Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Web Caching

Overview. HTTP/1.1 caching support. Harvest. ICP. Summary cache. Adaptive Web caching. Cisco Cache Engine.

1 HTTP/1.1 caching support

- Expiration mechanisms
  - Eliminate need to send request in many cases. Saves RTTs and server load.
  - Server-specified expiration time
- Validation mechanisms
  - Eliminate need to send full responses in many cases. Saves bandwidth and server load.
  - Origin server hands out a validator (such as last modification time) together with full response.
  - Cache/client includes validator in future request. Server can then determine if full response is needed.

2 Harvest

- A general object cache based on work done in the Harvest information directory and access project.
- Design rationale: lots of read-sharing in the Internet (actually old NSFnet ftp traffic) + hierarchical internet topology + cost of miss much lower than in traditional memory systems.
- Static hierarchy with explicit parent-child and sibling pointers.
- Upon miss, query siblings. Upon timeout query parent.
- Consistency via expiration times and last mod time (heuristic); no global invalidation (doesn't scale).

3 Internet Cache Protocol (ICP)

- Originally done in the context of Harvest, but can (and is) used as protocol in many distributed cache systems.

- In version 2.0, (may) maintain notion of siblings and parent. Upon miss, multicast to siblings. If no response after timeout, contact parent. Or contact origin server. I.e., ICP is only a communication protocol and does not impose any particular architecture of caches.
- Problem: upon each miss, all siblings process message from one of the caches, increasing cache load.
- Could partition URL space amongst proxies (Cache Array Routing Protocol does this) to fix this.

4 Summary cache

- Goal: to reduce processing overhead at caches and inter-proxy communication.
- Problem: how to summarize cache contents in a succinct way?
- Possible errors that need to be tolerated:
  1. False misses. A thinks B doesn't have URL when it does. This generates extra message to origin server.
  2. False hits. A thinks B has URL but it doesn't. This generates extra query.
  3. Stale hits. A thinks B has URL and B does but it's stale. This generates extra query.
- Two parts to solution:
  1. Maintain a concise directory of neighbor caches contents.
  2. Update directory from time-to-time.
- Maintaining concise directories—options include:
  1. Exact URL (MD5 of URL). Too big.
  2. Server names (MD5 of server names). Saves about 10X, but too many false hits.
  3. Bloom filters.
- Solution: *Bloom filters.* Old but powerful and cute technique. Used in other systems, e.g., distributed database joins etc.
- Suppose you have a set $A = (a_1, a_2, \ldots, a_n)$ and you want to check if an element b belongs to A.
- Pick k independent hash functions $h_1, h_2, \ldots, h_k$ with range in $\{1, 2, \ldots, m\}$ and for each a_i set bit number i of an $m-$bit vector to 1 if any $h_j(a) = i$.
- Suppose person querying has the hash functions you used and the $m-$bit vector. Then the task becomes easy.
- Of course, you could get false misses or hits. Picking k properly minimizes the probability of false hits.
- Replace a_i's with the URLs and b with the query URL and you're home!

- Paper has results with m equal to 8, 16 and 32 times the number of documents in cache. If we pick 32X, with 8GB of cache space and 8KB average document size, m is 256 Mbits = 32 MBytes. (m, remember, is the size of the bit-vector.)
- Paper has results with $k = 4$ (number of hash functions). Sensitivity to number doesn't seem high, but no results in paper (maybe in tech report?).

5 Adaptive Web caching

- Demand-driven "diffusion" of pages from origin server to caches using multicast.
- Each cache a member of one or more multicast cache groups; overlap between groups.
- Uses IP multicast for information discovery and delivery. Randomized damping and suppression with response multicast to cache group. Upon miss in cache group, request propagates up in the "direction" of origin server.
- Cache group management protocol to auto-configure cache grous.
- Need some changes to existing protocols to perform cache group scoping etc.

6 Cisco's cache engine

- Goals:
  - Simple to deploy and manage.
  - Transparent to Web clients (unlike proxy caches).
- Basic operation
  - A cache engine is located close to Cisco IOS router
  - Router redirects all Web traffic (hard-coded to port 80 traffic) via cache engine. Cache engine either sends response ("hit") or forwards request to origin server ("miss").
  - The request may travel through further caches up the hierarchy.
  - Autonomous caches.
  - If hit in cache engine, "hijack" TCP connection and serve request. Actually, SYN response arrives from cache engine (presumably) mimicking the origin server's address. The subsequent GET arrives, cache engine checks if URL exists and is current. If not, then engine establishes separate connection to origin server.
  - What happens when user hits [shift]reload (in netscape)? (Technically, this is supposed to blow through caches and get to origin server, but this doesn't happen in reality in many cache products. The HTTP header "pragma: no cache" is supposed to be honored.)
- WCCP
  - Web cache control protocol (WCCP) between router and cache engine(s). Enables load balancing.

  - WCCP used to keep track of address space partitioning amongst caches and handling additions and removal of engines.
  - Last year, Cisco announced that it will license its WCCP protocol for communication between IOS and cache engines with other vendors.

7 Summary

- Web caching clearly beneficial in reducing load on network and server and in reducing latency.
- Do we need a communication protocol between caches? How much does it really improve hit rates? Transparent caching doesn't need this type of explicit communication (but has other drawbacks).
- Dynamic content? (I don't think the idea of "active caches" proposed by many people is necessarily a good one; if you can convince content providers to make all dynamic content into applets and also share data on which applet operates, then the right thing to do is to run these applets on client (Java security problems will almost certainly get fixed with time, and anyway many people run Java without worry today).
- What about the large numbers of unpopular pages?

