Program

Multi-disk performs better (results in less delay) than the skewed program. This problem can be traced to 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 source S. The number of buses to destination D_1 is more than those for other destination. But all the buses for destination D_1 are clustered in a small window of time w in a day, whereas buses to other destination are staggered in more or less equal intervals over the day. Under this scenario, if a person is unable to be reach S within the interval w, can not get to destination D_1 for the entire day. However, the probability of getting a bus to destinations D_1 and D_2 is higher. This happens due the fact that the buses to other destinations are not clustered.

The probability of a request arriving during the time interval is directly proportional to the length of the interval. Consequently, if the inter-arrival rate (broadcast rate) of a page is not fixed, then the expected delay increases. When the inter-arrival rate of page is fixed, the expected delay to satisfy the requirements 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 can not go into *doze* mode to reduce the power consumption, because the time of arrival of the requested page can not 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 update semantics at the client. Therefore, some of the desirable features of a broadcast program are:

- The inter-arrival time of copies of a data item should be fixed.
- There should be a well-defined unit of broadcast after which it should repeat (periodicity). This helps to define the update semantics.

- It should use as much bandwidth as possible subject to above two constraints.

4.6.5 Algorithm for Broadcast Program

A simple algorithm for allocation of bandwidth for a periodic push based broadcast is as follows:

Step 1 Order the pages of database from *hottest* to *coldest.*

Step 2 Partition the list of pages into multiple ranges where each range containing the pages with similar access probabilities. Each range constitutes a logical broadcast disk. Let there be N_{disks} disks $D_1, \ldots, D_{N_{disk}}$.

Step 3 Choose the relative frequency of broadcast f_i for each disk D_i. The restriction is f_i are integers.

So, if two disks D_1, D_2 having respective frequencies $f_1 = 3$, $f_2 = 2$, then during the time the pages in D_1 appear 3 times, the pages of D_2 appears 2 times in the broadcast.

Step 4 Split each disk into number of smaller units Each unit is called a *chunk*. Let T_{max} be the LCM of relative frequencies. Split each disk into $T_i = T_{max}/f_i$ chunks.

For example, C_{ij} is the jth chunk of disk D_i, where $j = 1, 2, \ldots, T_i$.

Step 5 Create the broadcast program interleaving the chunks of each disk as follows:

```
for $(i = 1; i \leq T_{max}, i++)$
    for $(j = 1; j \leq N_{disks}, j++)$
        Broadcast chunk $C_{j,(i \textbf{ mod } T_j)}$;
```

Example Consider a three disk program in which pages of D_1 to be broadcast 2 times as frequently as D_2 and 4 times as frequently as D_3. That is, the frequencies are: $f_1 = 4, f_2 = 2$ and $f_3 = 1$. $T_{max} = 4$. Splitting disk D_1 in chunks results one chunk C_{11}. The disk D_2 is partitions into two chunks

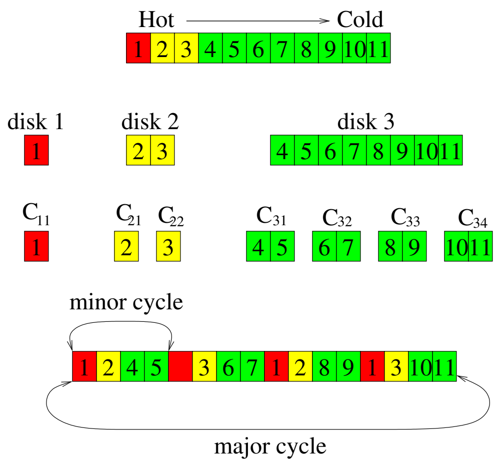


Figure 4.6: Bandwidth Allocation by Broadcast Algorithm

C_{21}, C_{22}, and D_3 gets split into four chunks $C_{31}, C_{32}, C_{33}, C_{34}$. Note that chunk size can be variable across disks. Figure 4.6 illustrate the broadcast program generated by the applying the above algorithm. As shown in the figure 4.6 there is a minor cycle consisting of three chunks – one from each disks D_1, D_2, D_3. The number of minor cycles, in a major cycle, is $T_{max} = 4$. Each minor cycle consists of 4 pages, one each from D_1 and D_2 and two from D_3. So, the major cycle consists of 16 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.

4.6.6 Parameters for Tuning Disk Model

Three main parameters to be tuned to particular access probabilities 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 a page to the fastest disk will add 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.

