Les tables de hachage distribuées une introduction

Juliusz Chroboczek
Univerité de Paris 7
jch@pps.jussieu.fr

23 février 2011

Déroulement de HTTP

1. L'utilisateur choisit `http://www.pps.jussieu.fr/~jch/`;
2. résolution des noms : 134.157.168.1 port 80;
3. le client se connecte à 134.157.168.1 et dit
   ```
   GET /~jch/ HTTP/1.1
   Host: www.pps.jussieu.fr
   ```
4. déroulement de la transaction.

Lors de la phase (3), le client et le serveur négocient la ressource de nouveau : l'URL est répétée.

Résolution des noms (2)

La résolution de noms a beaucoup de flexibilité :

- peut retourner plusieurs adresses
  (ex. serveur *multi-homed*) ;
- peut retourner *un sous-ensemble des adresses*
  (ex. DNS round-robin) ;
- peut *en principe* retourner des adresses fausses
  (ex. A et AAAA quand le serveur est IPv4)
  (plus convaincant pour SMTP ?).

Grande liberté dans la conception du protocole.

Techniques de résolution des noms

- Serveur de résolution des noms (HTTP, SMTP, Napster, ed2k, BitTorrent) ;
- *gossip* naïf ;
- *gossip* structuré (Kazaa, Skype, Gnutella) ;
- DHT (eMule, BitTorrent).

Techniques hybrides possibles :

- BitTorrent, eMule : serveur + *gossip* + DHT ;
- Spotify : serveur + *gossip* ;

Serveur de résolution de noms (1)

Serveur de résolution de noms :

- un pair annonce une ressource qu'il sert au serveur ;
- un pair demande au serveur la liste des adresses des pairs qui détiennent la ressource qui l'intéresse.

Avantages :

- simple à implémenter ;
- simple à analyser ;
- rapide.

Serveur de résolution de noms (2)

Défauts :

- vulnérable aux pannes ;
- vulnérable à la censure ;
- vulnérable à la politique.

(Je n'ai pas dit que ça ne passait pas à l'échelle.)

Solutions possibles :

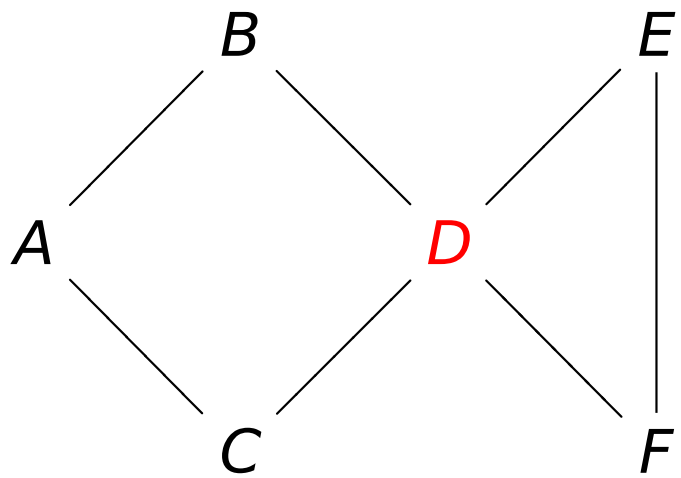
- plusieurs serveurs, choix aléatoire ;
- plusieurs serveurs, choix du plus proche ;
- plusieurs serveurs, distribution de charge.

Remarque : bien fait, ça marche très bien.

Gossip (1)

Gossip :

- chaque pair maintient entre k et l connexions vers des pairs ;
- chaque pair annonce à ses voisins la liste des pairs auxquels il est connecté.

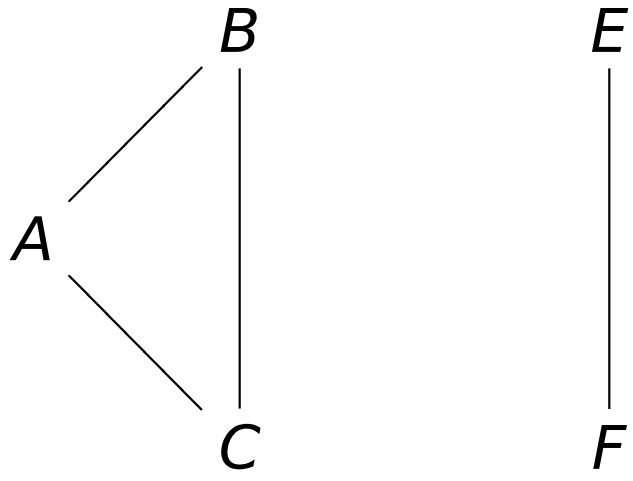


$(k = 2, l = 4)$

Lorsqu'un pair recherche une ressource, il inonde une requête à travers le réseau.

Gossip (2)

Le Gossip pur a tendance a créer des cliques d'« amis » bien connectés entre eux.
Si le réseau devient déconnecté, il est impossible de le reconstituer :



Solution partielle : fixer k grand ($k \geq 100$).

