Peer-to-peer web objects caching proxy

Tomasz Drwięga

20.03.2013 / Seminary

Outline

Sformułowanie problemu

Problem cache'owania można przedstawić z różnych perspektyw.

Dostawca treści

Wiele żądań może spowodować "zalanie" (ang. *flooding, swamping*) serwera.

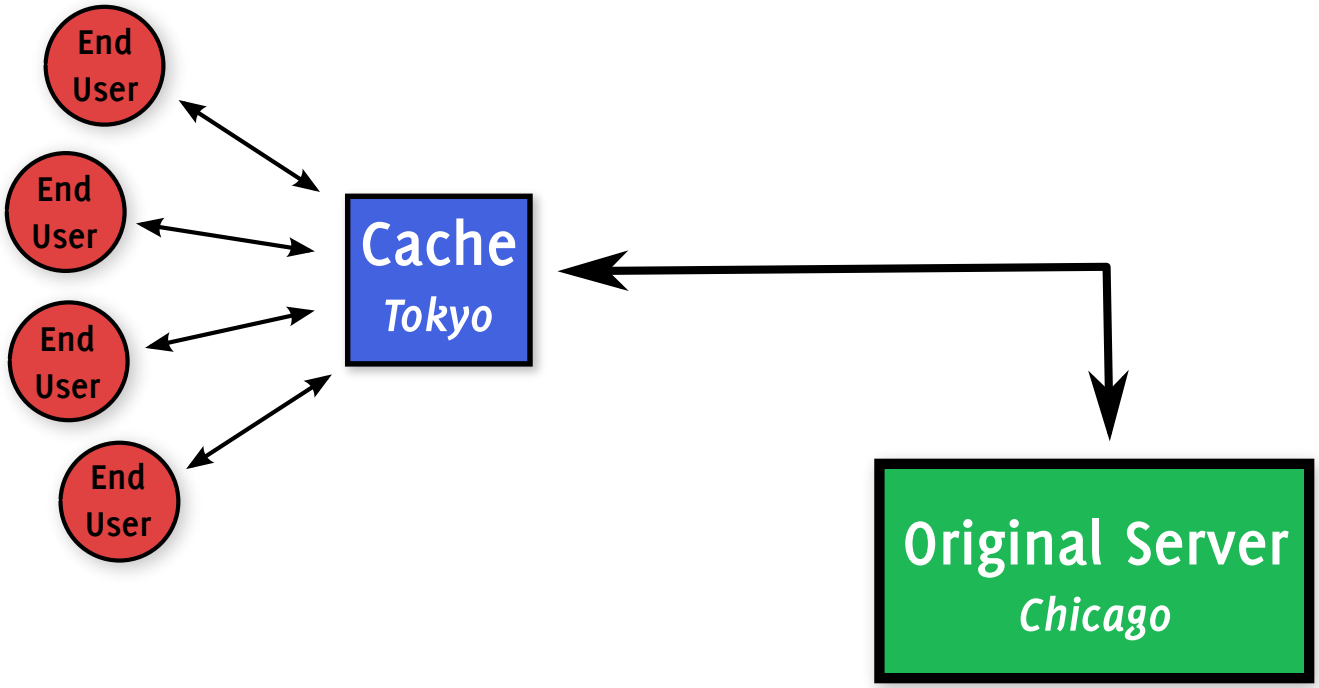
Administrator sieci

Wielokrotny transfer tego samego zasobu może prowadzić do obniżenia jakości usług.

Użytkownicy

Pobieranie dużych plików z odległych serwerów może odbywać się ze znaczącym opóźnieniem.

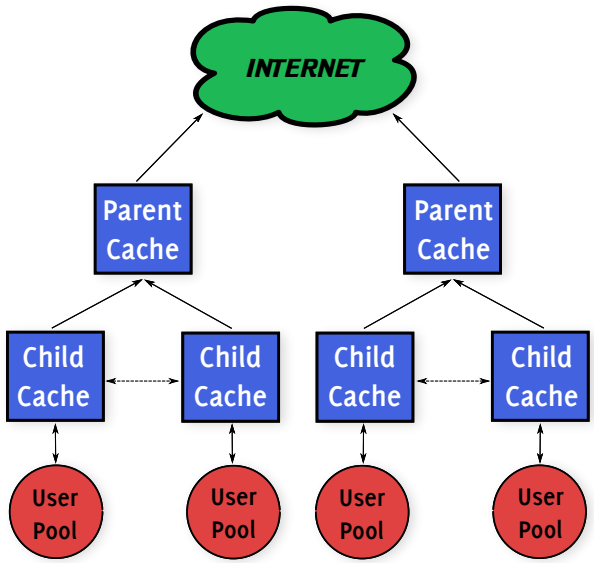
Squid object cache



Rozwiązanie: Wprowadzenie serwerów pośredniczących, które będą powielać oryginalne zasoby.

