

Cooperative Caching Using Hash Routing

Based on:

- ✓ David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, Yoav Yerushalmi, **Web Caching with Consistent Hashing**, Proceedings of the Eighth International World Wide Web Conference, May, 1999.
- ✓ Vinod Valloppillil and Keith W. Ross. **Cache Array Routing Protocol v1.0**. Internet Draft, February 1998.

Cooperative Caching

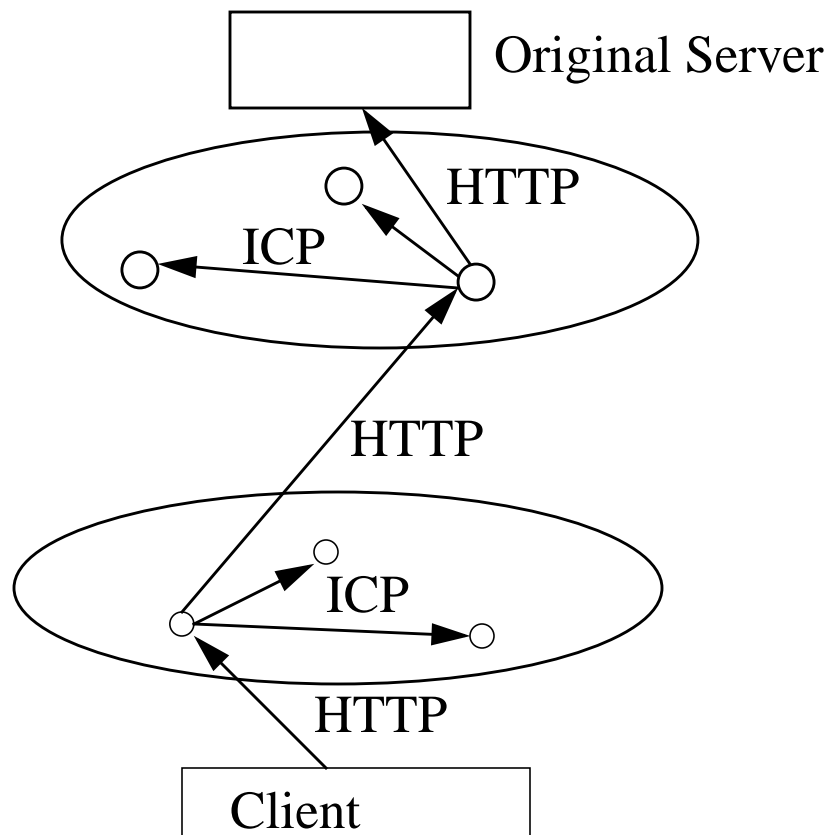
- ✓ Hit Rate dependent by $\log(\text{no. of clients})$
- ✓ A cache can directly obtain an object from one of neighboring caches, if not it addresses its parent.
- ✓ Two cooperative cache schemes:
 - ✗ ICP based (Squid)
 - ✗ CARP

ICP Based Cache Schemes

- ✓ A simple protocol for querying the neighbors about the existence of a file (RFC 2186)
- ✓ Determines:
 - ✗ which neighbor cache has a document,
 - ✗ relative latency of neighbor caches.
- ✓ When one cache queries another, there are three possible situations:
 - ✗ ICP hit message returned
 - ✗ ICP miss message returned
 - ✗ no response
- ✓ When ICP cache can not fulfill a request from its own cache then:
 - ✗ cache queries all neighbors with ICP
 - ✗ cache obtains object from first neighbor to respond with a hit
 - ✗ cache stores copy of an object and forwards copy to requestor
 - ✗ if there are no hits:
 - cache either forwards request to parent in hierarchy
 - forwards request directly to the origin server

ICP Based Cache - Example

- ✓ Client contacts level 1 cache, which does not have the requested object
- ✓ Cache uses ICP to query its siblings
- ✓ If a sibling in the first level has the object, the cache retrieves it from the sibling and forwards it to the client
- ✓ If no sibling has the object, the cache forwards the request to its parent, a 2nd level cache(HTTP req.)
- ✓ The parent cache asks its neighbors about the existence of the document. If there is no hit it asks the original server.