4.6.7 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 essentially means data items traverse the levels of disk hierarchy. But this would influence the client side caching strategy for pre-fetching (hoarding).

- *Disk structure*: The hierarchy of disks itself may be changed. For example, the ratios of disk speeds can be modified. An entire disk can be removed or added.

- *Disk contents*: The contents of disk may change. Some page may be added or removed.

- *Disk values*: Some pages on broadcast may change in value.

The dynamicity due to *Item placement*, *Disk structure* and *Disk contents* apply to both read-only as well as update type broadcasts. Whereas, *Disk value* introduces dynamicity that is applicable to update scenario only. *Item placement* or *Disk structure*, changes the relative frequencies and/or order of appearance of data items already being broadcast. The value of data item changes only when it is updated. Implying that in absence of update the first two modifications primarily affects the performance. The performance of client side caching gets affected as caching algorithms needs the information about latency of data items. This results in client side storage to be sub-optimally used. However, an advance notification may reduce the problem.

Dynamicity introduced due to *Disk Contents* does not influence the items that appear on broadcast. However, some items which appeared previously may disappear and some new items may appear. It can be viewed an extreme case of *Item Placement* – the items removed as having infinite latency to appear in another disk. The clients can cache an item before it disappears if an advance warning is provided. Or a client can even drop the item 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.

4.6.8 Unused or Empty Slots in Broadcast Disk

The broadcast schedule may consists of many unused slot, if it is not possible to partition a disk evenly into required number of chunks. However, those unused slots can be used for broadcast of some additional information such as indexes, updates, invalidations; or even extremely important additional pages. The number of disks should be small (normally in the order of 2 to 5), whereas the number of pages to be broadcast is substantially large. It means that the unused slots, if any, will be quite small in number. We can adjust the relative frequencies slightly to reduce the number of unused slots.

However, 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 placed on the broadcast channel. We divide the pages into 3 disks, D_1 having 1 page, D_2 having 2 pages and D_3 having 9 pages. D_1 is smallest and fastest, D_3 is largest and slowest. Let D_1 consists of 1 page D_2 consists of 2 pages and remaining 9 pages belong to D_3. Let the frequencies of D_1, D_2 and D_3 be 3, 2, and 1 respectively. That is during 1 spin of D_3, D_2 spins 2 times and D_1 spins 3 times. $T_{max} = LCM(3, 2, 1) = 6$. Therefore, according to the chunking algorithm pages of D_1 should be divided into $6/3 = 2$ chunks, D_2 pages should divided into $6/2 = 3$ chunks and those of D_3 should be divided into $6/1 = 6$ chunks. The number of pages per chunk in disk D_1, D_2 and D_3 are

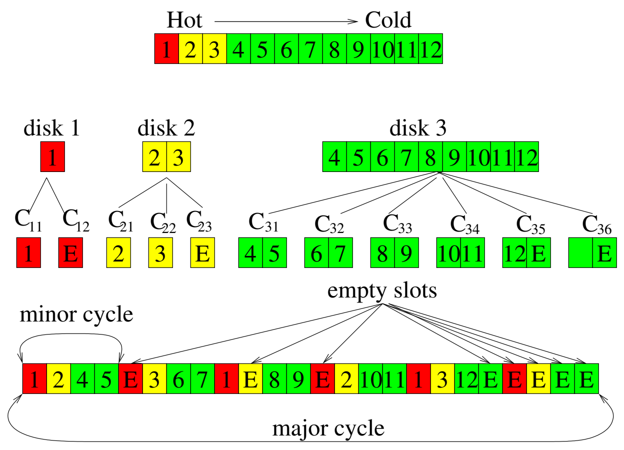

Hot
Cold
1 2 3 4 5 6 7 8 9 10 11 12
disk 1
disk 2
disk 3
C11 C12 C21 C22 C23 C31 C32 C33 C34 C35 C36
1 E 2 3 E 4 5 6 7 8 9 10 11 12 E E
empty slots
minor cycle
1 2 4 5 E 3 6 7 1 E 8 9 E 2 10 11 1 3 12 E E E E E
major cycle

Figure 4.7: Many unused slots may be generated by broadcast program.

