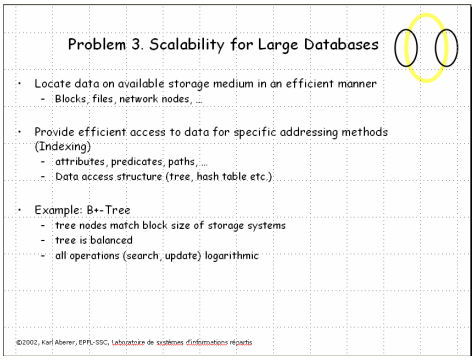
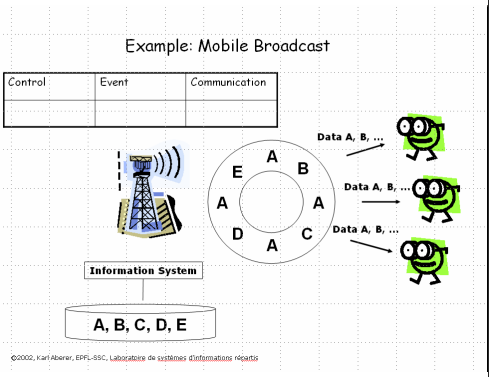
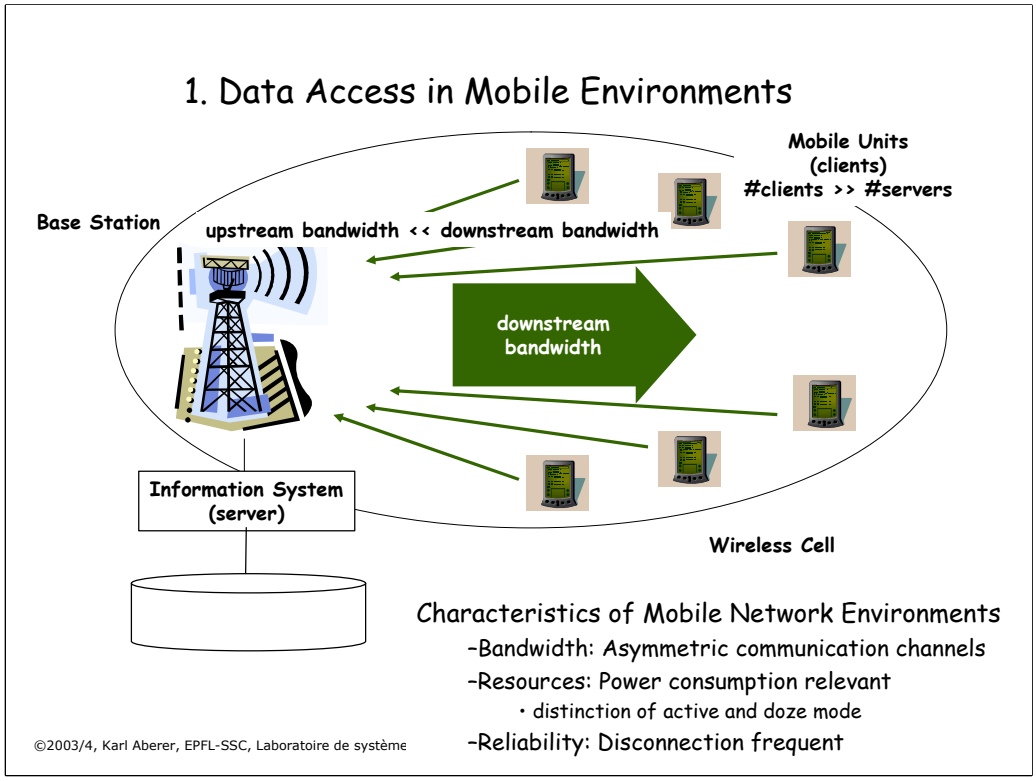
Distributed Data Management
Part 2 - Data Broadcasting in Mobile Networks

Today's Questions

1. Data Access in Mobile Environments
2. Scheduling Broadcast Disks
3. Client Caching
4. Indexing Broadcasts Disks







With the availability of mobile communication networks we have to re-consider the problem of data access to servers taking into account the changed physical environment. A typical situation is having a base station that covers a wireless cell, in which many mobile clients communicate with the base station. The base station provides access to information systems using a fixed network infrastructure. The characteristics of the environment can be fairly different depending on the underlying networking technology (e.g. GSM phones, WLAN or satellite). The problem we will consider in the following is where clients want to gain access to data that is accessible through the base station.

With respect to the network the following characteristics make the problem of data access different from the access through conventional, fixed networks.

(1) The network is (or better can be) asymmetric, such that the bandwidth for sending data from the server to (all) clients (downstream) in the cell is much higher than vice versa (upstream). On the one extreme it is possible that clients have no possibility to send data to the server (e.g. a satellite), on the other extreme they bandwidths can be the same (e.g. a WLAN). Still , in the latter case all mobile units would compete for the same bandwidth.

(2) A second important issue is power consumption. Energy is a scarce resource at the mobile unit, and often active communication is by orders of magnitude more energy-consuming than switching to a doze mode. So techniques for minimizing the active time of a mobile unit are important.

(3) Frequent disconnections can occur, e.g. when mobile units change a wireless cell. This poses problems in terms of data consistency, e.g. when at that moment an update transaction is executed and is aborted. Basic assumptions on transactions in fixed networks, e.g. having a reliable communication channel, no longer hold. We will not explore this third issue here in the following.

Another typical characteristics in mobile environments is that often the number of clients is specifically high as compared to the number of servers.

Exploiting Asymmetry: Data Broadcast

- Downstream bandwidth B (can be) high compared to upstream bandwidth U
  - e.g. satellite downlink with 400Kb/s
  - sometimes no upstream connection available
- Numbers of clients C (can be) high compared to number of (possible) different requests R
  - e.g. stock quotes, tickers, newsletters etc.
  - therefore downstream capacity can be shared and downstream capacity per client increases since B/R > B/C
- If B sufficiently large then B/R > U *
  - then it is more efficient to simply immediately send all answers to potential requests over the downstream broadcast channel than to wait for individual requests of clients: data broadcast or data push
  - capacity independent of number of clients C

* we assume that requests and responses consume same bandwidth

Asymmetry in a communication environment means that the down-stream capacity is substantially larger than the upstream capacity. The capacity is however not only related to the bandwidth of the communication link, which can be asymmetric in a mobile environment, but also to the distribution of data that has to be transmitted over the communication link. In general, with a growing number of clients it is likely that the number of different data items that need to be transmitted over the communication link grows slower than the client population. For example, in a news ticker scenario it is very well possible that the number of news items is much lower than the number of clients. This situation in fact can occur also in fixed networks, and therefore some of the techniques we will be looking at in the following apply there as well.

