Distributed Hash Tables i ich zastosowania

Maciek Bryński

Seminarium z Systemów Rozproszonych, 2005/2006

Spis treści

Czym są DHT?

Co to jest?

- Klasa systemów rozproszonych.
- „Struktura danych” <klucz, dane>.
- Mając dany klucz można uzyskać dane.

Cechy:

- Odporność na awarię wezłów.
- Niski koszt wyszukania odpowiedniego węzła w stosunku do rozmiary sieci.
- Duża wydajność.

Czym są DHT?

Co to jest?

- Klasa systemów rozproszonych.
- „Struktura danych” <klucz, dane>.
- Mając dany klucz można uzyskać dane.

Cechy:

- Odporność na awarię wezłów.
- Niski koszt wyszukania odpowiedniego węzła w stosunku do rozmiary sieci.
- Duża wydajność.

Typowe zastosowania

- Sieci P2P
  1. Overnet — zamknięta implementacja protokołu Kademlia (eDonkey, MLdonkey)
  2. Kad Network — eMule, MLdonkey, aMule
  3. wiele innych — sporo aplikacji P2P implementuje własne sieci oparte na protokole Kademlia.
- Rozproszone systemy plików
  4. CFS — Cooperative File System
- Inne
  5. DDNS (Distributed DNS)
  6. Rozproszone cache'owanie stron WWW

Typowe zastosowania

- Sieci P2P
  1. Overnet — zamknięta implementacja protokołu Kademlia (eDonkey, MLdonkey)
  2. Kad Network — eMule, MLdonkey, aMule
  3. wiele innych — sporo aplikacji P2P implementuje własne sieci oparte na protokole Kademlia.
- Rozproszone systemy plików
  4. CFS — Cooperative File System
- Inne
  5. DDNS (Distributed DNS)
  6. Rozproszone cache'owanie stron WWW

Typowe zastosowania

- Sieci P2P
  1. Overnet — zamknięta implementacja protokołu Kademlia (eDonkey, MLdonkey)
  2. Kad Network — eMule, MLdonkey, aMule
  3. wiele innych — sporo aplikacji P2P implementuje własne sieci oparte na protokole Kademlia.
- Rozproszone systemy plików
  4. CFS — Cooperative File System
- Inne
  5. DDNS (Distributed DNS)
  6. Rozproszone cache'owanie stron WWW

Istniejące protokoły routowania

- Protokoły, które zostały opisane po raz pierwszy w 2001 r.
  1. **Chord**
  2. CAN — Content Addressable Network
  3. Pastry
  4. Tapestry
- Protokoły, które zostały opisane po 2001 r.
  5. **Kademlia**
  6. DKS — Distributed K-ary System
  7. Bamboo

Istniejące protokoły routowania

- Protokoły, które zostały opisane po raz pierwszy w 2001 r.
  1. **Chord**
  2. CAN — Content Addressable Network
  3. Pastry
  4. Tapestry
- Protokoły, które zostały opisane po 2001 r.
  5. **Kademlia**
  6. DKS — Distributed K-ary System
  7. Bamboo

Chord — wstęp

Założenia

- Węzły położone są na okręgu *mod* 2^n.
- Za przechowywanie danych o kluczu k odpowiedzialny jest następnik k, czyli węzeł o najmniejszym $id \geqslant k$.
- Podczas podłączania i odłączania z sieci może nastąpić migracja danych.

Czasy operacji

- Wyszukiwanie klucza — $O(log(n))$
- Podłączenie do sieci — $O(log^2(n))$
- Odłączenie od sieci — $O(log^2(n))$

Chord — wstęp

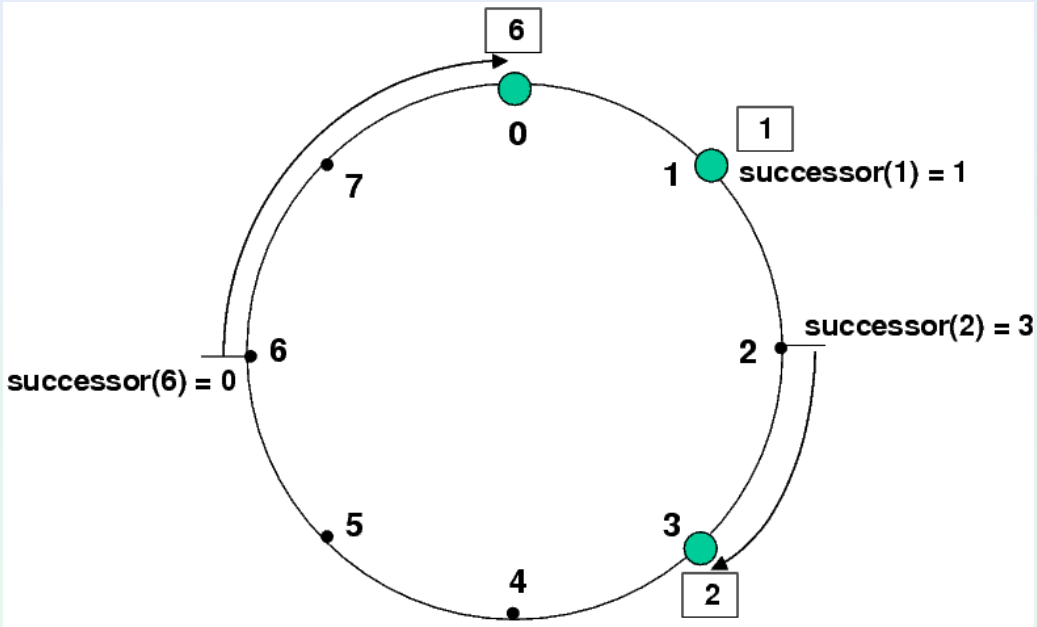
Założenia

- Węzły położone są na okręgu *mod* 2^n.
- Za przechowywanie danych o kluczu k odpowiedzialny jest następnik k, czyli węzeł o najmniejszym $id \geqslant k$.
- Podczas podłączania i odłączania z sieci może nastąpić migracja danych.

Czasy operacji

- Wyszukiwanie klucza — $O(log(n))$
- Podłączenie do sieci — $O(log^2(n))$
- Odłączenie od sieci — $O(log^2(n))$

Przykładowa sieć



Rysunek: Przykładowa sieć

Przykładowa sieć dla $n = 3$ zawierająca 3 węzły.