$\lceil 1/2 \rceil = 1$, $\lceil 3/3 \rceil = 1$ and $\lceil 9/6 \rceil = 2$ respectively. So, the chunking requires a padding of 1 empty page each to be added to the pages in D_1 and D_2, and in the case of D_3, a padding of 3 empty pages will be needed. In other words, 5 empty pages are inserted as padding to generate integer number of pages for chunking of broadcast data. Figure 4.7 illustrates the process of chunking and depicts that 9 empty pages appear in broadcast cycle on a single major cycle consisting of 25 pages. So, 36% of the bandwidth is wasted.

4.6.9 Eliminating unused slot

Unused slots in the chunks in a broadcast schedule are like holes arising out of disk fragmentation. Three types of chunks exists in a schedule.

1. Fully used chunks: all the slots in such chunks 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 chunks are free.

The simple strategy to eliminate unused slots is to compact the holes in the broadcast schedule. However, this compacting technique applies only under some assumptions [3]. These assumptions are: (i) client population do not change, (ii) no update is allowed, (iii) clients do not employ pre-fetching or caching, (iv) clients do not use the uplink channel, (v) when a client switches to a public channel it can retrieve data pages without wait, (vi) a query result is represented by one page, (vii) A server uses only one channel for broadcast, (viii) broadcast is reliable, (ix) the length of each page is the same.

The compaction algorithm first needs to compute the indices of unused slots in wasted chunks of a disk. The computation of the indices turns out to be simple due to the assumptions presented above. But to concretise the computation formula a few notations becomes 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 the algorithm of section 4.6.5. According to this algorithm, the number of slots

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

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

Consider the example shown in figure 4.7. The number of slots in a chunk in three different disks as determined by the algorithm are:

$$NS_1 = \left\lceil \frac{1 \times 3}{6} \right\rceil = 1, NS_2 = \left\lceil \frac{2 \times 2}{6} \right\rceil = 1, \text{ and } NS_3 = \left\lceil \frac{8 \times 1}{6} \right\rceil = 2.$$

The execution of the original algorithm organizes the broadcast schedule into chunks consisting of 24 slots. Out of these 8 slots are empty. So, 24-8 = 16 slots have useful data. In the modified algorithm, a cut line is placed at the end of the slot 16. To the right of this cut line there are 8 slots, but only 3 of which have useful data. However, to the left the cut line, there are exactly 3 wasted slots. The modified algorithm moves the data from the right to the left of the cut line to empty slots preserving their order of occurrences. The movement of data is illustrated by figure 4.8. So the underlying strategy of the algorithm is compact the holes present in partially wasted chunks by moving data appearing to the right of the cut-line. It 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

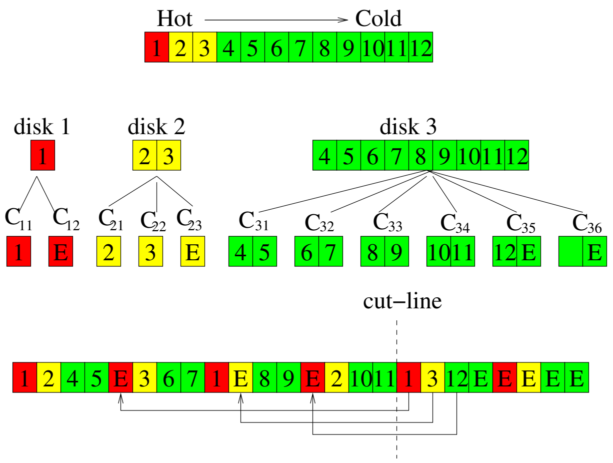


Figure 4.8: Eliminating unused slots.

wasted chunk (with some free slots). The number of fully wasted chunks in D_i is given by

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

Therefore, fully wasted chunks in disk D_i are C_{ij} where, $NC_i - FW_i + 1 \leq j \leq NC_i$.

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 just only one partially wasted chunk and it occurs only when

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

The index of the first empty slot in the partially wasted chunk C_{ij} is $j = NC_i - FW_i$. The total number of wasted slots in the 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 know how the empty slots are identified, the compaction algorithm becomes straightforward. The detailed steps of the modified algorithm [3] now appears below.

Step 1. Calculate the total slots (denoted as TS) and total wasted slots (denoted as TWS) in one major broadcast cycle generated by the original broadcast disk program provided in section 4.6.5

Step 2. Find out the cut-line (denoted as CL) which equals to $(TS - TWS)$.

Step 3. Find out the nonempty slots after the CL and record them in the array $Moved$.

