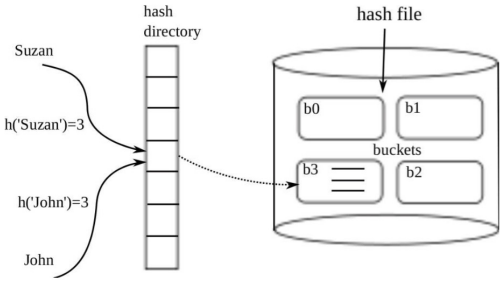
Distributed Hash Table

Nicolas Travers

Introduction au hachage distribué

Fonction de Hachage h(x)

- Placement physique des données
- Recherche de données
- Découpage horizontal :
  - Partitionnement *(buckets)*
  - N partitions => Divise le temps de recherche par N



Introduction - Problème du Hachage statique

Hachage statique

- Nombre de partitions fixe
- Taille des partitions augmente
- Pas d'élasticité dans un contexte distribué

➢Besoin d'un hachage non statique

- Nombre de partitions évolue
- Taille des partitions reste fixe
- Partitions et table de hachage distribuées

Anneau Chord - Distributed Hash Table (DHT)

Nombre de serveurs (virtuels) fixe

- Repose sur des puissances de 2
- Les serveurs sont organisés en anneaux
- Chaque serveur à en charge une partie des données de l'anneau

Table de hachage

- Fonction *modulo* du nombre de serveurs virtuels
- La table est fragmentée sur l'ensemble des serveurs

Utilisé pour DynamoDB (Amazon), Cassandra (Facebook), Redis, Torrent, Blockchain, Git, etc.

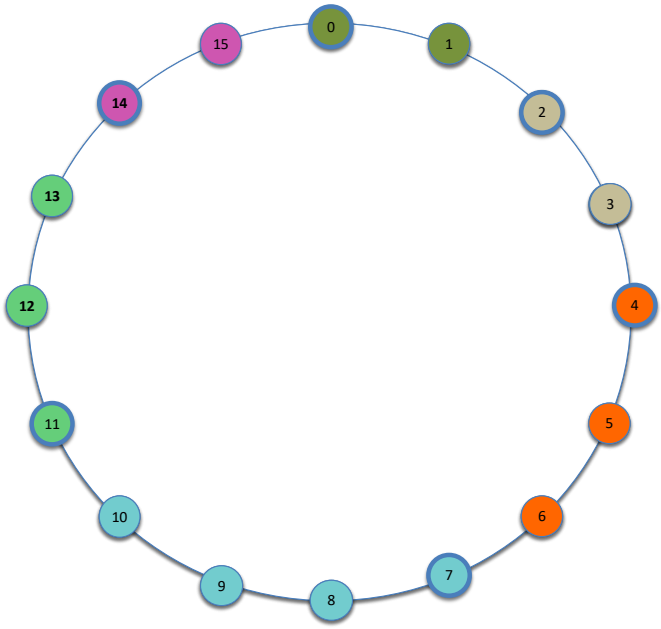
Anneau Chord - Exemple

2^4 serveurs virtuels

- Serveur physique S_i :
  - Données -> $h(S_i)$
  - Données suivants $h(S_i)$ sans serveur physique
    - $h(S_i) < h(k)$

Soit les serveurs physiques suivants :

- 0, 4, 7, 11, 14
- Un nouveau serveur arrive placé en 2



Anneau Chord - Détails

Tolérance aux pannes

- Réplication des données d'un serveur physique sur les 3 serveurs physiques précédents
  - Le serveur précédent a déjà les données en cas de panne
  - Un réplica peut répondre à une requête
- Exemple : S0 est répliqué sur S14, et S11

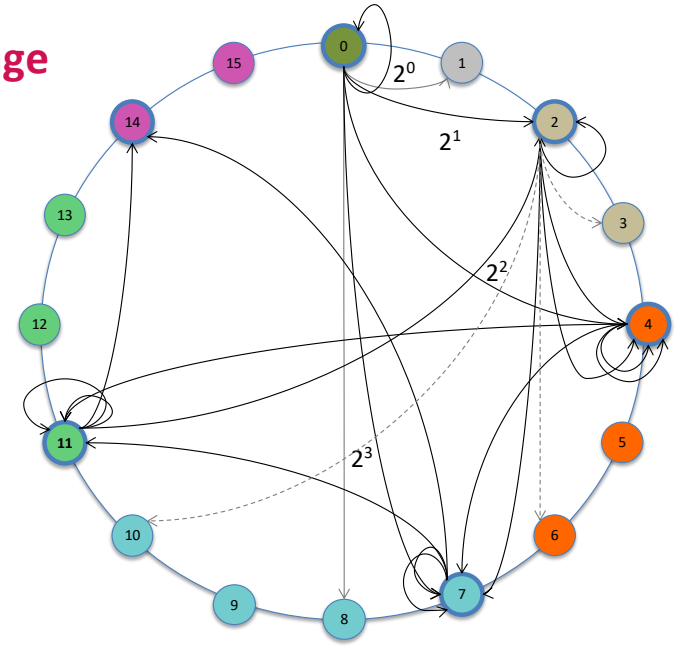
Répartition de charge

- Lorsqu'un serveur physique est ajouté à l'anneau, il est placé virtuellement sur 2 autres nœuds de l'anneau
  - Meilleur équilibrage (cas d'hétérogénéité des données)
  - Données répliquées encore mieux distribuées
- Exemple : S0 est également présent en S5 et S11