Drawbacks of ICP Based Scheme

- ✓ ICP message overhead
- ✓ Replication of objects: popular objects get replicated in all caches

Traffic Processing Analysis (K. Ross)

Hash Routing:

$$\begin{aligned} effort_{HR} &= \frac{1}{N} \cdot [P_{HR} \cdot 2H + (1 - P_{HR}) \cdot 4H] \\ &= \frac{H}{N} \cdot [4 - (2 \cdot P_{HR})] \end{aligned}$$

ICP:

$$\begin{aligned} e_I &= \frac{1}{N} \cdot [P_{ICP} \cdot 2H + (1 - P_I) \cdot (4H + 2I \cdot (N - 1))] + \frac{N-1}{N} \cdot (1 - P_I) \cdot 2I \\ &= \left(\frac{H}{N} \cdot [4 - (2 \cdot P_I)] + \frac{N-1}{N} \cdot (1 - P_I) \cdot 4I \right) \end{aligned}$$

Hash Routing Overview

Let's consider:

- ✓ the space of URL addresses U and a hash function h which maps U to an interval I
- ✓ $I = I_1 \cup I_2$, $I_1 \cap I_2 = \Phi$
- ✓ Two proxies P1, P2 such that an URL u is served either by P1 iff $h(u) \in I_1$ or P2 iff $h(u) \in I_2$.

Problem:

how should the intervals be choosed such that:

- ✗ introducing a new proxy only a small number of the objects are located in a wrong place
- ✗ if a cache fails all remaining objects are still were they are supposed to be.

Cache System Characteristics

Balance: Items are distributed to buckets randomly;

Monotonicity: When a bucket is added to a view, the only items reassigned are those that are assigned to the new bucket

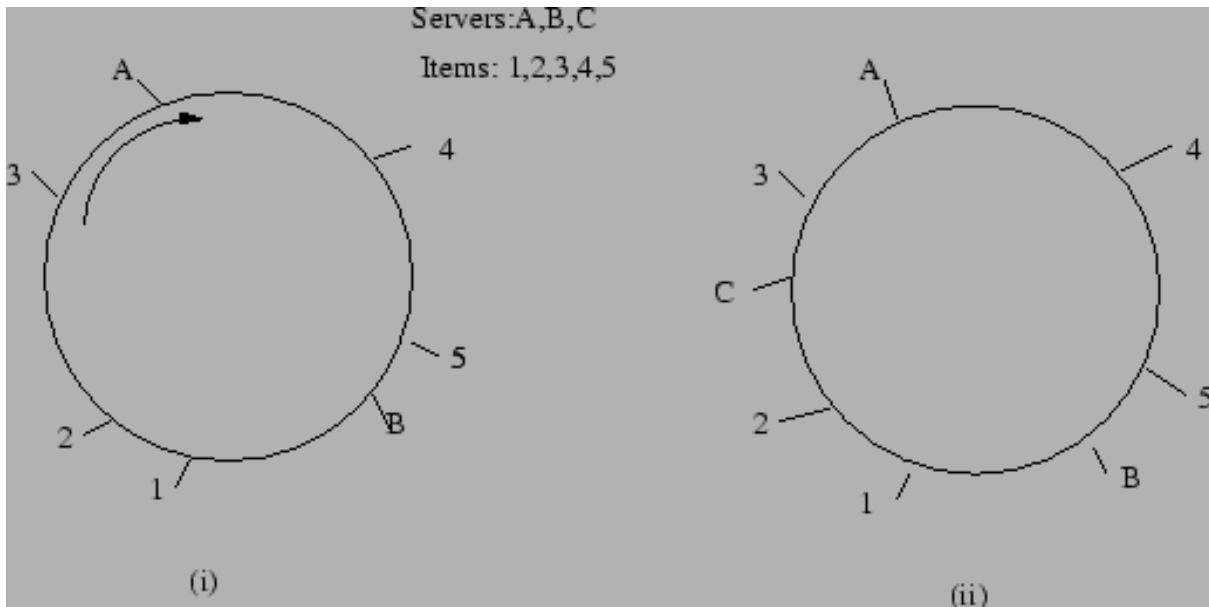
Load: The Load of a bucket is the number of items assigned to a bucket. Ideally the load should be small.