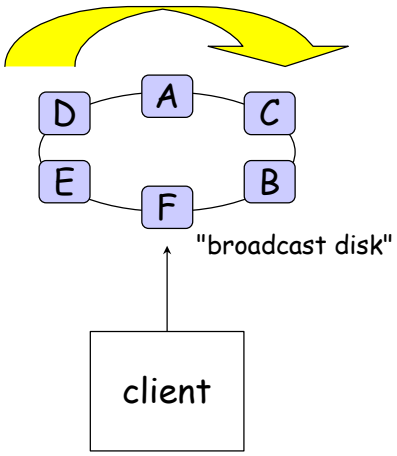
At this point we can make two observations:

(1) if the number of clients C increases, the number of different requests R may increase slower. Thus the downstream capacity per request will be higher than the downstream capacity per client. In other words if we can share the downstream capacity for multiple clients when they have the same request the downstream capacity is higher in total.

(2) if D is reasonably large, R is not too large and the upstream capacity U is low, then it can occur the downstream capacity per request becomes large than the upstream capacity. If we assume that a request and response in a client-server interaction consume approximately the same bandwidth (which is acceptable for data requests), this implies it makes no more sense to issue requests from clients, but the server simply sends all possible answers over a broadcast channel. Since D/R is independent of the number of clients C this approach scales arbitrarily well in the number of clients. If U=0, i.e. clients have no upstream capability, broadcast is the only possible approach in any case.

Periodic Broadcast

- Periodically broadcast all data items
  - The network appears to a client like a disk
- Problem: Accessing the "broadcast disk" the client would like to minimize
  - Latency (Expected Delay): time a client waits till a required data item appears
  - Tuning time: time a client spends actively listening to the network (power saving)
- Assumptions
  - Client access patterns do not change
  - Clients make no use of upstream capability
    - no feedback to the server
  - Data is read-only
  - Data items are equally sized (pages) and are identified by a key



Periodic broadcast is a simple way to push data to the clients. The server selects the data items to be disseminated, defines an order on them and sends them in this order periodically over the data channel. We assume that the data items of the same size and that they are identified by a key (in a relational database these could be fragments of relations, but as well pages of the storage system, depending on the level of abstraction that is used to provide the data to the client, in an XML database these could be XML documents). When the client receives this stream of data items it appears as if the whole database is stored on a disk with a peculiar access characteristics. In particular, the latency of this access can be very high as the "broadcast disk" does only support sequential access to the data.

An important problem in access to the broadcast is whether the client has to actively listen. Many mobile devices distinguish a doze mode from an active mode, which consumes by orders of magnitude less energy. In case the client has additional information on the time when interesting data items will occur on the broadcast disk, it has the possibility to switch to the doze mode in those times where no relevant data occurs.

In the following we will make some additional assumptions on the characteristics of the problem. In particular, we will assume that the problem is static, and that clients do not actively communicate with the server, in particular they cannot make updates to the database. Hybrid approaches where communication can occur in both directions have been investigated but are beyond the scope of this lecture.

Example

- TV Videotext
  - Database: e.g. 1000 text pages, all equal size (approx.)
  - Keys: page number
  - no upstream capability and read-only
  - access pattern fairly constant
- Latency: time to wait till page appears -> minimize
- Tuning Time: have to be on TV channel when page appears
- Capacity
  - B e.g. 500 pages /minute
  - then B/R 0.5 pages/minute

A very popular example of a broadcast medium is videotext, which exhibits many interesting characteristics of the problem. Actually we can observe also there the problem of "tuning time", although not related to power consumption, but related to the time we have to stay at a specific TV channel to obtain the page (assume that you are not interested in the current TV program). The numbers are somewhat arbitrary and just serve for illustration.

What Do You Think ?

- How can we optimized latency and tuning time for periodic data broadcast (e.g. videotext) ?

Summary

- How does a mobile environment differ from a fixed network, when accessing databases ?
- When does it make sense to broadcast data ?
- What are the criteria for optimizing access to a data broadcast ?

2. Scheduling Broadcast Disks

- Naive Approach: Flat organization
  - Set of all data objects is cyclically broadcasted
  - Expected delay = N/2, N number of data items
- Key Idea for improvement: Take into account the access frequencies to data objects when scheduling a broadcast
  - More frequently used items should appear also more frequently on the broadcast disk

Access frequencies

-D set of data items

-For each d_i $\hat{I}$ D the probability p_i that d_i will be accessed is given,

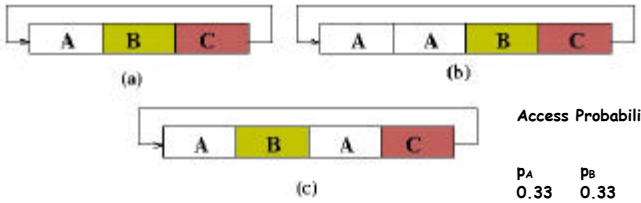
$$\sum_{d_i \in D} p_i = 1$$

The naïve approach to creating a data broadcast is to organize it in a flat fashion, i.e. each data item occurs exactly once. Then it is easy to see that if the broadcast contains N data items that the expected time to wait for a specific data item starting at some random time is N/2. In practice different data items are accessed with different frequency and therefore a possible optimization would be to include more frequently requested data items more often into the broadcast. In the following we introduce a method for doing this assuming that access frequencies are known.

Remark: you might have noticed that the Videotext system is doing something similar: main pages appear in general much faster than less popular ones.

Example

- Assume A is accessed more frequently: $p_A > p_B, p_C$



(a) flat broadcast disk
(b) skewed broadcast disk
(c) multilevel broadcast disk

Access Probability			Expected Delay (in broadcast units)		
p_A	p_B	p_C	Flat	Skewed	Multi-Disk
0.33	0.33	0.33	1.50	1.75	
0.50	0.25	0.25	1.50	1.63	
0.75	0.125	0.125	1.50	1.44	
0.90	0.05	0.05	1.50	1.33	
1.00	0.00	0.00	1.50	1.25	

Observations

- (b) is always worse than (c) in terms of average delay
- Expected delay depends on the ordering of data items on the broadcast disk
- In (b) and (c) access to B and C becomes slower
- A is on a broadcast disk which "spins twice as fast" as the disk on which B and C are stored: multilevel broadcast disk

This example illustrates that when replicating more popular data items in the data broadcast, the interleaving of data items can be organized in different ways. Assume that A is more popular than B and C and therefore it has been decided to include two copies of A into the broadcast. There exist two possibilities for doing this: either sending two A's consecutively (which results in a skewed broadcast) or to interleave A with the two other data items. The interleaved broadcast is also called a "multidisk" broadcast, since from the viewpoint of the client it appears, as if the broadcast consists of two different disks it can alternatively access, but the speed at which the "disk" containing A rotates is twice the speed at which the "disk" containing B and C rotates.