Cache hierarchiczny

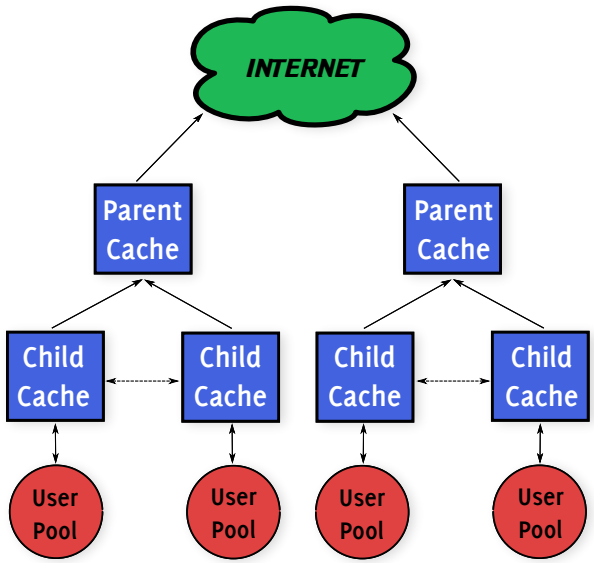
Wiele serwerów cache'ujących można zorganizować w hierarchię [Chankhunthod et al., 1995].



Serwery, znajdujące się w liściach mogą wymieniać się zasobami (cache kooperacyjny). Prowadzi to jednak do nadmiernej komunikacji [Povey et al., 1997; Wolman et al., 1999].

Cache hierarchiczny

Wiele serwerów cache'ujących można zorganizować w hierarchię [Chankhunthod et al., 1995].



Serwery, znajdujące się w liściach mogą wymieniać się zasobami (cache kooperacyjny). Prowadzi to jednak do nadmiernej komunikacji [Povey et al., 1997; Wolman et al., 1999].

W kierunku Consistent Hashing [Karger et al., 1997]

Głównym problemem w systemach cache'ujących jest określenie, który z serwerów może być odpowiedzialny za dany zasób.

Naiwny podział

Załóżmy, że poszukujemy zasobu R, który przydzielamy do serwera o indeksie S:

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

Kiedy dodajemy lub usuwamy serwery, niemal każdy zasób przydzielony jest do innego serwera.

W kierunku Consistent Hashing [Karger et al., 1997]

Głównym problemem w systemach cache'ujących jest określenie, który z serwerów może być odpowiedzialny za dany zasób.

Naiwny podział

Załóżmy, że poszukujemy zasobu R, który przydzielamy do serwera o indeksie S:

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

Kiedy dodajemy lub usuwamy serwery, niemal każdy zasób przydzielony jest do innego serwera.

W kierunku Consistent Hashing [Karger et al., 1997]

Głównym problemem w systemach cache'ujących jest określenie, który z serwerów może być odpowiedzialny za dany zasób.

Naiwny podział

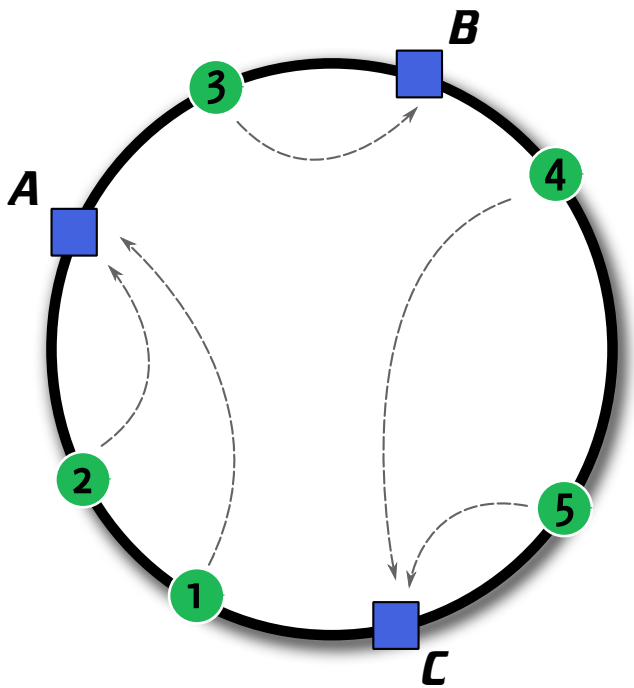
Załóżmy, że poszukujemy zasobu R, który przydzielamy do serwera o indeksie S:

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

Kiedy dodajemy lub usuwamy serwery, niemal każdy zasób przydzielony jest do innego serwera.