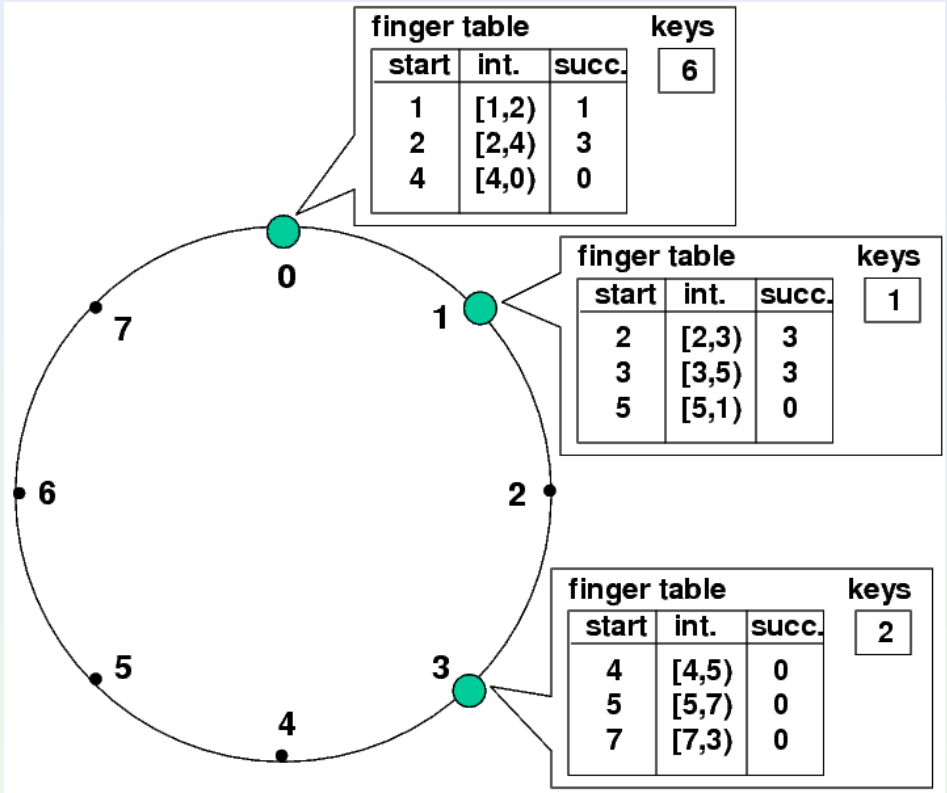
Struktury danych

Każdy węzeł w sieci posiada następujące informacje o sieci:

- następnik — kolejny węzeł na okręgu
- poprzednik — poprzedni węzeł na okręgu
- tablice $finger[k]$ — dla danego k węzeł, który jest w odległości $\geqslant 2^k$ od naszego

Całkowity rozmiar struktur danych w węźle to $O(log(n))$.
Istotne jest, żeby te informacje były aktualne.

Przykładowa sieć



Rysunek: Przykładowa sieć

Tablice *finger*[*k*] dla przykładowej sieci.

Wyszukiwanie klucza

Wyszukanie klucza odpowiada znalezieniu następnika dla id.

n.FindSuccessor(*id*)

n = FindPredecessor(*id*);
return *n*.successor;

n.FindPredecessor(*id*)

$n' = n$;
while ($id \in (n', n'.\text{successor}]$)
    $n' = n'.\text{ClosestPrecedingFinger}(id)$;
return n';

n.ClosestPrecedingFinger(id)

for $i = m$ downto 1
    if ($finger[i].\text{node} \in (n, id)$)
        return $finger[i]$.node;
return *n*;

Wyszukiwanie klucza

Wyszukanie klucza odpowiada znalezieniu następnika dla id.

n.FindSuccessor(*id*)

n = FindPredecessor(*id*);
return *n*.successor;

n.FindPredecessor(*id*)

$n' = n$;
while ($id \in (n', n'.\text{successor}]$)
    $n' = n'$.ClosestPrecedingFinger(*id*);
return n';

n.ClosestPrecedingFinger(id)

for $i = m$ downto 1
    if ($finger[i].\text{node} \in (n, id)$)
        return $finger[i]$.node;
return *n*;

Wyszukiwanie klucza

Wyszukanie klucza odpowiada znalezieniu następnika dla id.

n.FindSuccessor(*id*)

n = FindPredecessor(*id*);
return *n*.successor;

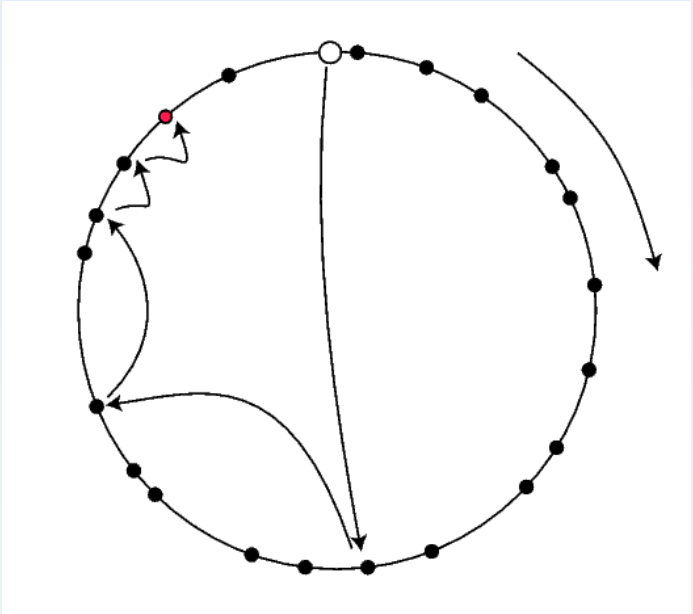
n.FindPredecessor(*id*)

$n' = n$;
while ($id \in (n', n'.\text{successor}]$)
    $n' = n'$.ClosestPrecedingFinger(*id*);
return n';

n.ClosestPrecedingFinger(id)

for $i = m$ downto 1
    if ($finger[i].\text{node} \in (n, id)$)
        return $finger[i]$.node;
return *n*;