The table gives the calculated values for the expected delays for different combinations of access frequencies. Of how these are precisely calculated is shown on the next slide, but qualitatively it is no problem to understand the result. Some of the observations are:

-for the flat organization the expected delay (averaged over all data items !) does not change.

-the skewed organization is always worse than the multidisk configuration. So we see that the scheduling of data items in the data broadcast is not only a bandwidth allocation problem (then skewed and multidisk would be the same), but has to take into account ordering.

-The "break-even" point where the multidisk configuration starts to pay off is reached when A is accessed more than half of the time

What cannot be seen from the table is that in fact the access to B and C become slower in the multidisk configuration, and therefore for a uniform distribution of access frequencies the multidisk configuration is worse. This loss is however compensated increasingly by the fact that the more frequently requested data item A is accessed faster.

Computation of Expected Delay

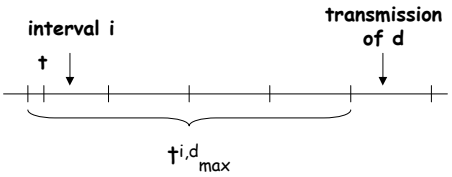
Expected delay for one item d

$$delay_d = \frac{1}{N}\sum_{i=1}^{N}\int_0^1 (t_{\max}^{i,d} - t)dt = \frac{1}{N}\sum_{i=1}^{N}(t_{\max}^{i,d} - \frac{1}{2})$$

N is the number of transmission intervals
Duration of one transmission interval is 1
t is the arrival time of the request
$t^{i,d}_{max}$ is the waiting time when request arrives at start of interval i
In the case transmission of d has already started wait for the next transmission

Expected delay for all items d_i with access probabilities p_i

$$delay_{tot} = \sum_{d_i \in D} p_i delay_{di}$$



The computation of the expected delay is not very complicated but requires some care. For a given data item d the delay is computed for each interval of the data broadcast separately. One assumes that if the request arrives within the interval i where the requested data item d lies, then one has to wait for the next transmission of this data item, as the complete interval is required to obtain the complete data item.

For example, for broadcast scheme (b) (skewed broadcast) the expected delay for a request for A is computed as follows:

Arrival of request in interval 1: $t^{i,d}_{max}$ = 1 (next interval), expected delay 0.5

Interval 2: $t^{i,d}_{max}$ = 3 (beginning of next broadcast), expected delay 2.5

Interval 3: $t^{i,d}_{max}$ = 2 (beginning of next broadcast), expected delay 1.5

Interval 4: $t^{i,d}_{max}$ = 1 (beginning of next broadcast), expected delay 0.5

Total: (0.5+2.5+1.5+0.5)/4=1,25

For B and C we have delay = 2

Therefore for equally distributed probabilities, we obtain an expected value of 1/3*1.25+1/3*2+1/3*2=7/4=1.75

Broadcast Disk Organisation

Theorem 1: The broadcast schedule with minimum overall mean access time results when the instances of each data item are equally spaced.

Theorem 2 (Square-Root Rule): Assuming that instances of each data item are equally spaced, minimum overall mean access time is achieved when the frequency f_i of each data item d_i is

$$f_i \propto \sqrt{p_i}$$

The overall mean access time is then

$$delay_{optimal} = \frac{1}{2}\left(\sum_{d_i \in D} \sqrt{p_i}\right)^2$$

Remark: Since equal spacing cannot necessarily be always obtained this gives a lower bound

For finding good broadcast disk organizations we will make use of two theoretical results.

The first generalizes our observation on skewed broadcasts, which are always worse than multidisks. It can be shown that the mean access time to a broadcast is minimized if ALL data items are distributed over the broadcast such that the space between them is constant. So any broadcast organization should try to achieve this property. One has to be observe however, that for concrete cases this property not necessarily can be achieved. For example, if we would have to distributed 2 data items A and 3 data items B, there exists no broadcast that allows equal spacing. So only approximations of schedules with equal spacing will be possible in general.

The second result applies in case data items are equally spaced. It says that then it is possible to determine the optimal frequencies with which data items have to occur in the data broadcast, provided that the access probabilities are known. The interesting observation is, that these two quantities are not proportional (as one might expect at the first moment !), but rather that the frequency in the broadcast has to be proportional to the root of the access probability. This means that the popular data items appear in general less frequently in relation to the number of accesses that are made to them.

Broadcast Scheduling Algorithm

- Find scheduling algorithm that creates
  - periodic broadcast
  - with fixed inter-arrival times for data items
  - optimized frequencies of data items
  - and uses under these constraints as much bandwidth as possible

- Algorithm Overview
  1. order data items according to frequencies and cluster data items with same (similar) frequencies: defines broadcast disks
  2. determine the optimal frequencies for the broadcast disks
  3. distribute the data items evenly spaced over the broadcast disk

Given the two results mentioned before the task is now to find a broadcast organization that satisfies the conditions on an optimal broadcast as closely as possible (we have seen already that we ill in general not be able to satisfy them exactly).

The general approach will consist of three main steps: first identify those data items that should occur with the same frequency in the broadcast, since they are accessed with the same (or similar) probabilities. This defines the different "disks" of the broadcast. Then based on the access probabilities to each of the disks optimal frequencies of occurrence are computed according to Theorem 2. In the last step a broadcast disk is configured such that the data items of each disk occur exactly with the (approximate) optimal frequency determined and that the data items are equally spaced.

Determining Optimal Frequencies

- Broadcast disks $D_1,\ldots,D_k$ with access probabilities to each data item of $p_1,\ldots,p_k$ have been defined, such that $p_1 > p_2 > \ldots > p_k$
- Then (f_{min} is freely chosen)

$$f_j = \left\lfloor f_{\min} \sqrt{\frac{p_j}{p_k}} \right\rfloor, j = 1,\ldots,k$$

- Example:
  - d1 with access probability 1/2
  - d2 and d3 with access probability 1/8
  - d4 – d11 with access probability 1/32
  - 3 broadcast disks D_1={d1}, D_2={d2,d3}, D_3={d4,d5,d6,d7,d8,d9,d10,d11}
- Optimal frequencies (f_{min}=1)

$$f_1 = \left\lfloor \sqrt{\frac{\frac{1}{2}}{\frac{1}{32}}} \right\rfloor = 4 \qquad f_2 = \left\lfloor \sqrt{\frac{\frac{1}{8}}{\frac{1}{32}}} \right\rfloor = 2 \qquad f_3 =$$

Defining the different disks and determining their corresponding access probabilities is trivial provided we have a set of data items with access probabilities given. This first step is illustrated in the example. The next step is to determine the optimal frequencies. Since the theorem only determines the proportion among the access probabilities and frequency of data items in the broadcast, we can choose one of the frequencies freely. For practical reasons, since we want to obtain integer frequency values we choose the frequency of the data item with smallest probability of access freely (fmin). From there all the other frequencies are determined due to Theorem 2. We have to multiply the minimal frequency with the square root of the proportion of the smallest access probability and the access probability for the disk for which we determine the frequency. Since the frequency must be an integer (remember: the frequency is the number of copies of the data item/disk in the broadcast), we take the next lower integer value. A sample computation of frequencies is given for our running example.