Consistent hashing

Celem jest poprawienie procesu dodawania i usuwania węzłów tak, żeby nowy serwer przejął równy udział od pozostałych.



Consistent hashing

Każdy węzeł (oraz każdy zasób) jest mapowany do punktu na okręgu jednostkowym. Węzeł jest odpowiedzialny za klucze, które znajdują się między nim a jego poprzednikiem [Karger et al., 1999].

Rozproszone Tablice Haszujące (DHT)

Zdecentralizowany (autonomiczny), samo-organizujący się system typu peer-to-peer, oferujący usługę przypominającą tablicę haszującą. DHT są dodatkowo odporne na błędy i skalowalne.

Badania nad DHT były motywowane istniejącymi systemami:

- **Napster** P2P z centralnym serwerem indeksującym, który obsługiwał wyszukiwanie
- **Gnutella** P2P rozsyłające zapytanie do każdego z węzłów
- **Freenet** w pełni rozproszony, ale nie gwarantujący, że dane zostaną odnalezione

Cztery główne DHT (2001)

CAN, Chord, Pastry, Tapestry

Rozproszone Tablice Haszujące (DHT)

Zdecentralizowany (autonomiczny), samo-organizujący się system typu peer-to-peer, oferujący usługę przypominającą tablicę haszującą. DHT są dodatkowo odporne na błędy i skalowalne.

Badania nad DHT były motywowane istniejącymi systemami:

Napster P2P z centralnym serwerem indeksującym, który obsługiwał wyszukiwanie

Gnutella P2P rozsyłające zapytanie do każdego z węzłów

Freenet w pełni rozproszony, ale nie gwarantujący, że dane zostaną odnalezione

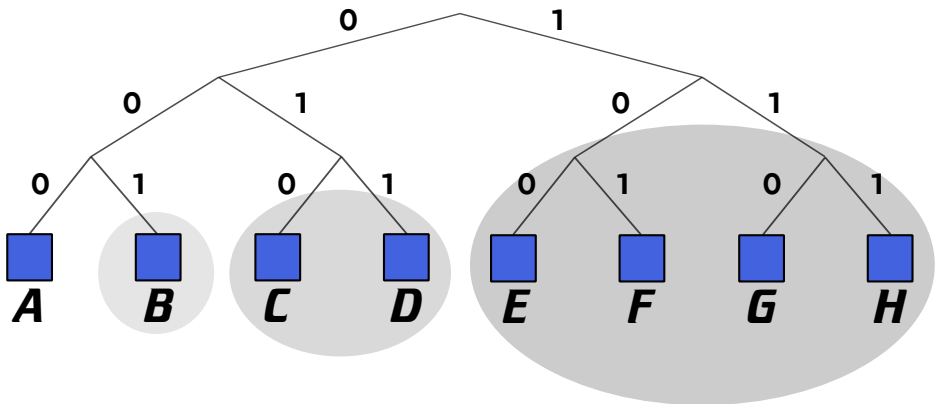
Cztery główne DHT (2001)

CAN, Chord, Pastry, Tapestry

Kademlia [Maymounkov and Mazieres, 2002]

Podobnie jak pozostałe DHT, Kademlia podczas wyszukiwania kontaktuje się tylko z $O(\log n)$ węzłami.

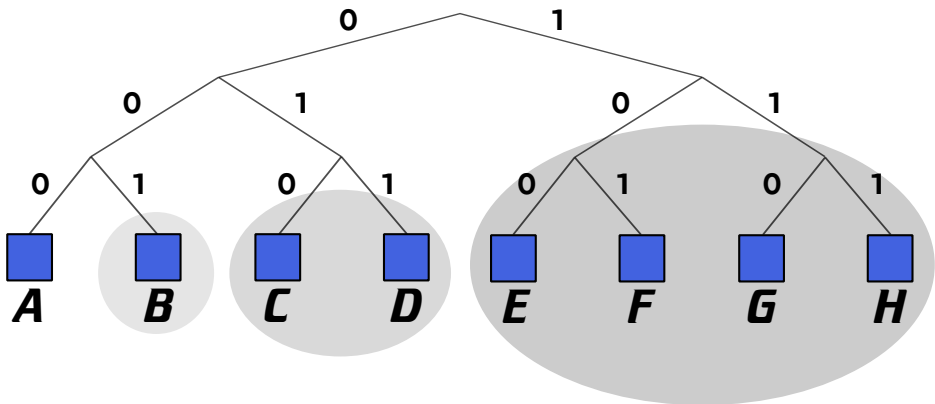
Każdemu węzłowi przypisany jest 160-bitowy klucz. Węzeł utrzymuje tablicę routingu, w której przechowuje listy (*k-buckets*) węzłow, z którymi dzieli prefiks długości i, ale różni się na na bicie $i+1$. Takie *k-buckets* istnieją dla każdego $i \in [0, 160)$.