Spread: The Spread of an item is the number of buckets an item is placed in over a set of views. Ideally the spread should be small.

Robust Hashing (CARP)

- ✓ chooses a hash function $h(u,s)$, which is a function of both the URL u and proxy server s_i
- ✓ when a client wants an object at an URL u it computes N values $h(u, s_1), \dots, h(u, s_N)$ and it chooses to be served by the proxy server s_j which in the highest for the previously computed values.

Consistent Hashing (MIT-Akami)



- ✓ Both URLs and caches are mapped to points on a circle using a standard hash function. A URL is assigned to the closest cache going clockwise around the circle. Items 1, 2, and 3 are mapped to cache A. Items 4, and 5 are mapped to cache B.
- ✓ When a new cache is added the only URLs that are reassigned are those closest to the new cache going clockwise around the circle. In this case when we add the new cache only items 1 and 2 move to the new cache C. Items do not move between previously existing caches.

Heterogeneous Servers

Processing power and storage capacity can vary among servers

MS Proxies:

Target Probabilities: p_1, p_2, \dots, p_N

Introduce multipliers: x_1, \dots, x_N

$$x_1 = (N \cdot p_1)^{1/N}$$

$$x_k = \sqrt[N-k+1]{\frac{(N-k+1) \cdot (p_k - p_{k-1})}{k-1} + x_{k-1}^{N-k+1} \prod_{i=1}^{k-1} x_i}$$

Calculate $h(u, s_i), i = 1, N$

Route URL u to sibling k with highest weighted

score: $Z_k = x_k \cdot h(u, s_k)$

MIT-AKAMI Proxies:

For each proxy associate a number of points on the circle proportional with its processing power.

Membership List Update - Microsoft Proxy Server

Browser Site:

- ✓ uses an auto-configuration JavaScript file downloadable from a nearby server;
- ✓ autoconfiguration file contains the most updated version of the membership list.

Proxy Site:

- ✓ each member manages its own membership list
- ✓ **upstream array:** each time the TTL of the table expires it asks for a new version
- ✓ **local table mgmt:** each proxy watches all HTTP requests to an array member. If a request fails, the local proxy marks that proxy member as down in its table for a given TTL period and doesn't forward requests to that member until the TTL expires and the next table query shows it is active.

Membership List Update - MIT-AKAMI Cache System

- ✓ browser uses an **autoconfiguration script** to map the URL addresses to a set of 1000 virtual names.
- ✓ each virtual name is mapped to an IP address of a cache by **AKAMI DNS resolver**
- ✓ the DNS resolver reads the mapping between virtual name and IP address from a file which is updated by a **DNS helper** application which
 - ✗ monitorizes the status of the caches
 - ✗ implements consistent hashing to map virtual name to IP cache address

MS Proxy Server vs. MIT-AKAMI Cache System

MS Proxy Server:

- ✓ Even Distribution at Proxies
- ✓ Support for Heterogeneous Proxies
- ✓ Modification in the browser and Proxy Server
- ✓ Locality solved through a hierarchy scheme
- ✓ Proxy list membership update done by:
 - ✗ querying the upstream group
 - ✗ determine the state of local members

MIT-AKAMI Cache System

- ✓ Uneven Distribution on the Proxies (can be improved)
- ✓ Can be extended to support Heterogeneous Proxies
- ✓ No modification in the browsers
- ✓ Locality solved by name conventions and a network of dns helpers
- ✓ Proxy list membership update done by local dns helper