Broadcast Schedule Generation

- Problem: how to distribute data items evenly spaced with correct frequencies ?
- Example: d1,d1,d1,d1,d2,d2,d3,d3,d4,d5,d6,d7,d8,d9,d10,d11
- Idea :
  distribute D_1 (f_1=4)
  d1 d1 d1 d1
  interleave D_2 (f_2=2)
  d1 d2 d1 d3 d1 d2 d1 d3
  interleave D_3 (f_3=1)
  d1 d2 d4 d5 d1 d3 d6 d7 d1 d2 d8 d9 d1 d3 d10 d11
- This works since size(D_3)*f_3=8 and size(D_2)*f_2=4 are multiples of size(D_1)*f_1=4 !
- What if this is not the case ?

Now that the frequencies are known for each disk, we have to distribute the different disks according to the frequencies over the broadcast. This is in fact not a completely trivial problem. In our example we could start as indicated above by first distributing disk D1, then interleaving the data items from disk D2, by alternating the occurring multiple items, and then interleaving the data items from disk D3. The resulting broadcast satisfies all required properties, e.g. the data item from disk D1 appears 4 times (frequency 4) and all data items are equally spaced. However, this works only due to the specific structure of this example.

Broadcast Schedule Generation

- Partition each disk D_i into a number of smaller units (chunks) C_{ij} ,j=1,…,c_i

  $c_i = c_{max}/f_i$ and c_{max} = LCM(f_1,…,f_k) (least common multiple)

- If size(D_i) cannot be divided by c_i fill it up with data items to disk D_i* till size(D_i*) can be divided by c_i
  - The data for filling up can be frequently used data or other information such as indexes and updates
  - However, the number of disks will be small as compared to the number of pages, therefore few gaps occur in practice

- Then generate c_{max} copies of each chunk C_{ij} and distribute them evenly

- Consequence: for disk D_i size(D_i*) / c_i * c_{max} = size(D_i*) * f_i

- Therefore each data item in D_i appears exactly f_i times in the broadcast !

In case the frequencies are not in such a "nice relationship" as before (e.g. assume f1=2, f2=3,f3=5), we have to apply a "trick". The idea is to find partitions of the set of data items in each disks into chunks in a way that when later producing a constant number cmax of copies of each of the chunks all data items occur with the expected frequencies. Since the number of copies of chunks is constant it is then straightforward to generate a broadcast schedule by simply regularly distributing the cmax copies of the chunks over cmax bins.

In order to properly partition the disks into chunks we have to determine depending on the required frequency the number of chunks. This can be done in the described above, starting out from the least common multiple of all frequencies. The computation above also shows that by choosing the numbers of chunks in this way generates the desired frequencies for each disk.

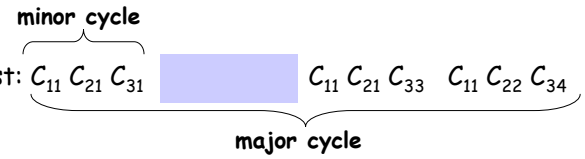
Of course not every disk can be precisely divided into the desired number of chunks. For example, taking the frequencies f1=2, f2=3,f3=5, c1 would be 15. Now if disk D1 contains a single data item only it can obviously not be divided into 15 pieces. Therefore, what is done, is to fill the disk up with other information (in this case 14 more data items) to make it divisible. In practice the problem is much less severe as it appears in this extreme case, since normally the number of data items on each disk will be large as compared to the frequencies values involved and thus only very few data items need to be added.

Broadcast Schedule Generation Algorithm

Broadcast Schedule Generation Algorithm (k disks)

```
broadcast = empty sequence;
for l=0 to c_max-1
        for i=1 to k
                j := l mod c_i + 1;
                broadcast := append(broadcast, C_ij)
```

- Example: $c_{max} = 4$, $c_1 = 1$, $c_2 = 2$, $c_3 = 4$
  Disks: D_1={d1}, D_2={d2,d3}, D_3={d4,d5,d6,d7,d8,d9,d10,d11}
  Chunks: C_{11}={d1}, C_{21}={d2}, C_{22}={d3}, C_{31}={d4,d5}, C_{32}={d6,d7}, C_{33}={d8,d9}, C_{34}={d10, d11}



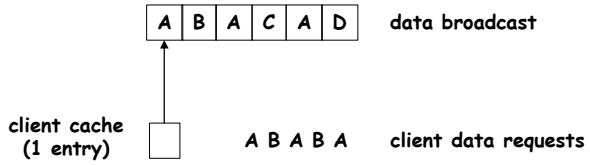
Once the chunks are determined it is fairly straightforward to distributed them over the cmax bins. A possible algorithm doing that is given above. Note that by distributing the chunks regularly over cmax bins, the broadcast schedule consists of two types of cycles a minor cycle, were periodically each disk is having some (possibly varying) data items, and a major cycle which is the whole broadcast schedule.

Summary

- Why is a flat broadcast disk organization not necessarily optimal ?
- What is the effect of ordering multiple data items in a broadcast disk on latency ?
- What is the square-root rule for broadcast disk organizations ?
- How are the frequencies of the major and minor cycle in a broadcast disk computed ?

3. Client Caching

- Broadcast schedules
  - based on average access probability: average over ALL clients
  - not necessarily optimal for a single client
- What can clients do individually ?
  - Caching: keep data items that they received
  - Caching strategy: when forced to replace (cache is full), replace those that are least likely to be needed in the future



What caching strategy should clients use ?

