Topics for Discussion

April 11

HTTP Caching

What performance benefits might be achieved?
Faster response time. Why?
Draw a space-time diagram illustrating this.
Decreased bandwidth demands on WAN. Why?

What performance deterioration might occur?
Increased response time! Why? In the extreme case, if the cache hit rate is 0, caching just adds delay – the time to pass through the cache. Also if the cache is slow with respect to the the network plus the origin server, then using a cache will increase response time. What's this space-time diagram look like?

Even if the cache is faster than the network and origin server and the hit rate is positive, potential net benefit to some requests may be outweighed by slowing other requests. That is the average (or median or whatever value is chosen to optimize) response time may be worse with a cache than without one

Moral of the story: cache carefully!

What semantics problems might occur with caching?
Client is given old (stale) data.

How does HTTP avoid the semantics problems that might occur with caching?
Complicated!
First, it does not do the obvious thing – have the server tell the proxy when a page changes. Why not in HTTP? Do you know of any systems that do?

The next most obvious thing is for the proxy to ask the server whether the page has changed. How does this work?

1. Client sends a GET request to the proxy
2. Proxy finds the page in its cache
3. Proxy sends a GET request to the origin server with an If-Modified-Since header containing the modification date of the page in the cache
4. Server looks at the master page
5. If the master page has changed since the modification date of the page in the cache then the server sends the page to the proxy
6. otherwise server sends a Response with a 304 (Not Modified) Status Code
7. The proxy sends the most recent page to the client

The next most obvious thing is for the proxy to guess whether the page has changed, so many heuristics follow. These really don't interest me greatly. For details, see 13.2 Expiration Model in RFC 2616.

**How would you design and build a caching proxy server?**

What do you need?

- A cache
- Server and client code

Simple, high-level algorithm:

1. Get request
2. Look for URL in cache
3. if page is found in cache then send it to the client
4. otherwise get page from origin server, send the response to the client and, if allowed, cache it
Elaborate:

“Look for URL in cache”:
1. Canonicalize URL (e.g., convert spaces and %20 to a standard form)
2. Look up in a hash table

“Get page from origin server”:
If there already exists a connection to the server use it otherwise establish a new connection (connections must timeout).
If the origin server does not reply in time then the proxy must time out, and send a 504 Gateway Timeout response. (Illustrate with a space-time diagram.)

“if allowed”:
If a whole bunch of conditions holds, then it can be cached. E.g., if “Cache-Control” says any of these, do not cache the page (see 14.9 of RFC 2616):
• private
• no-cache
• no-store

“cache it”
Finally, if the cache fills up, one must decide what to discard (or decide to not insert the new page).

**Server Implementation**

*How does a server socket deal with ports and port numbers when accepting new connections?*

Does each new connection get a assigned a free port at the server? Let's try and see. Here's a tiny server program:
Here's Ethereal's trace of messages:
What do we see? All connections have the same port number at the server! How can this work? Won't the server confuse the data between connections?
Well, what identifies a connection?
A 5-tuple: Sender port, sender IP, TCP, receiver port, receiver IP
So if TCP data is flowing from the client (sender) to the receiver and two different connections have the same receiver port, receiver IP what distinguishes them? If the clients are on different hosts at least the sender IPs must differ. If the clients are on the same host then the sender IPs may be the same, but the 'sender ports' must differ, because a client cannot allocate a port that is already in use! Therefore the 5-tuple always differs between connections.

**How does one avoid deadlocks?**

Well, what causes deadlocks? Cycles in the waits-for graph. An easy way to avoid deadlocks is to rank the monitors (i.e., the objects that can be locked) and then follow the rule “Always try to acquire locks in increasing rank order”. That is, never try to lock a monitor that has a rank that is equal or higher than the rank of a lock that is already held.

Why does this work?
Could “increasing rank” be replaced with “decreasing rank”?
Does the order of releasing locks matter?

**How can code that reads from a stream be unit tested without a network or file?**

Use java.io.PipedInputStream and java.io.PipedOutputStream. The trick is that one can connect the output of a PipedOutputStream to the input of a PipedInputStream. Testing code can write to the piped output stream while the code being tested can read from a connected PipedInputStream. Then verify the code being tested generates the what it should.

Let's look at my example, ReadFromInputStreamTest, which tests ReadFromInputStream.

**How about the reverse: how would you test code that writes to a stream without a network or file?**

Again, connect the output of a PipedOutputStream to the input of a PipedInputStream. The code being tested writes to the piped output stream while the testing code reads from the connected PipedInputStream. Then verify that what should come out of the PipedInputStream actually comes out of the PipedInputStream.

Tada!