Anneau Chord - Table de hachage distribuée

Chaque nœud connaît n nœuds

- Table de hachage distribuée
- Saut de 2^i nœuds, i entre 0 et n-1
- Si $x+2^i$ est *virtuel*
  - on prend le nœud *physique* qui en a la charge (précédent)



Anneau Chord - Commentaires & Algorithme

Table de hachage distribuée

- Maintenance locale
- Distribution de charge
- Pas de contrôleur centralisé

Algorithme de routage

1. Recherche de *v* sur le serveur Si
2. On cherche le serveur x = h(v)
3. Si i est en charge de x ➔ Fin
4. Dans la table de hachage locale
   Si $T(i + 2^0) <= x < T(i + 2^1)$ ➔ Serveur T[0]
   Si $T(i + 2^{j-1}) <= x < T(i + 2^j)$ ➔ Serveur T[j]
   Si $T(i + 2^{n-1}) <= x$ ➔ Serveur T[n-1]

Anneau Chord - Recherche

La valeur 54 est recherchée sur le serveur S_{11}

H(54) = **6**

S_{11}:

~~$T(11+2^0)$= *11 <= 6 < 11* = $T(11+2^1)$~~

~~$T(11+2^1)$= *11 <= 6 < 14* = $T(11+2^2)$~~

~~$T(11+2^2)$= *14 <= 6 < 2* = $T(11+2^3)$~~

$T(11+2^3)$= ***2 <= 6***

S_2:

~~$T(2+2^0)$= *2 <= 6 < 4* = $T(2+2^1)$~~

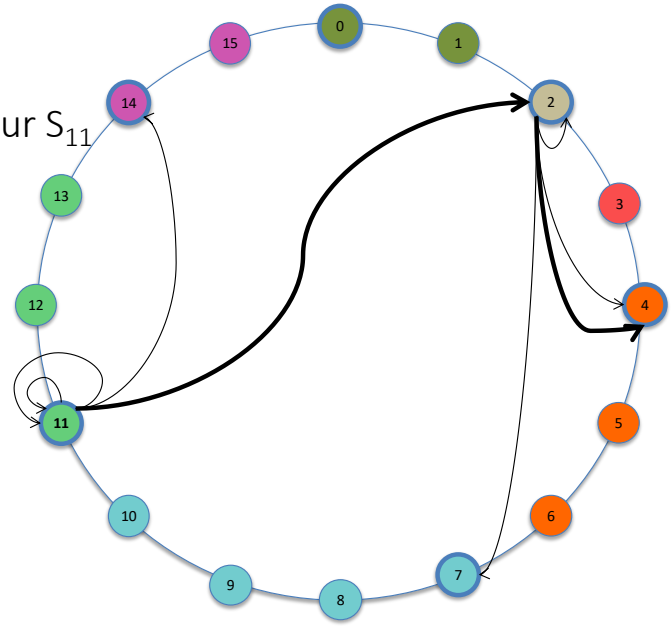
~~$T(2+2^1)$= *4 <= 6 < 4* = $T(2+2^2)$~~

$T(2+2^2)$= ***4 <= 6 < 7*** = $T(2+2^3)$

Cout maximum de l'algorithme

$O(\log(2^n))$ ➔ $O(n)$

n ➔ Nombre de nœuds actuel



Anneau Chord - Ajout/Suppression de noeuds

Ajout :

S_i contacte S_j, allocation dans l'anneau (nœud i)

Table de routage :

S_j utilise sa table de routage pour trouver les nœuds composant la table de routage de S_i

Les nœuds de l'anneau doivent être mis à jour

Contact de $i - 2^0$, $i - 2^1$, $i - 2^2$, $i - 2^3$ et $i - 2^4$

Si absent, contacter le nœud physique suivant

Données :

Récupération des données du serveur précédent : S_{i-1}

Valeurs *v* de S_{i-1} dont : $h(S_i) <= h(v)$

Suppression :

Les réplicas (3 prédécesseurs) garantissent la sortie d'un nœud

Mise à jour des tables de routages

Anneau Chord - Conclusions

Table de hachage distribuée (DHT)

- Pas de table de hachage globale
- Efficace : O(n)
- Réplication pour la tolérance aux pannes
- Auto-gestion