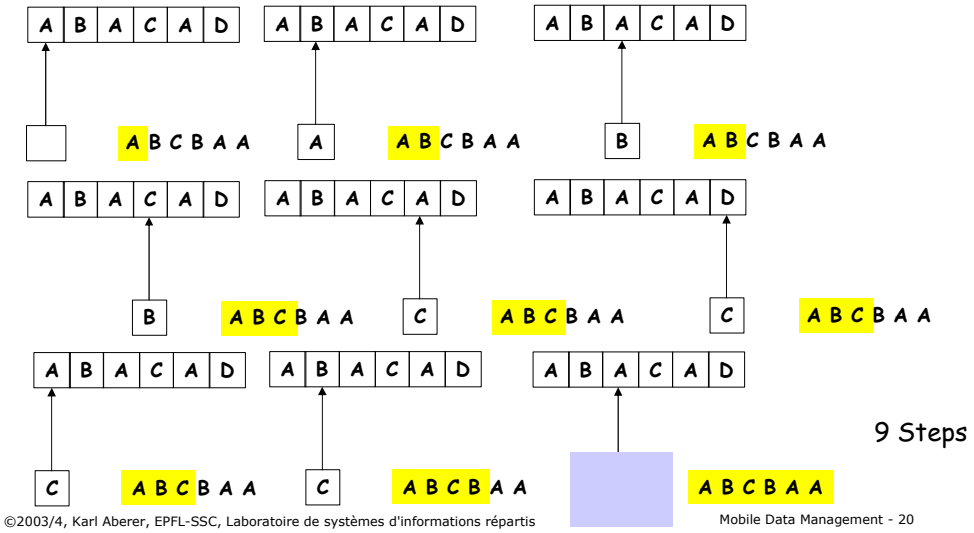
The scheduling of a broadcast is only optimal for the average behavior of clients, but not necessarily for a specific client with behavior that deviates from the average. Therefore clients can apply further optimizations to improve performance, in particular latency. A standard technique to that end is caching, as it is, for example, also used by Web browsers.

Caching is replicating and storing data that has already been used or requested by the application for later reuse. Caching is distinguished from replication, where data that not yet has been used/requested by the application is actively replicated.

On a mobile device the storage capacity is often limited. Therefore an important question is which data is to be kept in the cache, or, in case the cache is full, which data is to be evicted from the cache when a new data item arrives. Of course, the goal is to keep always those data items that are most likely to be used in the future. Predicting this requires to take into account any possible knowledge that can be gained on future accesses to data AND on future availability of data.

Strategy 1: Least Recently Used (LRU)

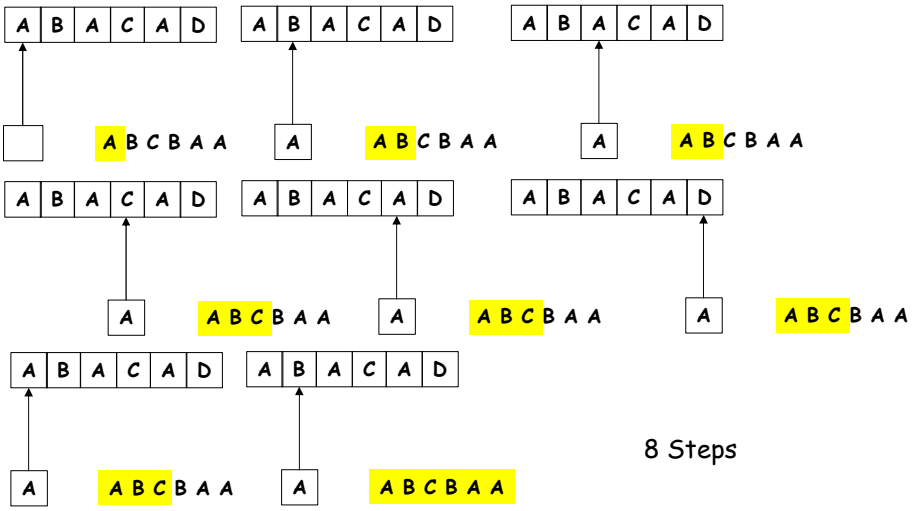
- Evict the data item that has been not been used for the longest time



A popular strategy in caching (which is e.g. frequently used when caching secondary storage pages in main memory) is least recently used (LRU). When the cache is full and a new data item arrives, the one data item is evicted that has not been used by the application for the longest time. The example illustrates of how this approach works: the cache is of size 1. After the first step the application has accessed data item A and puts it into the cache. The next item the application needs is B. When the applications receives the next item from the data broadcast, namely B, it will replace the cache with B. The same holds in the next step for C. When the application sees A again it keeps C, since C has been used by the application more recently. As soon as the application receives B again from the broadcast it can also access C from its cache, and then obtains in the next and last step the remaining data item A. This gives a total of 9 steps, i.e. time units of the data broadcast.

Strategy 2: Most Probable Accessed (MPA)

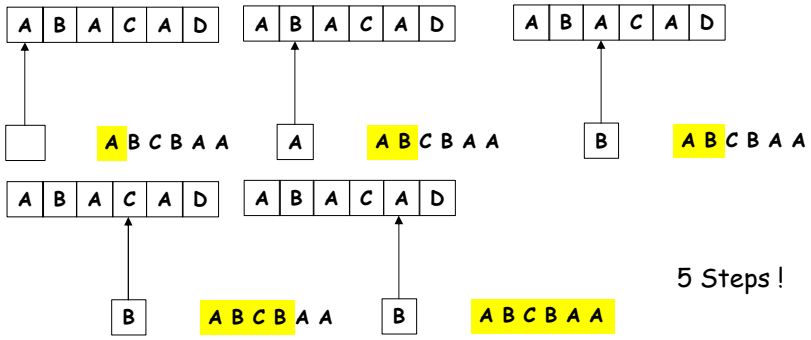
- Keep the data item that will be used most often in the future



Assume the client knows about the probability of accesses to the data items: A is accessed three times, B twice and C once. The strategy for evicting data items from the cache is to always keep the data item that is most probably accessed, which is A. In fact, this strategy is slightly superior to LRU for this example. Nevertheless we will see also this strategy is not optimal: the drawback is that it does not take into account the availability of the data items.

Strategy 3: Cost-based Cache Replacement (PIX)

- Cost-based cache replacement (PIX)
  - Replace data items with lowest ratio P/X,
  - Where P = probability of access and X = frequency of broadcast
- Example:
  - PIX(A) = $\frac{1}{2}$ / $\frac{1}{2}$ = 1, **PIX(B) = 1/3 / 1/6 = 2**, PIX(C) = 1/6 / 1/6 = 1



A better strategy is to consider the availability of data items as compared to their probability of access. Data items that are less frequently used, but that are even less frequently available are more "rare" than those that are both frequently used and frequently available. Therefore, it makes no sense to cache A, even if it is requested frequently, since it is very frequently available. On the other hand B is a bit less frequently requested but much less frequently available, and therefore the best candidate for caching. The example demonstrates this point.

Therefore a good criterion to evict data items from the cache is the access probability-to-broadcast frequency ratio (short: PIX). The lower the value is the less important the data item is, and the better a candidate it is for eviction.

Cost-based Cache Replacement

- PIX is not practical
  - Perfect knowledge of access probabilities required
  - Comparison of all cached data items required (scan through all data items)
- Idea: approximate PIX by an LRU style algorithm (LIX) that considers access probabilities
- LRU (least recently updated)
  - Cached data items are kept in a list
  - If a data item is accessed is moved to the top of the list
  - On a cache miss (when a new data items needs to be loaded) the data item at the end of the list is evicted

The problem is that PIX is not practical as it has two drawbacks. First, the access probabilities are not exactly known and second, finding the page with the lowest ratio requires essentially to scan through all cached data items, which is impractical since this operation has to be performed for every data item access. That is one of the advantages of LRU: it can be very efficiently implemented by keeping all cached items in a list, prepending data items that are accessed by the application on the top of the list, and removing data items from the end of the list when a data item needs to be evicted. All of this can be done with a low, constant number of operations.