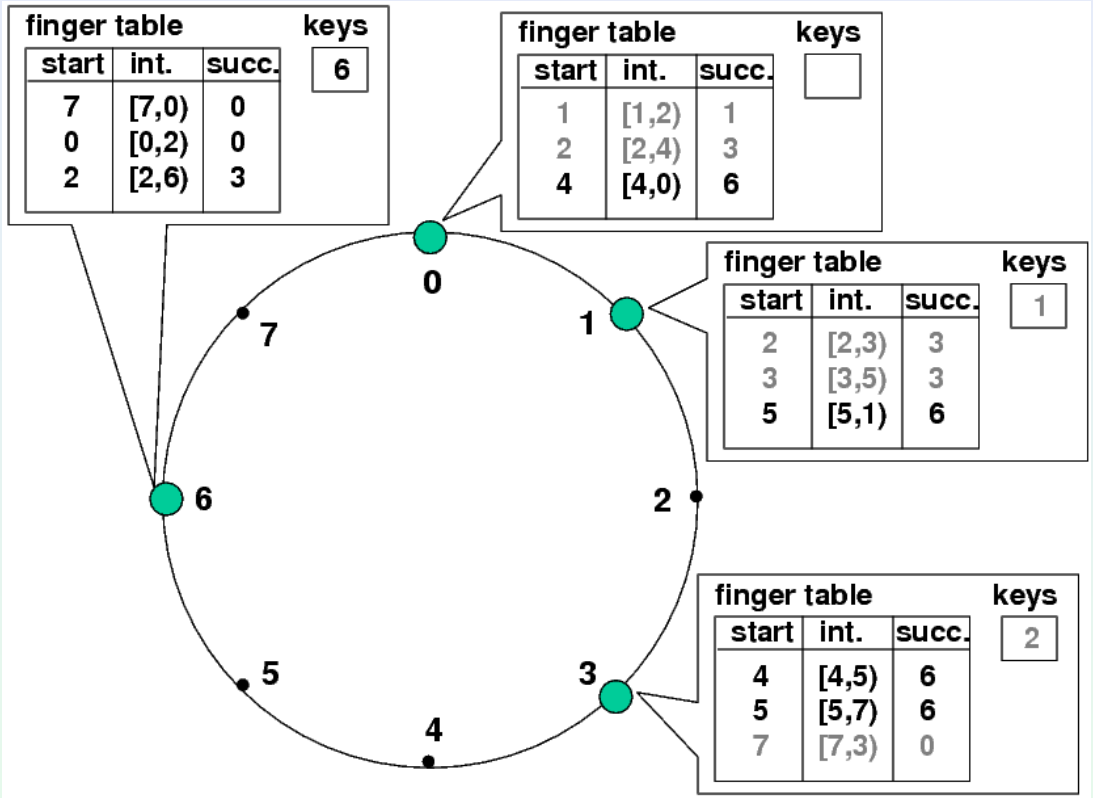
Przykładowe wyszukiwanie



Rysunek: Przykładowe wyszukiwanie

- Każdy krok wyszukiwania zmniejsza odległość o połowę.
- Oczekiwany czas wyszukiwania $O(log(n))$.

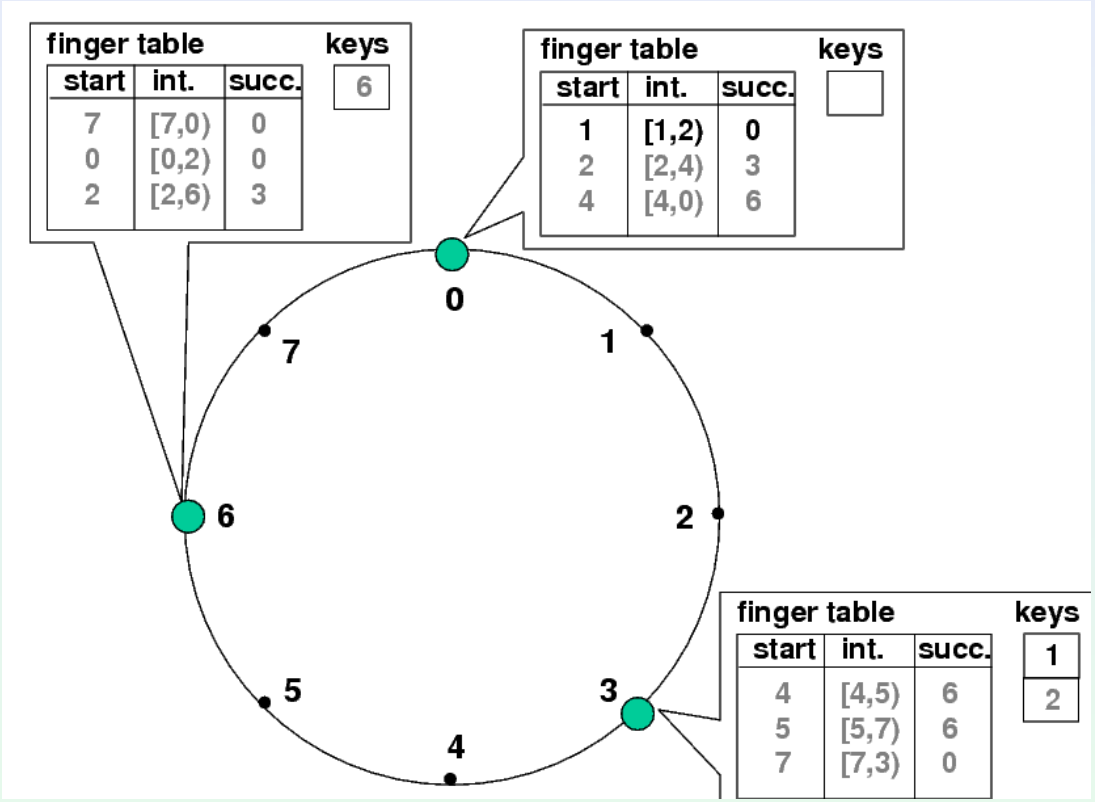
Dołączenie węzła do sieci



Rysunek: Dołączenie węzła

- Tablice *finger*[*k*] po dołączeniu nowego węzła.