Gossip structuré

Deux catégories de pairs :

- les super-pairs, qui gossipent entre eux ;
- les pairs ordinaires, connectés à un super-pair.

Les super-pairs sont élus dynamiquement, et ont un nombre très important de voisins.
Exemples : Gnutella, Skype.

Rappels : tables de hachage

Une fonction de hachage est une fonction qui est « statistiquement injective » :

si $x \neq y$ alors probablement $h(x) \neq h(y)$

Une table de hachage est une structure où les données sont stockées selon la valeur d'une fonction de hachage :

0	1	h(x)	3	h(y)=h(z)	h(t)	6	7
		x		y,z	t		

DHT

Dans une table de hachage distribuée, la fonction de hachage détermine le pair où la donnée est stockée. La difficulté consiste alors à retrouver le bon pair pour une valeur de hash donnée.
Deux algorithmes dominants :

- Kademlia (Maymounkov et Mazières) ;
- Chord (Stoica, Morris, Karger, Kaashoek et Balakrishnan).

Kademlia

Dans la suite de cet exposé, je vais présenter l'algorithme Kademlia.

> Kademlia : A Peer-to-Peer Information System Based on the XOR Metric. Petar Maymounkov and David Mazières. In *1st International Workshop on Peer-to-peer Systems (IPTPS'02)*. 2002.

- Expérience pratique (BitTorrent, eMule, Overnet, divers botnets) — des centaines de millions de nœuds ;
- intellectuellement satisfaisant (« joli »).

Identificateurs

Dans Kademlia, un réseau de nœuds stocke un ensemble de données.
Typiquement,

- chaque participant au protocole est un nœud ;
- les données sont des adresses.

Les nœuds et les données sont identifiées par des valeurs arbitraires de 160 bits. Typiquement,

- les identificateurs des participants sont tirés au hasard ;
- les identificateurs des données sont des SHA-1.

La métrique XOR

Soient deux ids A et B. On définit

$$\delta(A,B) = \frac{1}{2} \sum_{i=0\cdots159} 2^{-i} \cdot \mathrm{XOR}(A_i, B_i)$$

En binaire,

$$\begin{aligned} 0.A &= 0.01010101010\cdots \\ 0.B &= 0.01000101110\cdots \\ \delta(A,B) &= 0.00010000100\cdots \end{aligned}$$

Propriétés :

- δ est une métrique ;
- la notion de « les k trucs les plus proches de A » est bien définie (à la différence de la métrique d'arbre).

Kademlia : invariants (1)

Les nœuds collaborent pour maintenir deux invariants :

- chaque donnée *d* est stockée dans le nœud le plus proche de *d* ;
- *A* « est voisin de » suffisamment de nœuds vivants pour trouver tout nœeud en temps logarithmique.

« Être voisin de » signifie connaître l'adresse et maintenir un contact régulier (« ping »).

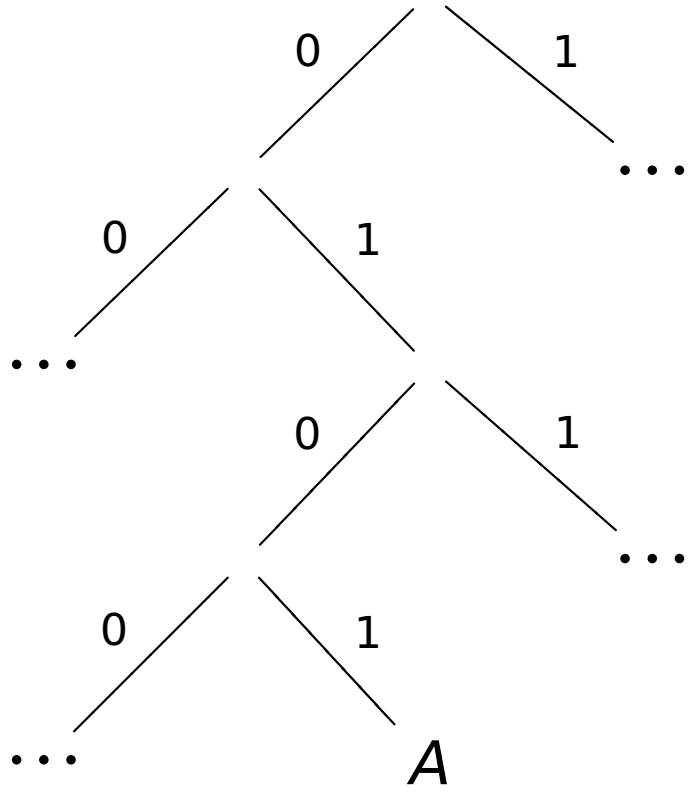
Kademlia : invariants (2)

Pour tout A' préfixe strict de A, soit $A' \cdot b$ qui n'est pas préfixe de A ($b \in \{0, 1\}$).
Alors A est voisin d'un nœud dont $A' \cdot b$ est préfixe (s'il existe).

$$
\begin{aligned}
A \quad = \quad & 01010101010\cdots \\
& 1\cdots \\
& 00\cdots \\
& 011\cdots \\
& 0100\cdots \\
& 01011\cdots \\
& \ldots
\end{aligned}
$$

Kademlia : invariants (3)

A est voisin d'un élément de chacun des « ··· » :



Kademlia : invariants (4)

Conséquence :
Pour tout B, A connaît l'adresse d'un nœud C tel que

$$\delta(C, B) < \delta(A, B)$$

(C est plus proche de B).

