Peer-to-peer web objects cache

Tomasz Drwięga

20.03.2013 / Seminary

Outline

1. Problem statement
2. Previous work
   - Harvest (Squid) object cache
   - Consistent Hashing
   - DHT - Kademlia
3. P2P Caching
4. Challenges
   - Caching logic
   - Requests balancing
   - Caching/Streaming partial content

Problem Statement

Content provider

A lot of request can cause server to become “flooded” (“swamped”)

Network admins

Lot of outgoing traffic for the same resources results in lower QoS.

Users

Requesting large files from remote servers can cause significant delays.

Problem Statement

Content provider

A lot of request can cause server to become “flooded” (“swamped”)

Network admins

Lot of outgoing traffic for the same resources results in lower QoS.

Users

Requesting large files from remote servers can cause significant delays.

Problem Statement

Content provider

A lot of request can cause server to become “flooded” (“swamped”)

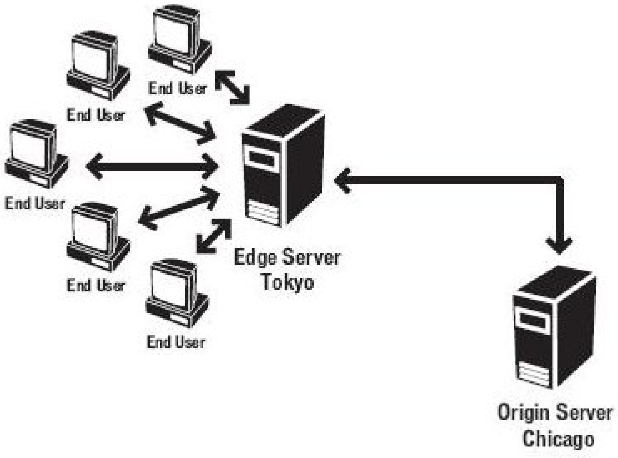
Network admins

Lot of outgoing traffic for the same resources results in lower QoS.

Users

Requesting large files from remote servers can cause significant delays.

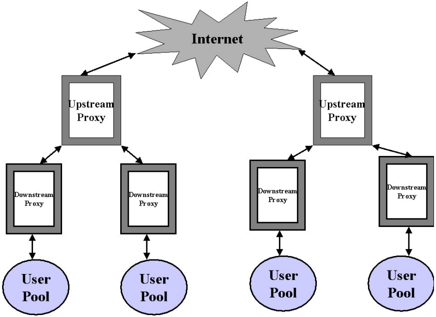
Squid object cache



Simple solution: introduce servers that will replicate original content.

Hierarchical cache

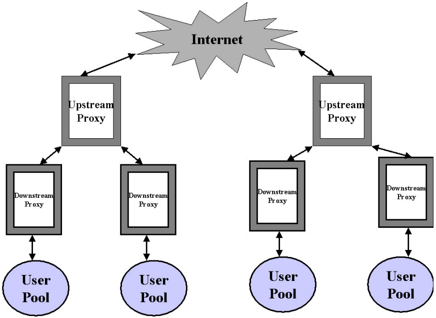
Multiple cache servers can be organized into hierarchy [2].



Leaf servers can transfer resources among one another (cooperative caching). However it leads to excessive communication [8] [9].

Hierarchical cache

Multiple cache servers can be organized into hierarchy [2].



Leaf servers can transfer resources among one another (cooperative caching). However it leads to excessive communication [8] [9].

Towards Consistent Hashing [4]

The main problem with multiple caching servers was to determine which server might contain the resource.

Naive Distribution

Let's search for resource R in server S:

$$S \equiv hash(R) \bmod n$$

Serious Flaw of Naive Distribution

When new servers are added or removed whole content has to be remapped to new targets.

Towards Consistent Hashing [4]

The main problem with multiple caching servers was to determine which server might contain the resource.

Naive Distribution

Let's search for resource R in server S:

$$S \equiv hash(R) \bmod n$$

Serious Flaw of Naive Distribution

When new servers are added or removed whole content has to be remapped to new targets.