Kademlia [Maymounkov and Mazieres, 2002]

Podobnie jak pozostałe DHT, Kademlia podczas wyszukiwania kontaktuje się tylko z $O(\log n)$ węzłami.

Każdemu węzłowi przypisany jest 160-bitowy klucz. Węzeł utrzymuje tablicę routingu, w której przechowuje listy (*k-buckets*) węzłow, z którymi dzieli prefiks długości i, ale różni się na na bicie $i+1$. Takie *k-buckets* istnieją dla każdego $i \in [0, 160)$.



Kademlia - metryka XOR

Kademlia korzysta z metryki XOR to określenia odległości między węzłami. Oznacza to, że węzły, które mają długi wspólny prefiks są blisko siebie. Upraszcza to formalną analizę, dowód poprawności i implementację.

Wiadomości protokołu:

PING	weryfikuje, że węzeł jest aktywny,
STORE	prosi węzeł o zapamiętanie pary (klucz, wartość)
FIND_NODE	zwraca k najbliższych węzłów dla zadanego ID
FIND_VALUE	zwraca k najbliższych węzłów lub przypisaną wartość dla danego ID

Kademlia - metryka XOR

Kademlia korzysta z metryki XOR to określenia odległości między węzłami. Oznacza to, że węzły, które mają długi wspólny prefiks są blisko siebie. Upraszcza to formalną analizę, dowód poprawności i implementację.

Wiadomości protokołu:

PING	weryfikuje, że węzeł jest aktywny,
STORE	prosi węzeł o zapamiętanie pary (klucz, wartość)
FIND_NODE	zwraca k najbliższych węzłów dla zadanego ID
FIND_VALUE	zwraca k najbliższych węzłów lub przypisaną wartość dla danego ID

Kademlia - porównanie

Metryka XOR upraszcza algorytm routingu - w przeciwieństwie do Pastry czy Tapestry ten sam algorytm jest używany podczas całego procesu.

Ponieważ XOR jest ???jednokierunkowa??? (czyli dla dowolnego x i Δ istnieje tylko jeden y taki, że $d(x, y) = \Delta$) wyszukiwania tego samego klucza zbiegają po tych samych ścieżkach.
Zatem możliwe jest cache'owanie par (klucz, wartośc) na węzłach znajdujących się na tej ścieżce.

Takie cache'owanie pozwala na przyspieszenie wyszukiwania popularnych zasobów w sieci.

Kademlia - porównanie

Metryka XOR upraszcza algorytm routingu - w przeciwieństwie do Pastry czy Tapestry ten sam algorytm jest używany podczas całego procesu.

Ponieważ XOR jest ???jednokierunkowa??? (czyli dla dowolnego x i Δ istnieje tylko jeden y taki, że $d(x, y) = \Delta$) wyszukiwania tego samego klucza zbiegają po tych samych ścieżkach.
Zatem możliwe jest cache'owanie par (klucz, wartośc) na węzłach znajdujących się na tej ścieżce.

Takie cache'owanie pozwala na przyspieszenie wyszukiwania popularnych zasobów w sieci.

Kademlia - porównanie

Metryka XOR upraszcza algorytm routingu - w przeciwieństwie do Pastry czy Tapestry ten sam algorytm jest używany podczas całego procesu.

Ponieważ XOR jest ???jednokierunkowa??? (czyli dla dowolnego x i Δ istnieje tylko jeden y taki, że $d(x, y) = \Delta$) wyszukiwania tego samego klucza zbiegają po tych samych ścieżkach.
Zatem możliwe jest cache'owanie par (klucz, wartośc) na węzłach znajdujących się na tej ścieżce.

Takie cache'owanie pozwala na przyspieszenie wyszukiwania popularnych zasobów w sieci.

P2P Caching

Zamiast pobierać zasoby z oryginalnego serwera przeszukujemy najpierw rozproszoną tablicę haszującą, opartą na Kademli.

Potencjalne zalety

- “Duże” zasoby mogą zostać pobrane szybciej (węzły należą do tej samej sieci LAN)
- Przepustowość łącza WAN jest oszczędzana

P2P Caching - Implementacja

Pierwsza próba: Wtyczka przeglądarkowa w Javascript

Łatwa w instalacji wtyczka używająca nowych API z HTML5.
Dlaczego nie? Żądania musiałyby być przetwarzane synchronicznie.