Step 4. Let us use a sequence number (SN) to denote the sequence of those slots in a major cycle as 1,2,..., TS. Find out the corresponding E_{ijk} of an SN before the cut-line and record the corresponding E_{ijk} in an array $Broadcast$. If the corresponding E_{ijk} is empty then replace it with the data recorded in $Moved$.

Step 5. Broadcast the contents of the Broadcast array in sequence.

```
for $(i = 1; i \leq CL; i++)$
    Broadcast $Broadcast[i]$;
```

In step 1, the total number of slots (TS) 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 total number of wasted slots (TWS) in one major cycle is

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

Steps 3 and 4 constitute the core of the compaction technique. The data slots to right of cut-line are moved into a separate array $Moved$ by step 3. The slots from which data have been placed in $Moved$ array can now be declared as empty. Step 4 performs movement of data from $Moved$ array to empty slots before the cut-line.

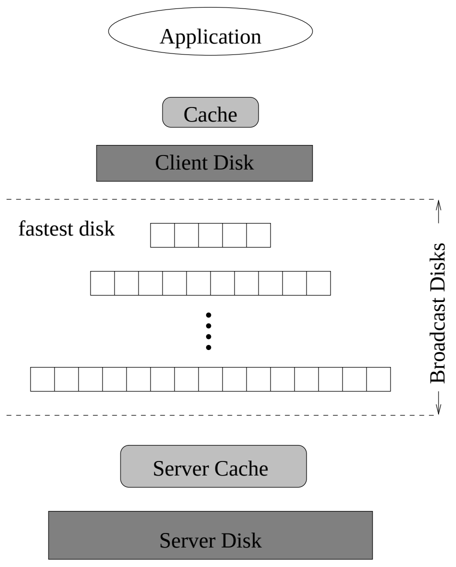


Figure 4.9: Memory hierarchy of broadcast disks

4.7 Memory Hierarchy

In a client server environment memory hierarchy consists of

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

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

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

- The fastest disk is top level and the slowest disk is bottom level.
- We can view this combination hierarchy in broadcast system as shown in the figure 4.9.

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. It is possible to create arbitrary fine-grained 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 level of memory hierarchy. In broadcast system this is true only for average cost. The instantaneous cost can vary from zero to the broadcast period. It may be cheaper to use back-channel for fetching the required data than to wait.

4.8 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 extreme cases, the average requirement pattern may completely mismatch the requirements of some clients while it may perfectly match the requirements of 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 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 data is accessed for the first time. So, 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. 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 wastage of cache resources if the data is not eventually accessed.

The two major issues need to be addressed before employing any caching strategy:

1. Victim selection,
2. Lowest utility caching.

Replacing a cached page by a new page is the primary concern in a caching strategy. It is critical to performance. For example, immediate references to the victim page may occur after it has been replaced while the new page may not be referenced at all. This may occur due locality associated with address references. So, the page having the lowest utility at that moment should be evicted.

In case of a dissemination based information system, the burden of data transfer rests with 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 study of the problem of client side caching with relative to pull 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 a push-based system the absence of a back-channel forces a client experiencing a page fault has to keep listening until the required page arrives on the broadcast channel. In wireless communication, the nature of the communication medium (air) is sequential. Therefore, no random access is possible. Consequently, the client must wait to access the data in the order it is sent by the server. The access cost is, therefore, non-uniform. The data is not equidistant from the client's cache. It happens due to the multi-disk framework. In traditional system the cost is uniform.

4.8.1 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 the best if it based on access probabilities. The access time remains uniform for pulling 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 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.

4.8.2 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 generate 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 figure 4.10. The client side cache filters accesses by storing certain pages (not necessarily hot pages) such that filtered access

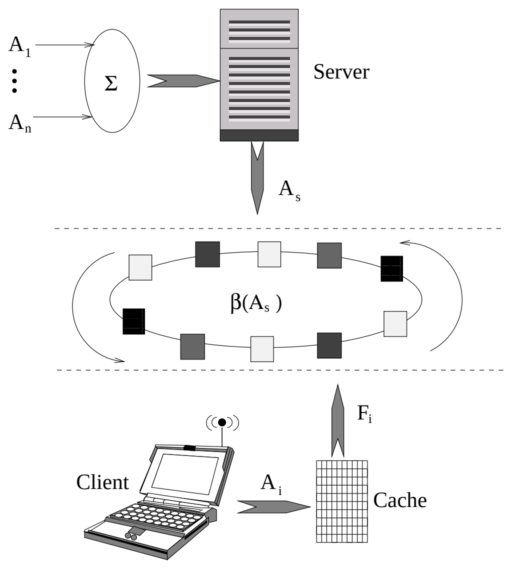
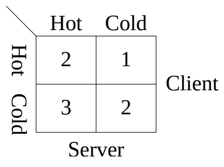


Figure 4.10: A view of client side cache

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.