Towards Consistent Hashing [4]

The main problem with multiple caching servers was to determine which server might contain the resource.

Naive Distribution

Let's search for resource R in server S:

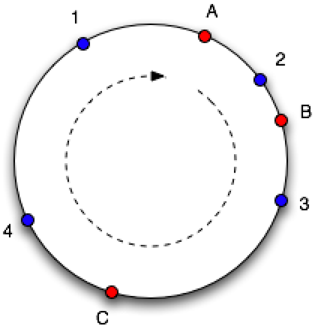
$$S \equiv hash(R) \bmod n$$

Serious Flaw of Naive Distribution

When new servers are added or removed whole content has to be remapped to new targets.

Consistent hashing

We would like to optimize process of adding/removing nodes so that new node takes its fair share of objects from others.



Consistent hashing

Every node (as well as the resource resource) is mapped to a point on unit circle. Node is responsible for keys after it's point and it's successor [5].

Distributed Hash Table

Distributed Hash Table

Decentralized (autonomous), self-organized peer-to-peer system that provides service similar to hash table. DHT should also be fault tolerant and scalable.

DHT Research was originally motivated by existing systems:

- **Napster** P2P with central index handling searches
- **Gnutella** P2P with flooding query model
- **Freenet** distributed, but no guarantee that data will be found

Four main DHTs (2001)

CAN, Chord, Pastry, Tapestry

Distributed Hash Table

Distributed Hash Table

Decentralized (autonomous), self-organized peer-to-peer system that provides service similar to hash table. DHT should also be fault tolerant and scalable.

DHT Research was originally motivated by existing systems:

- **Napster** P2P with central index handling searches
- **Gnutella** P2P with flooding query model
- **Freenet** distributed, but no guarantee that data will be found

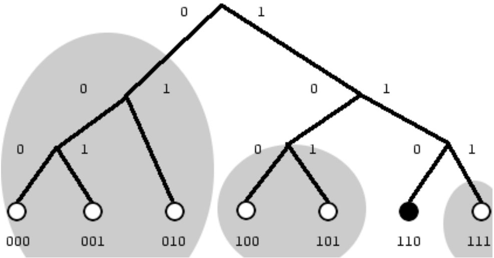
Four main DHTs (2001)

CAN, Chord, Pastry, Tapestry

Kademlia [6] (2002)

Like other DHTs, Kademlia contacts only $O(\log n)$ nodes.

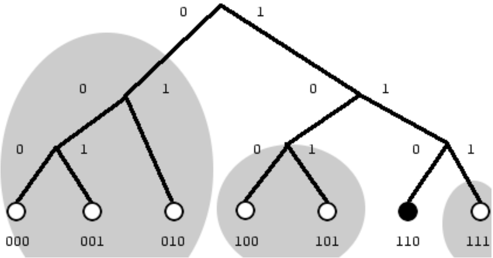
Every node and key has a 160 bit key. Node has a routing table that stores lists (k-buckets) of nodes that have specific distance from node (and shares prefix).



Kademlia [6] (2002)

Like other DHTs, Kademlia contacts only $O(\log n)$ nodes.

Every node and key has a 160 bit key. Node has a routing table that stores lists (k-buckets) of nodes that have specific distance from node (and shares prefix).



Kademlia - XOR metric

Kademlia uses XOR metric to define distance. It simplifies formal analysis, correctness proof and implementation.

Protocol messages:

- **PING** verifies that node is still connected,
- **STORE** stores (key, value) pair
- **FIND_NODE** returns *k* nodes closest to requested key
- **FIND_VALUE** returns *k* closest nodes to key or corresponding value

Kademlia - XOR metric

Kademlia uses XOR metric to define distance. It simplifies formal analysis, correctness proof and implementation.