Wtyczka Native Client dla Chrome

Łatwa w instalacji, oferująca dobrą wydajność, ale ograniczona tylko do przeglądarek Chrome.
Dlaczego nie? Brak dokumentacji, niewystarczające API.

Fallback: Proxy cache'ujące

Serwer proxy napisany w jęzku Python z użyciem frameworku Twisted i biblioteki Entangled.
Wada: Wymagana jest dodatkowa konfiguracji przeglądarki.

P2P Caching - Implementacja

Pierwsza próba: Wtyczka przeglądarkowa w Javascript

Łatwa w instalacji wtyczka używająca nowych API z HTML5.
Dlaczego nie? Żądania musiałyby być przetwarzane synchronicznie.

Wtyczka Native Client dla Chrome

Łatwa w instalacji, oferująca dobrą wydajność, ale ograniczona tylko do przeglądarek Chrome.
Dlaczego nie? Brak dokumentacji, niewystarczające API.

Fallback: Proxy cache'ujące

Serwer proxy napisany w jęzku Python z użyciem frameworku Twisted i biblioteki Entangled.
Wada: Wymagana jest dodatkowa konfiguracji przeglądarki.

P2P Caching - Implementacja

Pierwsza próba: Wtyczka przeglądarkowa w Javascript

Łatwa w instalacji wtyczka używająca nowych API z HTML5.
Dlaczego nie? Żądania musiałyby być przetwarzane synchronicznie.

Wtyczka Native Client dla Chrome

Łatwa w instalacji, oferująca dobrą wydajność, ale ograniczona tylko do przeglądarek Chrome.
Dlaczego nie? Brak dokumentacji, niewystarczające API.

Fallback: Proxy cache'ujące

Serwer proxy napisany w jęzku Python z użyciem frameworku Twisted i biblioteki Entangled.
Wada: Wymagana jest dodatkowa konfiguracji przeglądarki.

Wyzwania

Caching logic

- To cache or not to cache? That is the question!
- Removal of old items

Requests balancing

Requests for items from the 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 files at once

Wyzwania

Caching logic

- To cache or not to cache? That is the question!
- Removal of old items

Requests balancing

Requests for items from the 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 files at once

Wyzwania

Caching logic

- To cache or not to cache? That is the question!
- Removal of old items

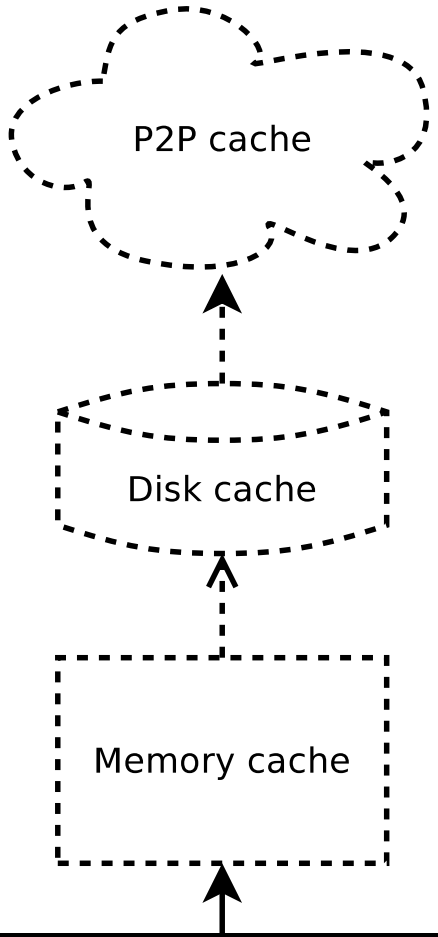
Requests balancing

Requests for items from the 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 files at once

Caching logic Motwani and Raghavan [1995] - multilevel cache



Retrieving an item from disk or P2P cache increases latency.

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

Requests balancing

We could use skip graphs [Aspnes and Shah, 2007] 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 a file be stored separately?

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

References I

James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms (TALG)*, 3(4):37, 2007.

Anawat Chankhunthod, Peter B Danzig, Chuck Neerdaels, Michael F Schwartz, and Kurt J Worrell. A hierarchical internet object cache. Technical report, DTIC Document, 1995.

David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel 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.

References II

David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. *Computer Networks*, 31(11):1203–1213, 1999.

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

Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

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

References III

Alec Wolman, M Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M Levy. On the scale and performance of cooperative web proxy caching. *ACM SIGOPS Operating Systems Review*, 33 (5):16–31, 1999.