Odłączenie węzła z sieci



Rysunek: Odłączenie węzła

- Tablice *finger*[*k*] po odłączeniu istniejącego węzła.

Chord — podsumowanie

Zalety

- Dobra wydajność.

Wady

- Skomplikowana implementacja.
- Duża wrażliwość na awarię węzłów.
- Brak równoległości przy wyszukiwaniu.

Kademlia — wstęp

Założenia

- Oparta na metryce XOR.
- Informacje o strukturze sieci są aktualizowane podczas wyszukiwania danych.
- Prawdopodobieństwo, że węzeł pozostanie w sieci jest zależne od czasu pobytu w tej sieci.

Czasy operacji

- Wyszukiwanie klucza — $O(log(n))$
- Podłączenie do sieci — $O(log(n))$
- Odłączenie od sieci — brak akcji

Kademlia — wstęp

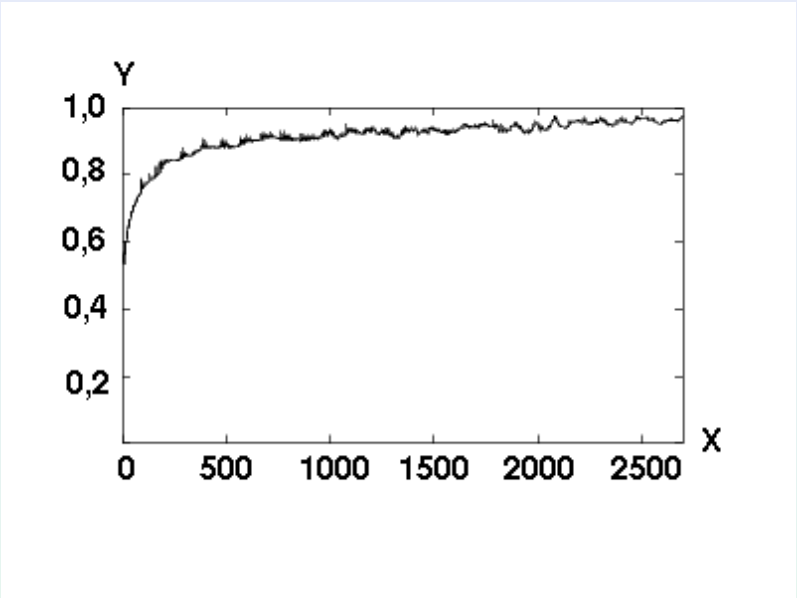
Założenia

- Oparta na metryce XOR.
- Informacje o strukturze sieci są aktualizowane podczas wyszukiwania danych.
- Prawdopodobieństwo, że węzeł pozostanie w sieci jest zależne od czasu pobytu w tej sieci.

Czasy operacji

- Wyszukiwanie klucza — $O(log(n))$
- Podłączenie do sieci — $O(log(n))$
- Odłączenie od sieci — brak akcji

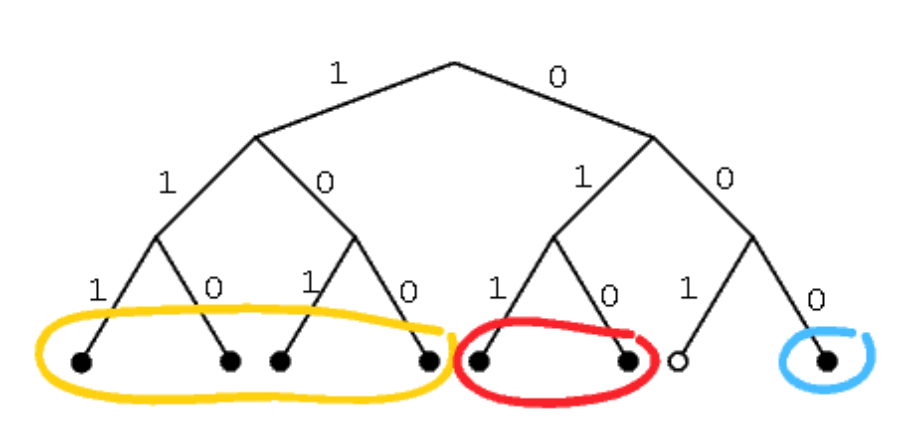
Statystyki dla sieci Gnutella



Rysunek: Prawdopodobieństwo pozostania w sieci

- Prawdopodobieństwo, że dany węzeł pozostanie w sieci kolejne 60 minut zależy od tego jak długo węzeł w danej sieci jest.
- Oś X — czas. Oś Y — prawdopodobieństwo.

Idea tablicy routingu



Rysunek: Drzewo węzłów

- Chcemy mieć kontakty w każdym poddrzewie.
- Dla każdego poddrzewa trzymamy conajwyżej k kontaktów.
- Rozmiar tablicy routingu jest proporcjonalny do ilości poddrzew —$O(log(n))$

Metryka XOR

Definicja

$$d(X, Y) = X \otimes Y$$

Intuicja

Różnice w wyższych bitach bardziej wpływają na odległość niż te w niższych.

$\underline{\mathbf{1}}001\underline{\mathbf{0}}1$
$\underline{\mathbf{0}}0011\underline{\mathbf{1}}$ odległość $= 2^5 + 2^1 = 34$

Interpretacja geometryczna

Węzły w tym samym poddrzewie są bliżej niż węzły w różnych poddrzewach.

Struktury danych

- Informacje o węzłach przechowujemy jako trójki <adres IP, port UDP, ID węzła>
- Dla każdego $0 \leqslant i < 160$ przechowujemy tzw. k-koszyk, czyli listę węzłów.
- W i-tym k-koszyku trzymamy co najwyżej k informacji o węzłach, których odległość od naszego węzła $\in [2^i, 2^{i+1})$.
- Dla małych i k-koszyki będą raczej puste.
- Tablica routingu jest drzewem k-koszyków.
- Informacje są aktualizowane podczas wyszukiwania kluczy.
- Wielkość struktur danych —$O(log(n))$.