Protocol messages:

PING	verifies that node is still connected,
STORE	stores (key, value) pair
FIND_NODE	returns *k* nodes closest to requested key
FIND_VALUE	returns *k* closest nodes to key or corresponding value

P2P Caching

Instead of downloading resource from original server we perform lookup in Kademlia DHT.

Potential benefits

- Large resources can be obtained faster (nodes are in the same LAN)
- WAN network bandwidth is saved

P2P Caching - Implementation

First attempt: Javascript browser plugin

Easy-to-install plugin that uses HTML5 APIs.

Why not? Requests has to be processed synchronously.

Native Client plugin for Chrome

Easy-to-install, good performance, but limited only to Chrome.

Why not? Lack of documentation, unsufficient APIs.

Fallback: Proxy [3]

Proxy server written in Python using Twisted framework and Entangled library implementing Kademlia.

Drawback: Requires additional configuration

P2P Caching - Implementation

First attempt: Javascript browser plugin

Easy-to-install plugin that uses HTML5 APIs.

Why not? Requests has to be processed synchronously.

Native Client plugin for Chrome

Easy-to-install, good performance, but limited only to Chrome.

Why not? Lack of documentation, unsufficient APIs.

Fallback: Proxy [3]

Proxy server written in Python using Twisted framework and Entangled library implementing Kademlia.

Drawback: Requires additional configuration

P2P Caching - Implementation

First attempt: Javascript browser plugin

Easy-to-install plugin that uses HTML5 APIs.

Why not? Requests has to be processed synchronously.

Native Client plugin for Chrome

Easy-to-install, good performance, but limited only to Chrome.

Why not? Lack of documentation, unsufficient APIs.

Fallback: Proxy [3]

Proxy server written in Python using Twisted framework and Entangled library implementing Kademlia.

Drawback: Requires additional configuration

Challenges

Caching logic

- Cache or not to cache?
- Removal of old items

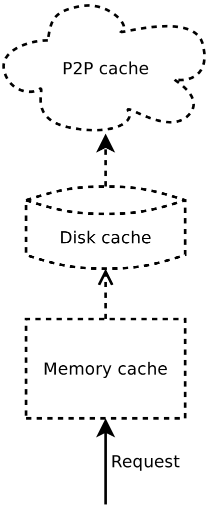
Requests balancing

Requests for items from same source can be routed to completely different parts of network (random hashing keys).

Caching and Streaming of partial content

- Cache parts of content
- Stream data instead of sending whole file at once

Caching logic [7] - multilevel cache



Retrieving item from disk or P2P cache increases latency.

We don't know if item even exists in P2P cache.

Requests balancing

We could use skip graphs [1] to ask for keys in some range (for instance resources from same domain).

Multiple files from same domain could be also stored as one resource in P2P network - we could retrieve whole bunch when any resource is requested (keywords based searching).

Streaming partial content

Increasing demand on multimedia content (e.g. Youtube)

Should parts of file be stored separately?

We need to retrieve parts in order to allow streaming of content.

References I

J. Aspnes and G. Shah.
Skip graphs.
ACM Transactions on Algorithms (TALG), 3(4):37, 2007.

A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell.
A hierarchical internet object cache.
Technical report, DTIC Document, 1995.

R. Guha and J. Wang.
Improving web access efficiency using p2p proxies.
Distributed Computing, pages 24–34, 2002.

References II

- D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin.
  Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web.
  In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

- D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi.
  Web caching with consistent hashing.
  Computer Networks, 31(11):1203–1213, 1999.

References III

P. Maymounkov and D. Mazieres.
Kademlia: A peer-to-peer information system based on the xor metric.
Peer-to-Peer Systems, pages 53–65, 2002.

R. Motwani and P. Raghavan.
Randomized algorithms.
Cambridge university press, 1995.

D. Povey, J. Harrison, et al.
A distributed internet cache.
Australian Computer Science Communications, 19:175–184, 1997.

References IV

A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy.
On the scale and performance of cooperative web proxy caching.
ACM SIGOPS Operating Systems Review, 33(5):16–31, 1999.