LIX Caching Strategy

- Use modified LRU for each broadcast disk separately
  - LIX maintains one list of data items for every broadcast disk with frequency f_j
  - Data items enter the list of their corresponding broadcast disk
  - When a data item is accessed it moves on top of it's list
  - A running access probability estimate p_i is computed whenever a data item d_i is accessed
  - When a data item d_i is to be evicted a LIX value lix_i approximating the PIX value for all data items on the bottom of the lists are computed
  - The data item with lowest value lix_i is evicted

Computing the access probability estimate p_i for d_i :

Initial value $p_i = 0$

If most recent access to d_i was at time t_i (initially $t_i = 0$) and t is the time of the current access then

$$p_i := \frac{c}{t - t_i} + (1 - c)\, p_i$$

c constant ($0 < c < 1$)

Approximation of PIX value (d_i belongs to broadcast disk with frequency f_j)

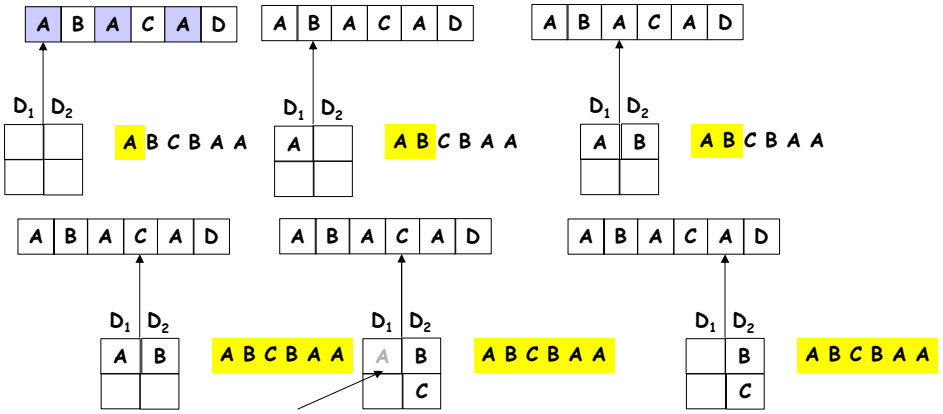
$lix_i = p_i / f_j$

The approach is to maintain a separate LRU-style list for each broadcast disk. As each disk has a fixed frequency the data items in one of the list are only distinguished by their access probabilities. As in LRU it is assumed that the bottom element of the list is the one with the lowest access probability (which must not be true but is sufficiently accurate). This gives one candidate for eviction for each list. In order to select among those the PIX value is used and the candidate with the lowest value is evicted. Since the PIX values are not known they are approximated by a running probability estimate (similarly as computing a running average), the so-called LIX value.

Example

- Two broadcast disks
  D_1={A}, D_2={B,C,D}, frequencies $f_1 = 3$, $f_2 = 1$
- Cache can store up to 2 data items

p_i	1	2	3	4	5
A	½	½	½	½	½
B	0	1/4	1/4	1/4	1/4
C					
D	0	0	0	0	0

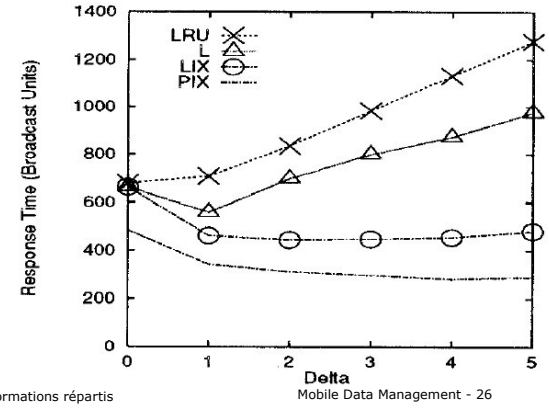


evict: $lix_A = \frac{1}{2}/3 = 1/6$, $lix_B = \frac{1}{4}/1 = 1/4$

On the top we see the running probability estimates that are kept for each data item in the cache. Only when the data item is accessed by the application the value is updated. We assume now that two cache lists are available, for each broadcast disk (frequency) one, and that the cache can hold at most two data items at the same time. The interesting step occurs when A and B are already in the cache and C arrives. Then one item needs to be evicted. As we can see from the computation of the lix value, A is evicted. This also shows that the lists corresponding to the different broadcast disks are not necessarily balanced.

Properties of LIX Caching Strategy

- Lists have non-constant length as eviction and replacement can affect different disks (resp. lists)
- Number of operations per replacement is constant
- Simulations show that LIX performs almost like PIX and much better than LRU



The illustration shows an example of simulation results comparing the strategies LRU, LIX and PIX. The different values of Delta corresponding to different experiments with changing ratios among the frequencies of the different broadcast disks. The performance criterion is the latency.

Summary

- Why is LRU not optimal for clients receiving a data broadcast ?
- What is cost-based cache replacement ?
- How is the cache (data structure) for LIX organized ?
- What is a running probability estimate and what is it used for ?

4. Indexing Broadcast Disks

- Goal: let the client doze till interesting information arrives in the broadcast
  - Power consumption in doze mode substantially lower (e.g. ratio 5000 on palmtop processors)
- Idea: Send index information, which allows the client to predict when interesting data arrives
- Assumptions
  - Consider flat broadcast disks only: no replication
  - Data items in D have search attributes
  - Data items with the same search attributes appear consecutively in the broadcast disk (clustered along search attribute)
  - Average number C of pages containing data items with the same attribute value is known (coarseness C of attribute)

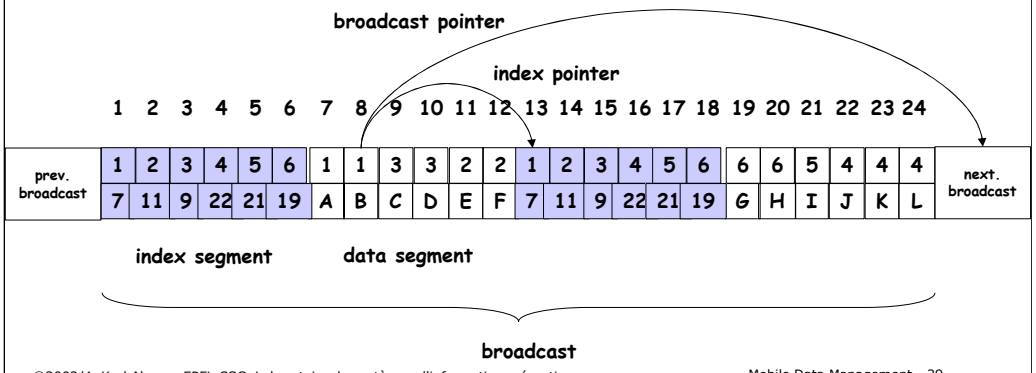
1	1	3	3	2	2	6	6	5	4	4	4	search attribute values
A	B	C	D	E	F	G	H	I	J	K	L	data items

When generating broadcast schedules we assumed that the client is listening to the broadcast till the desired data items arrive, which can be an energy-consuming task. Now we will show of how the energy consumption can be minimized by providing additional indexing information on the broadcast.

