

Peer-to-peer web objects caching proxy

Tomasz Drwięga

08.05.2013 / Seminarium

Agenda

- 1 Poprzednie prace
 - Harvest (Squid) object cache
 - Consistent Hashing
 - DHT - Kademlia
- 2 P2P Caching
- 3 Wyzwania
 - Algorytmy cache'owania
- 4 Przeprowadzone testy

Sformułowanie problemu

Problem cache'owania można przedstawić z różnych perspektyw.

Dostawca treści

Wiele żądań może spowodować “zalenie” (ang. *flooding*, *swamping*) serwera.

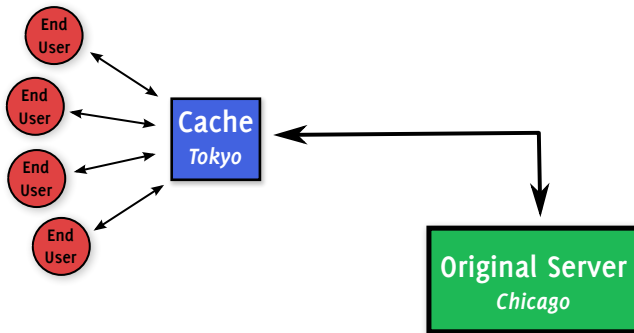
Administrator sieci

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

Użytkownicy

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

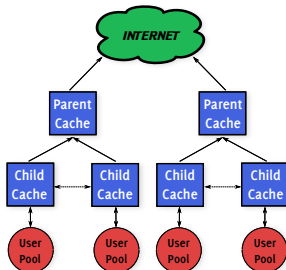
Squid object cache



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

Cache hierarchiczny

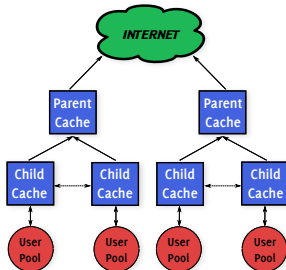
Wiele serwerów cache'ujących można zorganizować w hierarchię [Chankhunthod et al., 1995].



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

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

$$S \equiv \text{hash}(R) \bmod n$$

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

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

$$S \equiv \text{hash}(R) \bmod n$$

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

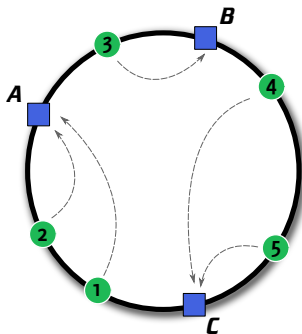
Założmy, że poszukujemy zasobu R , który przydzielamy do serwera o indeksie S :

$$S \equiv \text{hash}(R) \bmod n$$

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

Gnutella P2P rozsyłające zapytanie do każdego z węzłów w pewnym promieniu

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

Gnutella P2P rozsyłające zapytanie do każdego z węzłów w pewnym promieniu

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

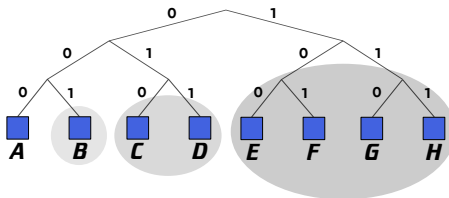
Cztery główne DHT (2001)

CAN, Chord, Pastry, Tapestry

Kademlia [Maymounkov and Mazieres, 2002]

Podobnie jak pozostałe DHT, Kademia 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*) węzłów, 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 danego 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 danego ID

FIND_VALUE zwraca k najbliższych węzłów lub przypisaną wartość dla danego ID

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 języku Python z użyciem frameworku Twisted i biblioteki Entangled.

Wada: Wymagana jest dodatkowa konfiguracja 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 języku Python z użyciem frameworku Twisted i biblioteki Entangled.