4.8.3 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



Server \ Client	Hot	Cold
Hot	2	1
Cold	3	2

Figure 4.11: Cost of prefetching

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 figure 4.11 below 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 approprtiate cost based caching should be developed.

4.8.4 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 10. 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 client is more than that of a, from the server prospective b is hotter than a, so the server broadcasts page b more frequently than it does in the case of page a. Therefore, a needs to be cached though its access probability is less than that of b. 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 page can not be measured or known beforehand, we may find an approximate measure of the probabilities. 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 that takes into account the broadcast frequency. It maintains the cache as a singly linked list. When a page is accessed it is moved to the top of the list. For each broadcast disk a separate linked list is maintained at the client. 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].

4.8.5 Pre-fetching Cost

The goal of pre-fetching is to improve the response time for client application. Pre-fetching is an optimistic caching strategy, where the pages are

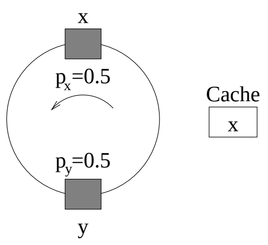


Figure 4.12: Illustrating pre-fetch cost

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 an additional burden on resources. But in a broadcast environment 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 degrees apart. There is a single cache slot at the client. The figure 4.12, 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.

4.9 Update Dissemination

In a client-server system update dissemination is a key issue concerning performance versus 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 application 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 the 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.

4.9.1 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 updates are initiated by the server. However, in absence of 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.

4.9.2 Data Consistency Models

The notion of data consistency is application dependent. For example, in database application, the consistency of data normally tied with transaction serializability. But many application may not require full serializability. So we need to arrive at some weaker forms of correctness. In a dissemination based system the notion of consistency is not fully understood till date. Therefore, the focus of the present discussion is on a broadcast disk environment where all updates are collected at the server, and the clients access the data in read-only fashion. The examples of such applications could be stock notification, weather report, etc. The models of data consistency normally employed are:

- *Latest value*: The client must access the recent most value. The clients perform no caching and the server always broadcast updated value. There is no serializability – no notion of mutual consistency among data items.

- *Quasi-caching*: Defined per-client basis using a constraint that specify the tolerance of slackness compared to *Latest value*. Client can use cached data items and the server may disseminate updates more lazily.

- *Periodic*: Data values change only at specified intervals. In a broadcast environment such intervals can be either the minor or the major cycles of a broadcast. If a client caches the values of broadcast data then it is guaranteed the data will remain valid for the remainder of the period in which the data is read.

- *Serializability*: In the context of transaction this the notion of consistency. But periodic model can implement serializability using optimistic concurrency control at clients and having the server broadcast the update logs.
- *Opportunistic*: For some application or under certain conditions 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.

Bibliography

[1] Acharya, S., Franklin, M., and Zdonik, S. Dissemination-based data delivery using broadcast disks. *Personal Communications, IEEE 2*, 6 (2001), 50–60.

[2] Ammar, M., and Wong, J. The design of teletext broadcast cycles. *Performance Evaluation 5*, 4 (1985), 235–242.

[3] Chang, Y.-I., and Yang, C.-N. A complementary approach to data broadcasting in mobile information systems. *Data & Knowledge Engineering 40*, 2 (2002), 181 – 194.

[4] Franklin, M., and Zdonik, S. A framework for scalable dissemination-based systems. In *International Conference on Object Oriented Programming Languages and Systems* (1997), OOPSLA '97, pp. 94–105.

[5] Franklin, M. J., and Zdonik, S. Z. Dissemination-based information systems. *IEEE Bulletin of the Technical Committee on Data Engineering 19*, 3 (1996), 20–30.

[6] Huang, Y., and Garcia-Molina, H. Publish/subscribe tree construction in wireless ad-hoc networks. In *Mobile Data Management (MDM'03)* (2003), pp. 122–140.