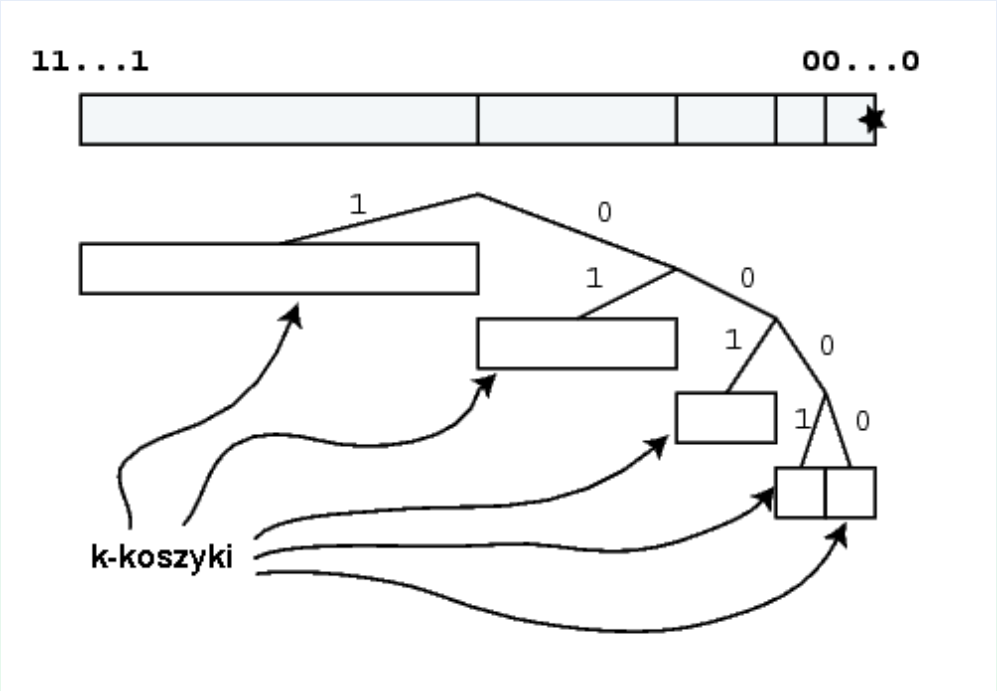
Struktury danych

- Informacje o węzłach przechowujemy jako trójki <adres IP, port UDP, ID węzła>
- Dla każdego $0 \leqslant i < 160$ przechowujemy tzw. k-koszyk, czyli listę węzłów.
- W i-tym k-koszyku trzymamy co najwyżej k informacji o węzłach, których odległość od naszego węzła $\in [2^i, 2^{i+1})$.
- Dla małych i k-koszyki będą raczej puste.
- Tablica routingu jest drzewem k-koszyków.
- Informacje są aktualizowane podczas wyszukiwania kluczy.
- Wielkość struktur danych —$O(log(n))$.

Struktury danych

- Informacje o węzłach przechowujemy jako trójki <adres IP, port UDP, ID węzła>
- Dla każdego $0 \leqslant i < 160$ przechowujemy tzw. k-koszyk, czyli listę węzłów.
- W i-tym k-koszyku trzymamy co najwyżej k informacji o węzłach, których odległość od naszego węzła $\in [2^i, 2^{i+1})$.
- Dla małych i k-koszyki będą raczej puste.
- Tablica routingu jest drzewem k-koszyków.
- Informacje są aktualizowane podczas wyszukiwania kluczy.
- Wielkość struktur danych —$O(log(n))$.

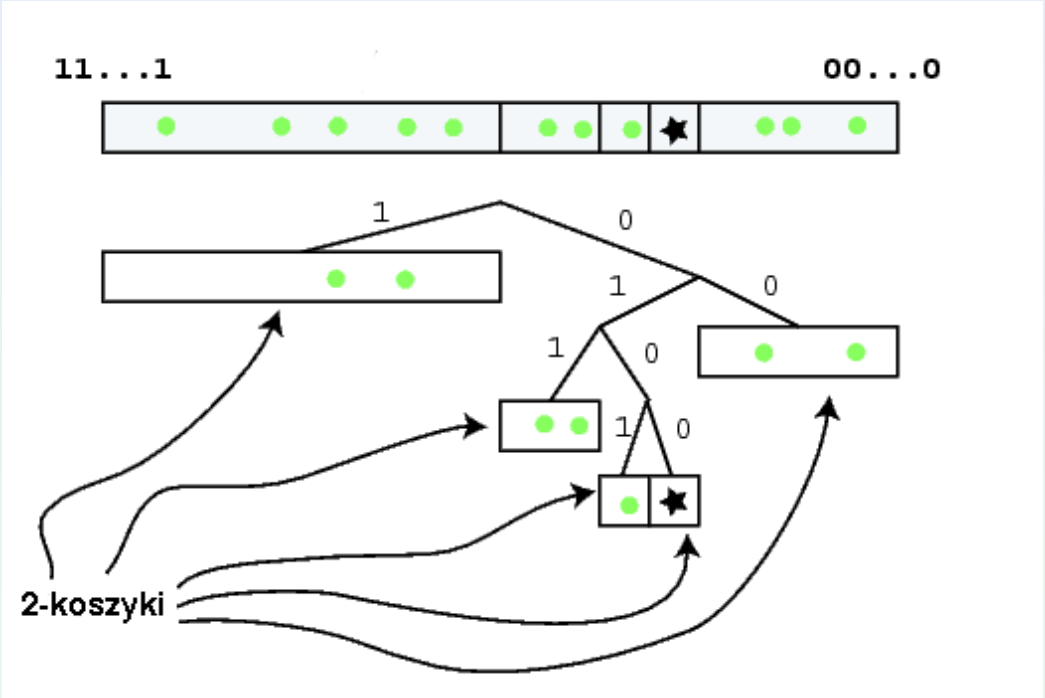
Struktury danych



Rysunek: Tablica routingu

Ogólny schemat tablicy routingu.

Struktury danych



Rysunek: Tablica routingu

Przykładowa tablica routingu dla $k = 2$.

Wyszukiwanie węzła

- Cel: Znalezienie k najbliższych węzłów dla podanego celu $T \in \{0,1\}^{160}$.
- Algorytm rekurencyjny: $findnode_n(T)$ zwraca wszystkie kontakty z pierwszego niepustego k-koszyka, który jest najbliżej T.
- Wyszukiwanie:
  $n_o =$ nasz węzeł
  $N_1 = findnode_{n_0}(T)$
  $N_2 = findnode_{n_1}(T)$ gdzie $n_1 \in N_1$
  ...
  Kończymy, gdy w którymś kroku nie uzyskamy żadnego nowego węzła. Wyszukiwanie można urównoleglić.