For studying indexing we will not consider any replication of data items within a broadcast schedule (as discussed earlier) in order to simplify the discussion. We assume that data items are identified by search attributes, of which the values can be shared by multiple data items. We assume that, however, the data items with the same search attributes are found consecutively in the data broadcast. We also know of how many data items share on average the same attribute value on the search attribute. This value is the coarseness of the attribute.

Broadcast Organization

- Smallest logical units: index pages and data items
  - Index pages: sequence of (attribute value, offset) pairs
  - Index segments: sets of contiguous index pages
  - Data segments: sets of contiguous data items
  - Broadcast: sequence of data segments interleaved with index segments

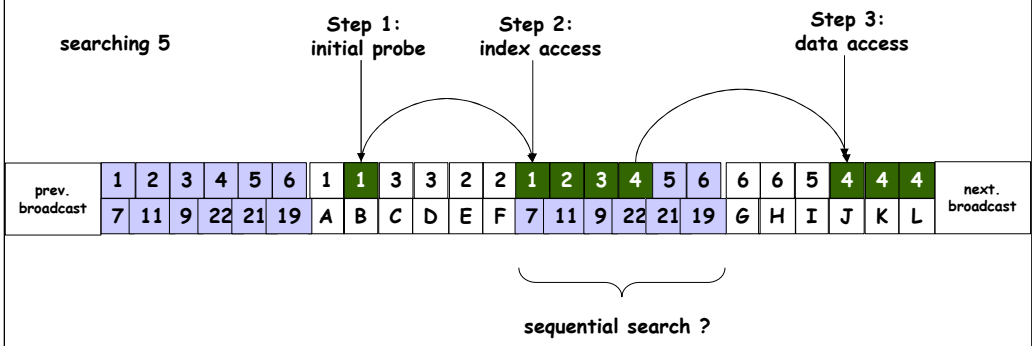


In an indexed broadcast we distinguish among index pages, that carry index information, and data pages that carry the data items. The index information is provided through (attribute_value, offset) pairs, where the offset is the position of the page counting from the beginning of the broadcast. Contiguous sequences of index and data pages are called index and data segments. The broadcast consists then of a sequence of data segments interleaved with index segments. The example shows a broadcast containing two index segments. In this (simplified) example the index consists of (attribute_value, offset) pairs that give for each of the possible attribute values 1,…,6 the offset to the position where data items with that attribute value occur.

In addition, each page maintains two more information items: a pointer to the beginning of the next index segment and a pointer to the next broadcast. Both are useful for a client. Whenever the next index segment or broadcast has to be accessed the client can go into the doze mode till they arrive.

Access Protocol for Data Retrieval

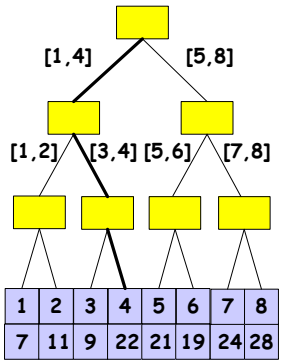
- Step 1 (initial probe): tune into broadcast and retrieve the start of the next index segment
- Step 2 (index access): tune into index segment and retrieve the start of the data page
- Step 3 (data access): tune into data segment when data arrives



This figure illustrates the typical sequence of operations that is performed for accessing a data item using an index. First, at a random time the client makes an initial probe. It accesses some data item (or index page) to obtain the start of the next index segment. Then it goes back into doze mode till it arrives. In the second step the client tunes into the index segment in order to obtain the offset of the desired data item. Once it has obtained the offset it goes back into the doze mode. Finally, when the data item arrives, it tunes in again and then retrieves the complete sequence of data items. So in total the client is only tuned in (and consumes energy) for the red (dark) pages of the broadcast.

However, we see also that the approach we take for searching in the index, namely sequential search, may become expensive in terms of tuning time. Sequential search may take long, and usually an index makes use of a data structure to perform a fast search. How can this be achieved within the data broadcast?

Index Tree

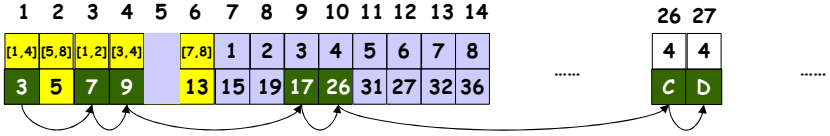


Depth of index tree: $k = \left\lceil \log_n \left(\frac{|D|}{C} \right) \right\rceil$

Index size: $|I| = \sum_{i=0}^{k-1} n^i - 1$

n = outdegree of tree

Example: |D|/C =8,
k = $\log_2(8)$ = 3
|I| =



It is actually possible to place a search structure into a data broadcast. We demonstrate of how this is done for a search tree. Assume we have 8 different attribute values to index. We build a binary tree for fast access (i.e. in $\log_2(n)$ time). It is possible to construct a sequence of index pages that represents the search tree within the data broadcast. On arranges the tree nodes, together with the information to route a search request, i.e. the interval covered by the child nodes, sequentially traversing the tree left-to-right and top-down (the root node can be omitted). When a client tunes into the start of the index segment, that has been extended in this way, it first finds the root node. From there it obtains the offset to the interval at the first level, then it tunes in at the second level and traverses the subtree from left-to-right till it finds the corresponding interval (this takes at most as many steps as the fan-out of tree nodes is, in the example the fanout n=2), and so on. Finally it arrives at the leaf level of the tree and goes from there to the data items. In this manner only O(log N) steps are required for traversing the index (as compared to O(N)) where N is the number of data items.

