Contents

1 Locate Servers 2
2 Framework for Network Applications 2
3 Caches 4
    3.1 LRU ........................................ 4
    3.2 Variable page sizes .......................... 7
    3.3 Modified LRU ................................. 7

Overview

The following issues appear in this lecture. (They are mentioned in the order of the layer hierarchy.)

- Applications
  - Framework; see section 2

- Services
  - Communication: Deliver files, byte streams
  - Content servers (which in turn need: load balancing, performance models, locating of servers, see section 1), and Caches, see section 3
  - Processing servers (postponed)

- Protocols

- Technology
1 Locate Servers

1. Cluster the addresses using the address structure. Ex.: N.H, i.e. Network.Host; see figure 1

2. Where to place the servers? Place the servers close to the busiest clusters. Ex.: 60% traffic in A, 30% in B, 10% in C, so you place a Server in A and one in B but probably none in C.

2 Framework for Network Applications

Object-Oriented Remote Procedure Call

Figure 2: Remote Procedure Call

You are in machine A and use machine B to execute some routine (see figure 2). What to do to avoid blocking B while processing in A?
A has a number of applications. Below this is a message service with an API. Under this is a Queue Manager (figure 3) which runs some priority queues. Between A and the rest of the world is the network.

Figure 3: RPC with Queue Manager

Issues in this context:

- **Reliability**, e.g.
  - send at least once (Ex. by acknowledgment scheme; figure 4)
  - send at most once
  - send exactly once (Ex. TCP, uses sequence numbers; figure 5)

- **Security**
  - Authentication: Message was sent from A to B; you then want to guarantee that the message comes from A
  - Encryption: Only the recipient can read the message. Ex.: Use public and secret key encryption.
3 Caches

How to update the cache (figure 6)? The objective is: Maximize “hit ratio”.

Model (figure 7): Store $k$ pages. Server has $N$ pages, $N \gg k$. A request arrives at the cache which cannot be served locally. The cache queries the server for the current page. The important question then is: “Update or not”.

3.1 LRU

A simple rule is LRU: Replace Least Recently Used page by page $i$.

We will now examine the performance of this rule. Assume that successive requests $i$ are independent and arrive with probability $p_i$, $i = 1, \ldots, N$. Assume that all pages are of the same size.

If the probabilities $p_1, \ldots, p_N$ were known, then we would place the $k$ pages with $k$ largest $p_i$’s in the cache. The probability that a request can be served locally is
Figure 7: Cache model

$p_1 + \ldots + p_k$.

Analysis

In steady-state

$$P(\text{cache contains } 1, \ldots, k) = \pi(1, \ldots, k) := \prod_{j=1}^{k} p_j$$

→ Pages that are requested often are likely to be in the cache.

Proof

$x = (1, \ldots, k)$ means that $1, \ldots, k$ are in the cache.

Let $k$ be the LRU, then $k - 1$ is the next LRU, and 1 is the most recently used.

$x_n$ is a Markov chain. If I know what is in the cache now, I can predict the following request.

$$x = (1, \ldots, k) \xrightarrow{i \notin k} (i, 1, \ldots, k - 1)$$  
$$x = (i, 1, \ldots, k) \xrightarrow{i \notin k} \text{ without } i$$  
$$(i, \ldots, i_k, \ldots) \xrightarrow{i \notin \{i_1, \ldots, i_k\}} (i, i_1, \ldots, i_k)$$

The complete Markov Chain solution is far too complicated. Instead we use a trick (which is generally a very useful one).

Let $P(x, y)$ be the probability that the state goes from $x$ to $y$. E.g.

$$x = (1, \ldots, k)$$  
$$y = (k + 3, 1, \ldots, k - 1)$$  

with $P(x, y) = P_{k+3}$
Another example:

\[ z = (3, 1, 2, 4, \ldots, k) \Rightarrow P(x, z) = P_3 \]
\[ v = (2, 1, 4, 3, 5, \ldots, k) \Rightarrow P(x, v) = 0 \]

Figure 8: The probability that the state goes from \( y \) to \( x \)

See figure 8. Must show: \( \pi P = \pi \)

\[ \sum_y \pi(y)P(y, x) = \pi(x) \quad \forall x \]

The key is like this:

Guess that \( x_n \) reversed in time is a Markov chain with transition probabilities \( p'(x, y) \).

\[ x = (1, 2, \ldots, k) \xrightarrow{P_{k+3}} \text{Forward} \quad y = (k + 3, 1, 2, \ldots, k - 1) \xrightarrow{P_k} \text{Reverse} \]

How do we proof that? We show (time-reversal trick; see figure 9):

\[ \pi(x)P(x, y) = \pi(y)P'(y, x) \quad \forall x, y \quad (1) \]

Figure 9: Equation (1) (time-reversal trick)
Assume that this is true. Then holds
\[ \sum_x \pi(x)P(x, y) = \pi(y) \]  
(2)

Instead of proving the balance equation (2) directly, we use the abovementioned trick. We will now prove (1) but will leave hands off (2).

\[ x = (1, 2, \ldots, k) \]
\[ y = (k + 3, 1, 2, \ldots, k1) \]
\[ P(x, y) = P_{k+3} \]
\[ P'(y, x) = P_k \]

And from (1):
\[ \pi(x)P(x, y) \sim \pi(x)P'(y, x) \]
\[ p(1) \ldots p(k)P_{k+3} = p(k + 3)p(1) \ldots p(k - 1)P_k \]

Q.e.d.

3.2 Variable page sizes

Model: Each \( a \) is requested with probability \( b \). All requests \( ahg \) are independent. Each page has the size \( c \). The cache can store \( ê \).

If you knew \( p_1, \ldots, p_N \): Which would be the best cache? The best cache would maximize \( \sum_{i \in I} P_i \) such that \( \sum_{i \in I} S_i \leq S \) over \( I \subset \{1, \ldots, N\} \).

This is the knapsack problem which is long-known and very difficult to solve.

Trying Intuition – one possible solution leads to Modified LRU.

3.3 Modified LRU

Replace least recently used page (figure 7) but only with a probability \( \frac{S_m}{S_i} \), with \( S_{min} := \min\{S_j, j = 1, \ldots, N\} \) as normalization constant.

\[ \pi(1, \ldots, k) = A \cdot \frac{P_1}{S_1} \cdots \frac{P_k}{S_k} \]

A is another normalization constant.

MLRU is more efficient than LRU (fig. 10). Proof also by Time-Reversal trick.

Ex. for LRU: Figure 11.

Another algorithm: CLIMB; figure 12.
Figure 10: MLRU vs. LRU

Figure 11: Example LRU

Figure 12: Example CLIMB