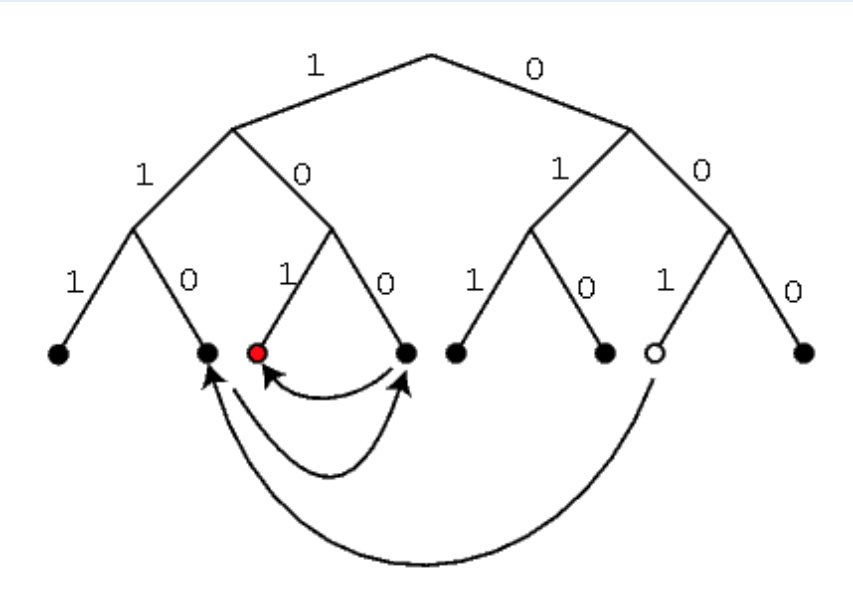
Wyszukiwanie węzła

- Cel: Znalezienie k najbliższych węzłów dla podanego celu $T \in \{0,1\}^{160}$.
- Algorytm rekurencyjny: $findnode_n(T)$ zwraca wszystkie kontakty z pierwszego niepustego k-koszyka, który jest najbliżej T.
- Wyszukiwanie:
  $n_o =$ nasz węzeł
  $N_1 = findnode_{n_0}(T)$
  $N_2 = findnode_{n_1}(T)$ gdzie $n_1 \in N_1$
  ...
  Kończymy, gdy w którymś kroku nie uzyskamy żadnego nowego węzła. Wyszukiwanie można urównoleglić.

Wyszukiwanie węzła

- Cel: Znalezienie k najbliższych węzłów dla podanego celu $T \in \{0,1\}^{160}$.
- Algorytm rekurencyjny: $findnode_n(T)$ zwraca wszystkie kontakty z pierwszego niepustego k-koszyka, który jest najbliżej T.
- Wyszukiwanie:
  $n_o =$ nasz węzeł
  $N_1 = findnode_{n_0}(T)$
  $N_2 = findnode_{n_1}(T)$ gdzie $n_1 \in N_1$
  ...
  Kończymy, gdy w którymś kroku nie uzyskamy żadnego nowego węzła. Wyszukiwanie można urównoleglić.

Wyszukiwanie węzła



Rysunek: Wyszukiwanie klucza

Każdy kolejny skok zbliża nas do szukanego węzła.

Umieszczenie i znajdowanie informacji

Umieszczenie danych

1. Liczymy klucz dla danych (np. przy pomocy SHA-1)
2. Znajdujemy *k* węzłów leżących najbliżej danego klucza.
3. Do każdego z węzłów wysyłamy żądanie umieszczenia pary <klucz, dane>.

Znajdowanie danych o podanym kluczu

1. Wyszukujemy węzły dla celu równego danemu kluczowi.
2. Przy czym w każdym kroku algorytmu dla znalezionych węzłów sprawdzamy czy nie posiadają one szukanych danych.
3. Po znalezieniu danych zapisujemy je dodatkowo w cache'u najbliższego węzła, który tych danych nie posiadał.

Umieszczenie i znajdowanie informacji

Umieszczenie danych

1. Liczymy klucz dla danych (np. przy pomocy SHA-1)
2. Znajdujemy k węzłów leżących najbliżej danego klucza.
3. Do każdego z węzłów wysyłamy żądanie umieszczenia pary <klucz, dane>.

Znajdowanie danych o podanym kluczu

1. Wyszukujemy węzły dla celu równego danemu kluczowi.
2. Przy czym w każdym kroku algorytmu dla znalezionych węzłów sprawdzamy czy nie posiadają one szukanych danych.
3. Po znalezieniu danych zapisujemy je dodatkowo w cache'u najbliższego węzła, który tych danych nie posiadał.

Działania dodatkowe

Ponowna publikacja danych

- Co godzinę każdy węzeł publikuje ponownie dane.
- Dodatkowo wymagane jest ponowna publikacja danych w ciągu 24 godzin przez węzeł, który te dane umieścił w sieci.

Odświeżanie zawartości k-koszyków

Jeżeli otrzymaliśmy wiadomość od węzła w.

- Jeżeli $w \in k$-koszyka przesuwamy go na początek listy.
- Jeżeli $w \notin k$-koszyka bierzemy węzeł v — ostatni z odpowiedniego k-koszyka i pingujemy.
  - Jeżeli v odpowiedział przesuwamy v na początek listy.
  - Jeżeli v nie odpowiedział to usuwamy v z listy i na początek wstawiamy w.

Działania dodatkowe

Ponowna publikacja danych

- Co godzinę każdy węzeł publikuje ponownie dane.
- Dodatkowo wymagane jest ponowna publikacja danych w ciągu 24 godzin przez węzeł, który te dane umieścił w sieci.

Odświeżanie zawartości k-koszyków

Jeżeli otrzymaliśmy wiadomość od węzła w.

- Jeżeli $w \in k$-koszyka przesuwamy go na początek listy.
- Jeżeli $w \notin k$-koszyka bierzemy węzeł v — ostatni z odpowiedniego k-koszyka i pingujemy.
  - Jeżeli v odpowiedział przesuwamy v na początek listy.
  - Jeżeli v nie odpowiedział to usuwamy v z listy i na początek wstawiamy w.