De proche en proche, cela permet de trouver B à partir de n'importe quel nœud.

Ébauche de preuve :

$$\begin{aligned} A &= 0101010101010\cdots \\ B &= 0101010xxxxx\cdots \end{aligned}$$

A est voisin de

$$C = 0101010yyyyy\cdots$$

Kademlia : redondance

Comment gérer les nœuds qui tombent en panne ?

Kademlia est k-redondant ($k = 8$ dans BitTorrent) :

- chaque donnée d est stockée dans les k nœuds les plus proches de d ;
- chaque nœud A connaît k nœuds dont $A' \cdot b$ est préfixe.

Cela permet de gérer la panne « simultanée » de $k - 1$ nœuds arbitraires (si vraiment on n'a pas de chance).

Kademlia : messages du protocole

Concrètement, les nœuds envoient les requêtes suivantes :

- ping, qui demande une réponse ;
- find(x), qui demande la liste des k nœuds les plus proches de x connus du destinataire ;
- put(h, d), qui stocke la donnée d de clé h ;
- get(h), qui recquiert la liste des données de clé h stockées dans le destinataire.

Kademlia : table de routage

Chaque nœud maintient une table de routage, qui, pour chaque préfixe A', contient la liste des k voisins correspondants (ainsi que leurs adresses) :

$$A = 0101010101010\cdots$$

ϵ	$1xxxx, 1yyyy\ldots$
0	$00xxxx, 00yyyy\ldots$
01	$011xxxx, 011yyy\ldots$
$\cdots$	$\cdots$

Chaque entrée de la table s'appelle un *bucket*.
Le *bucket* de préfixe le plus long est le *bucket* de A.

Kademlia : table de routage (2)

Insertion d'un nouveau nœud dans la table de routage : insert(B) :

- si le *bucket* correspondant n'est pas plein, on y insère B ;
- si le *bucket* correspondant est plein,
  - s'il contient des nœuds vieux, l'un d'eux est remplacé par B ;
  - sinon, si c'est le *bucket* de A, on le scinde ;
  - sinon, on ne fait rien (*drop*).

(Cette description est une grossière simplification.)

Kademlia : *bootstrap*

Initialement, la table de routage consiste d'un seul *bucket* peuplé par un petit nombre de nœuds.
Ces nœuds de bootstrap sont obtenus d'une façon externe à Kademlia. Par exemple, dans BitTorrent :

- nœuds sauvegardés précédemment ;
- pairs annoncés lors d'un *handshake* BitTorrent ;
- nœuds stockés dans un fichier `.torrent` ;
- nœuds stockés dans le DNS (`dht.transmissionbt.com`, `router.bittorrent.com`, etc.).

Kademlia : maintenance du voisinage

Chaque nœud A maintient périodiquement son voisinage :

$$\text{find}(A) \rightarrow B$$

où B est un élément choisi au hasard du *bucket* de A. La réponse à cette requête permet de (re)peupler le *bucket* de A, et éventuellement de le scinder.

Kademlia : maintenance du réseau

A maintient périodiquement chaque bucket b :

$$\text{find}(i) \rightarrow B$$

où

- i est un identificateur tiré au hasard dans b ;
- B est un élément tiré au hasard dans b.

Si b est vide, B est choisi dans un *bucket* voisin.
La réponse à cette requête permet de repeupler b de nœuds frais, et éventuellement d'en éliminer des vieux.

Kademlia : annonce ou recherche

La recherche d'une donnée d est un algorithme itératif.
L : liste de $l > k$ nœuds connus les plus proches de d, triée par ordre de distance croissante à d.

- L est initialisée à partir de la table de routage ;
- à chaque étape, on envoie find(d) aux éléments de L ; les réponses sont insérées dans L ;
- le processus termine lorsque L est stationnaire.

L contient alors les l nœuds les plus proches de d. On envoie put ou get aux k premiers « bons » éléments de L.

Kademlia : réalisation pratique

La discussion ci-dessus omet un certain nombre de détails essentiels :

- stratégie de *timeout* ;
- maintenance du statut « frais » des nœuds ;
- stratégie d'éviction des nœuds ;
- gestion des nœuds inaccessibles (NAT) ;
- . . .

Conclusion

Kademlia est probablement le seul algorithme distribué non-trivial implémenté à l'échelle de centaines de millions de nœuds. Malgré l'existence de millions de pairs boggués, la DHT de BitTorrent ne s'est toujours pas écroulée.

Cependant, les DHT ne résolvent pas (encore) le problème de la faim dans le monde :

- souvent plus lentes que d'autres solutions ;
- souvent plus complexes que d'autres solutions ;
- souvent plus difficiles à analyser que d'autres solutions.