Wada: Wymagana jest dodatkowa konfiguracja 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 języku Python z użyciem frameworku Twisted i biblioteki Entangled.

Wada: Wymagana jest dodatkowa konfiguracja przeglądarki.

Wyzwania

Logika cache'owania

W jaki sposób zarządzać zasobami w cache'u o ograniczonym rozmiarze.

Równoważenie obciążenia

Żądania o zasoby z tej samej strony internetowej mogą być routowane do zupełnie innych części sieci z powodu losowego wyboru kluczy.

Wybór / porównanie różnych DHT

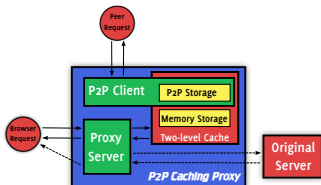
Pomimo tej samej teoretycznej złożoności routingu, różne sieci P2P mogą istotnie różnić się w zastosowaniu praktycznym.

Logika cache'owania

Pobieranie zasobu z sieci P2P zwiększa opóźnienie, popularne elementy warto przechowywać w pamięci.

Porównanie różnych algorytmów:

- LRU
- LFU
- FIFO



Wyniki eksperymentów

Dane testowe

- Problem ze znalezieniem aktualnych logów z serwerów (najnowsze z roku 2007).
- Możliwość wygenerowania danych losowych

Porównanie z innymi rozwiązaniami

- Wyniki z prac bazują na przestarzałych danych
- Brak kodu lub warunków na uruchomienie aplikacji

Wyniki eksperymentów

Użycie danych z serwera proxy z 2007 roku.

Algorytmy

Porównanie algorytmów cachowania: LRU, LFU i FIFO.

TOOD: !!!!

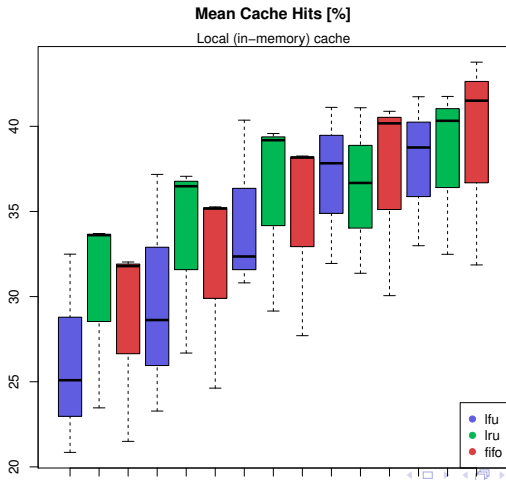
Równoważenie obciążenia

Grupowanie (bazowane na czasie) zasobów pochodzących z jednej strony (np. obrazki) pod wspólnym kluczem w sieci P2P.

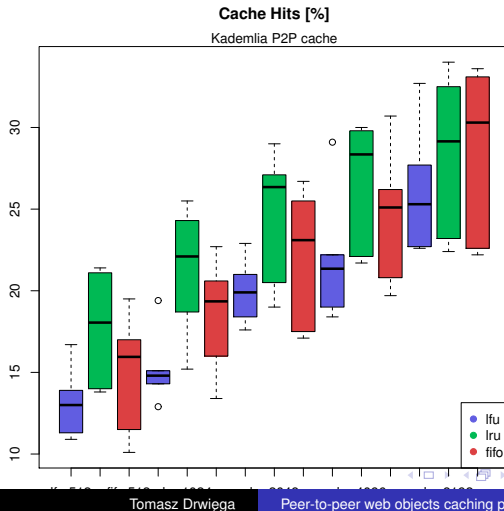
Porównanie opóźnień w Kademlii, Chordzie i Pastry.

TOOD: /!!!!

Wyniki eksperymentów - Local Cache



Wyniki eksperymentów - P2P Cache



Wyniki eksperymentów - Wnioski

TOOD: !!!!

Literatura I

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.

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.

Literatura II

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

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

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.