Podłączenie i odłączenie z sieci

Podłączenie do sieci

Musimy posiadać adres jakiegoś istniejącego węzła w.

1. Wstawiamy w do odpowiedniego k-koszyka.
2. Uruchamiamy operacje wyszukiwania z naszym ID jako celem.
3. Odświeżamy zawartość wszystkich k-koszyków. Podczas tej operacji nasz węzeł propaguje się w tablicach routingu innych węzłów.

Odłączenie z sieci

Odłączenie nie wymaga żadnej operacji. Po prostu się rozłączamy.

Podłączenie i odłączenie z sieci

Podłączenie do sieci

Musimy posiadać adres jakiegoś istniejącego węzła w.

1. Wstawiamy w do odpowiedniego k-koszyka.
2. Uruchamiamy operacje wyszukiwania z naszym ID jako celem.
3. Odświeżamy zawartość wszystkich k-koszyków. Podczas tej operacji nasz węzeł propaguje się w tablicach routingu innych węzłów.

Odłączenie z sieci

Odłączenie nie wymaga żadnej operacji. Po prostu się rozłączamy.

Kademlia — podsumowanie

Zalety

- Coraz częściej używane w sieciach P2P.
- Duża wydajność.
- Odporność na awarię węzłów.
- Duża ilość zastosowań.
- Prostota.
- Możliwość równoległego wyszukiwania.

Sieci P2P

Przechowywane informacje

- Dla danego klucza w DHT przechowujemy informacje o węzłach posiadających pliki o danym kluczu.
- Wyszukiwanie plików wg hasha (np. SHA-1)

Wyszukiwanie plików po nazwie

- Nazwa każdego pliku dzielona jest na słowa kluczowe.
- Lista plików zawierających dane słowo kluczowe przechowywana jest w DHT.

Sieci P2P

Przechowywane informacje

- Dla danego klucza w DHT przechowujemy informacje o węzłach posiadających pliki o danym kluczu.
- Wyszukiwanie plików wg hasha (np. SHA-1)

Wyszukiwanie plików po nazwie

- Nazwa każdego pliku dzielona jest na słowa kluczowe.
- Lista plików zawierających dane słowo kluczowe przechowywana jest w DHT.

Przykłady zastosowania

Overnet

Zamknięta implementacja protokołu Kademlia.

- eDonkey
- MLdonkey

Kad Network

- eMule
- MLdonkey
- aMule

Inne

Sporo aplikacji P2P implementuje sieci oparte na Kademlii.

- RevConnect

Przykłady zastosowania

Overnet

Zamknięta implementacja protokołu Kademlia.

- eDonkey
- MLdonkey

Kad Network

- eMule
- MLdonkey
- aMule

Inne

Sporo aplikacji P2P implementuje sieci oparte na Kademlii.

- RevConnect

Przykłady zastosowania

Overnet

Zamknięta implementacja protokołu Kademlia.

- eDonkey
- MLdonkey

Kad Network

- eMule
- MLdonkey
- aMule

Inne

Sporo aplikacji P2P implementuje sieci oparte na Kademlii.

- RevConnect

Cooperative File System — wstęp

Cechy

- System plików tylko do odczytu.
- Identyfikacja systemu plików po kluczu publicznym.
- Duża wydajność.
- Bezpieczeństwo.
- Odporność na awarię węzłów.

System składający się z trzech warstw.

1. Warstwa systemu plików
2. DHash
3. Chord

Cooperative File System — wstęp

Cechy

- System plików tylko do odczytu.
- Identyfikacja systemu plików po kluczu publicznym.
- Duża wydajność.
- Bezpieczeństwo.
- Odporność na awarię węzłów.

System składający się z trzech warstw.

1. Warstwa systemu plików
2. DHash
3. Chord

Trójwarstwowa budowa

Warstwa systemu plików

- Interpretuje pliki jako bloki.
- Tworzy abstrakcję systemu plików dla aplikacji.

DHash

- Odpowiada za przechowywanie bloków.
- Replikuje bloki w sieci.

Chord

- Odpowiada za znajdowanie danych w sieci.
- Przechowuje tablice routingu.

Trójwarstwowa budowa

Warstwa systemu plików

- Interpretuje pliki jako bloki.
- Tworzy abstrakcję systemu plików dla aplikacji.

DHash

- Odpowiada za przechowywanie bloków.
- Replikuje bloki w sieci.

Chord

- Odpowiada za znajdowanie danych w sieci.
- Przechowuje tablice routingu.

Trójwarstwowa budowa

Warstwa systemu plików

- Interpretuje pliki jako bloki.
- Tworzy abstrakcję systemu plików dla aplikacji.

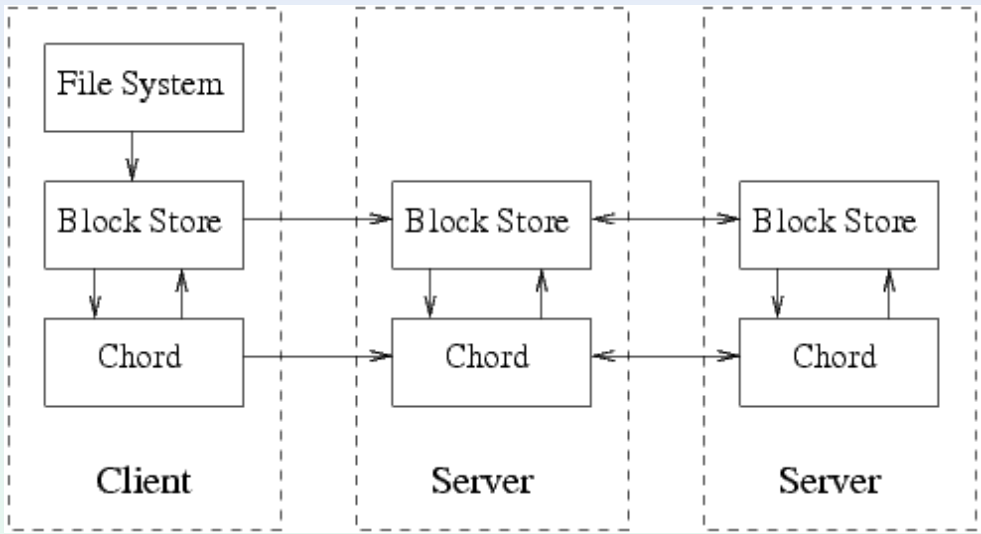
DHash

- Odpowiada za przechowywanie bloków.
- Replikuje bloki w sieci.