Latency vs. Tuning Time Tradeoff

- Optimize Latency
  - No index in broadcast

N = length of broadcast

Latency = broadcast wait = wait time + download time = N/2 + C

Tuning time = N/2 + C (no dozing)

- Optimize Tuning Time
  - Broadcast the complete index at the start of each broadcast

|I| = length of index, n out-degree of tree

Latency = probe wait + broadcast wait + download time =

$(N+|I|)/2+ (N+|I|)/2+C = N+|I|+C$

Tuning time <= $1+n \log_n(|D|/C)+C$

With data broadcasts we have a two-dimensional optimization problem: we have to optimize both latency and tuning time at the same time. It is easy to see that including index information in order to minimize the tuning time increases the latency. Therefore we have a classical trade-off situation. The two extreme cases in this trade-off are the minimization of latency and the minimization of tuning time. These two cases are easily identified: the minimal latency is achieved when no index information is included, and thus the broadcast is as short as possible (ignoring possible replication in the schedule). Then however no dozing is possible and the tuning time is large. On the other hand, the minimal tuning time is achieved, whenever it is possible to access an index. This time is composed of the time needed to access the index plus the time to access the data. This approach however increases the latency, since we have to wait first for the index to arrive before we can access the data. In the case where we include the index at the beginning of the broadcast, the resulting values for the latency and the tuning time are given above.

Minimizing Latency

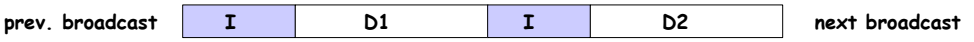
- Assume index is available: then tuning time <= 1+n $\log_n(|D|/C)+C$
  - how to optimize latency ?
- Latency depends on two factors
  - Probe wait: duration for getting the next index segment
  - Broadcast wait: duration for getting the data page containing the data item
- Tradeoff
  - Minimizing broadcast wait (e.g. by sending the index only at the beginning) increases probe wait
  - Minimizing probe wait (e.g. by sending the index in front of every data bucket) increases broadcast wait
- Optimal value in between !

An important observation is that whenever the index is available the tuning time is constant. This is also true when the index information is included multiple times. On the other hand including the index multiple times may help to decrease the latency. To understand the effect of including multiple indices we have to study ANOTHER trade-off: the latency consists of two components: the time needed till an index segment arrives and the time needed till (afterwards) the data item arrives. By including indices multiple times we can decrease the time to wait for the next index segment, in the extreme case we could include the index in front of every data item ! This however increases the total length of the broadcast and thus the time till the data items arrive.

(1,m) Indexing

- Idea: Broadcast the complete index m times during the broadcast (equally spaced)
- Analysis
  - Latency: $latency(m) = \underbrace{\frac{1}{2}\left(|I| + \frac{|D|}{m}\right)}_{\text{probe wait}} + \underbrace{\frac{1}{2}\left((m^*|I|) + |D|\right)}_{\text{broadcast wait}} + \underbrace{C}_{\text{download time}}$
  - Optimal value of m minimizing latency (solve latency(m)'=0)

$$m^* = \sqrt{\frac{|D|}{|I|}}$$



m=2

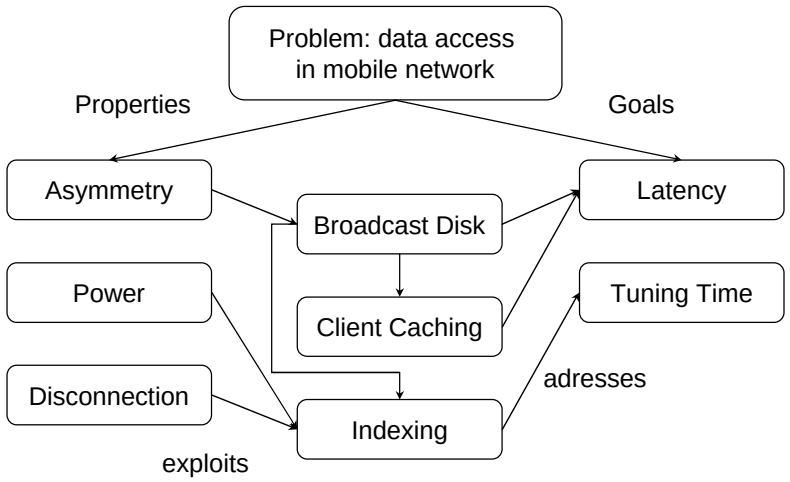
Thus there exists an optimal number of times of including the index into the broadcast (equally spaced). The index constructed by using this optimal value is called the (1,m)-Index. Determining the optimal value m for the number of times the index segment is included is not difficult: one computes the value of the latency, depending on m. Since we want to minimize this value, we compute the derivative (with respect to m) and set it to 0. Solving the equation obtained in this way, gives the optimal value m*.

One can observe some interesting properties of the resulting optimal value. For example, if the size of the index and the data segment are the same the optimal value is 1. For including two index segments the data segment should be at least four times larger than the index segment, and so on.

Summary

- How can the client put into dozing mode when searching a data item in the data broadcast ?
- Which additional information as required for data and index pages when indexing the data broadcast ?
- Which two kinds of tradeoff need to be taken into account when designing an indexing scheme ?
- What is (1,m) indexing ?

Overview



This overview shows of how the problem of data access in mobile networks has been tackled. It is interesting since it demonstrates of how problems are tackled in general. The situation is that a problem is characterized both by properties of the environment and by goals that are to be achieved. Between these two dimensions the space of solutions has to be developed.

Outlook

- Further techniques to optimize latency
  - Prefetching of pages that are expected to be used
  - Hybrid push-pull techniques
  - On-demand broadcast
    - Each broadcast based on pending requests
- Further techniques to optimize tuning time
  - Tree-based Indexing: interleave only parts of index needed in next data segment
  - Other indexing techniques (e.g. hashing)
- Considering updates
  - Broadcasting invalidation information for cache
  - Versioning Schemes

We showed in this lecture some of the basic techniques used in mobile data management. For completeness, we mention some important extensions of these techniques that have been studied: prefetching is used by clients in order to store pages that have not yet been used (as in caching) but that will probably be used in the future. Push-based techniques can be complemented by pull-based techniques, in case an upstream communication exists, in particular for the access to less frequently used data items. Broadcast schedules can also take into account pending requests.

The indexing methods we introduced can be further refined by including into index segments only information that is pertinent to the data that occurs in the immediately following data segment. Thus the index segment sizes can be substantially reduced. Also other search data structures, such as hashing can be applied.

When updates occur on the server side, the broadcasts should include information that allows clients to update their caches, in order to avoid stale data. Also versioning schemas may be employed such that when stale cache data is used, at least the stale items belong to the same version and are thus consistent with each other.

Reading

- Course material based on
  - Daniel Barbará: Mobile Computing and Databases - A Survey. TKDE 11(1): 108-117 (1999)
  - Swarup Acharya, Rafael Alonso, Michael J. Franklin, Stanley B. Zdonik: Broadcast Disks: Data Management for Asymmetric Communications Environments. SIGMOD Conference 1995: 199-210
  - Sohail Hameed, Nitin H. Vaidya: Log-Time Algorithms for Scheduling Single and Multiple Channel Data Broadcast. MOBICOM 1997: 90-99
  - Tomasz Imielinski, S. Viswanathan, B. R. Badrinath: Data on Air: Organization and Access. TKDE 9(3): 353-372 (1997)