Chord

- Odpowiada za znajdowanie danych w sieci.
- Przechowuje tablice routingu.

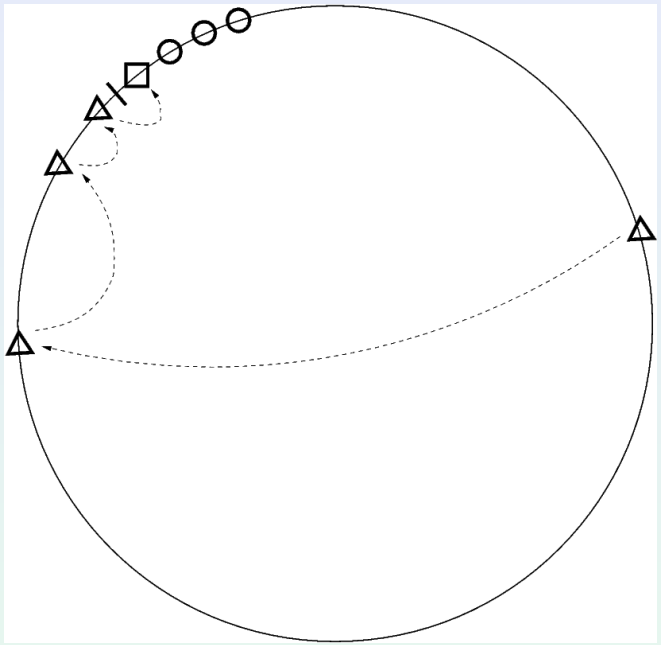
Trójwarstwowa budowa



Rysunek: Schemat budowy

- Połączenia pionowe to wywołania lokalnego API.
- Połączenie poziome to wywołania RPC.

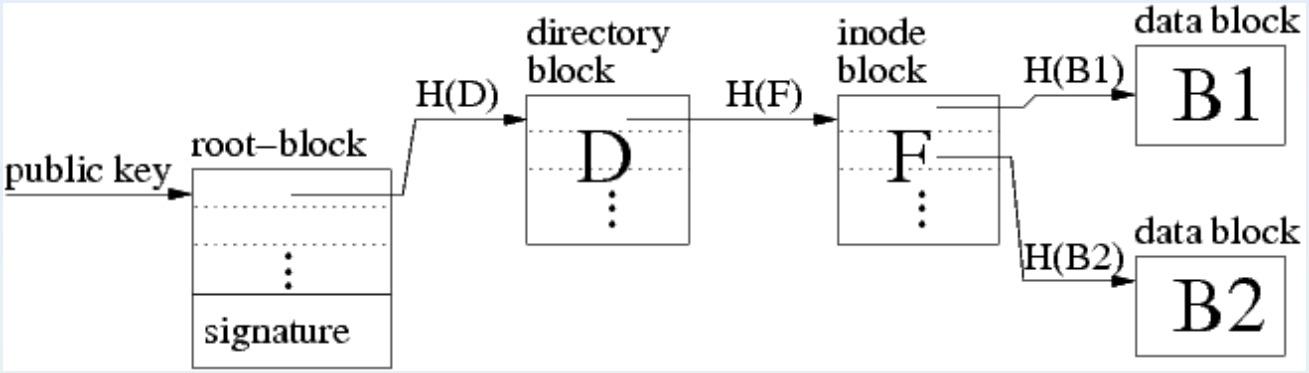
DHash



Rysunek: Umieszczenie kopii bloku

- kwadrat — główna kopia bloku
- kółka — kopie bloku w następnikach węzła z kopią główną
- trójkąty — kopie w cache'ach węzłów na ścieżce wyszukiwania

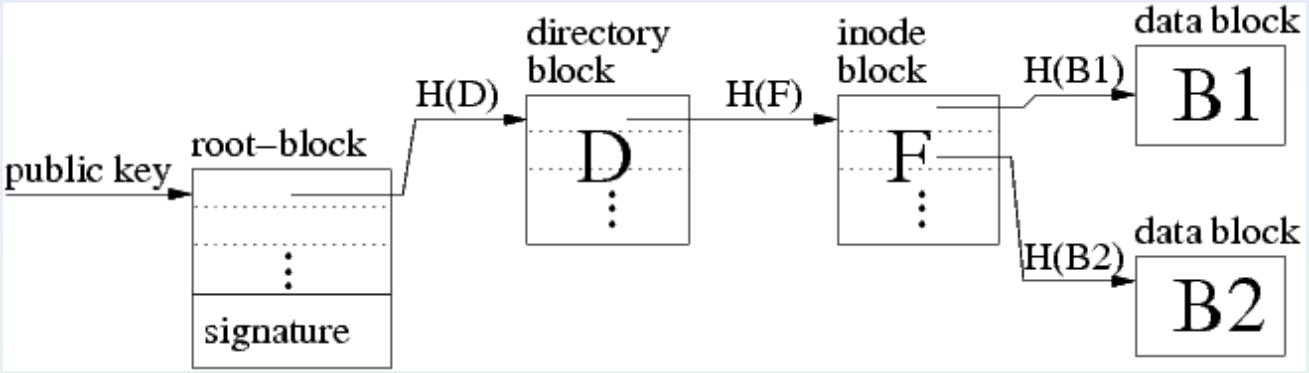
Implementacja



Rysunek: Schemat struktury

- Implementacja oparta jest o Self-certifying File System (SFS).
- Blok główny zawiera w sobie klucz prywatny.
- **Żadna ze sprawdzanych przeze mnie wersji nie działała.**

Implementacja



Rysunek: Schemat struktury

- Implementacja oparta jest o Self-certifying File System (SFS).
- Blok główny zawiera w sobie klucz prywatny.
- **Żadna ze sprawdzanych przeze mnie wersji nie działała.**

Podsumowanie

Cooperative File System

- Praca naukowa, która nie znalazła zastosowania w rzeczywistości.
- Pewnie pozostanie jako ciekawostka.