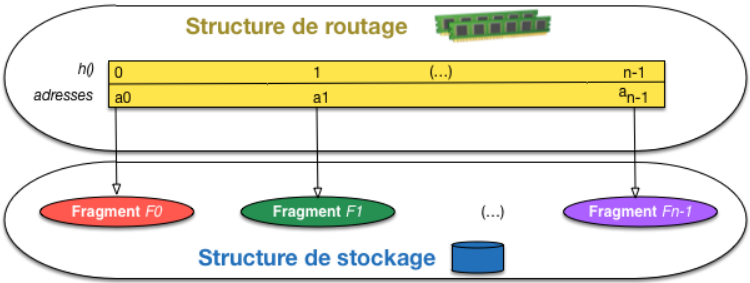
Bases de données documentaires et distribuées, http://b3d.bdpedia.fr

Partitionnement par hachage

Principe du partitionnement par hachage

On prend la clé ; on l'interprète comme un entier, on calcule le reste r de la division par N : le document est affecté au fragment r.



Simple mais... Fonctionne très bien pour N, le nombre de fragments, **fixe**.

Les opérations

Implantation très simple et très efficace.

- $get(i)$: calculer $r = h(i)$, et accéder au fragment dont l'adresse est a_r, chercher le document en mémoire ;
- $put(i, d)$: calculer $r = h(i)$, insérer d dans le fragment dont l'adresse est a_r ;
- $delete(i)$: comme la recherche, avec effacement du document trouvé ;
- $range(i, j)$: **pas possible avec une structure par hachage**, il faut faire un parcours séquentiel complet.

Elasticité

Si N change : tout devient faux.

Le problème

Tout repose sur l'hypothèse que la fonction $h()$ est **immuable**.

Soit $N = 5$, donc $h(i) = i \mod 5$. Soit un flux continu d'insertion et de recherche de documents, parmi lesquelles l'insertion, puis la recherche de l'identifiant 17.

- on effectue *put(17, d)*, d est inséré dans F_2 (tout le monde suit ?) ;
- les insertions continuent, jusqu'à la nécessité d'ajouter un sixième fragment : maintenant $h(i) = i \mod 6$.
- on effectue $get(17)$, on cherche dans F_5 (vous voyez pourquoi ?) qui ne contient pas d.

Pas de solution simple ! !

Systèmes distribués basés sur le hachage

Idée simple : tout le monde utilise le même hachage, les “blocs” sont les serveurs.

Le système est nécessairement **dynamique** (ajout de serveurs), donc on ne peut pas utiliser une fonction de hachage basée directement sur N, qui varie.

En 1997, un article a proposé une solution maintenant très largement utilisé : le **hachage cohérent** ou *Consistent haching*.

- D'abord conçu et utilisé pour des **caches distribués** (*memcached*)
- Popularisé pour la gestion de données par le système Dynamo (Amazon, 2007)
- Maintenant intégré à de très nombreux systèmes : Voldemort, Riak, Cassandra, ...

Hachage cohérent (*consistent hashing*)

On prend une valeur de N très grande et immuable, p.e. 2^{64}

- la fonction de hachage $h(v) = v \mod N$ est **fixe**, appliquée aux clés **et** aux serveurs (identifiées par leur IP) vers l'espace d'adressage $A = [0, 2^{64} - 1]$;
- A est organisé comme un anneau parcouru dans le sens des aiguilles d'une montre ;

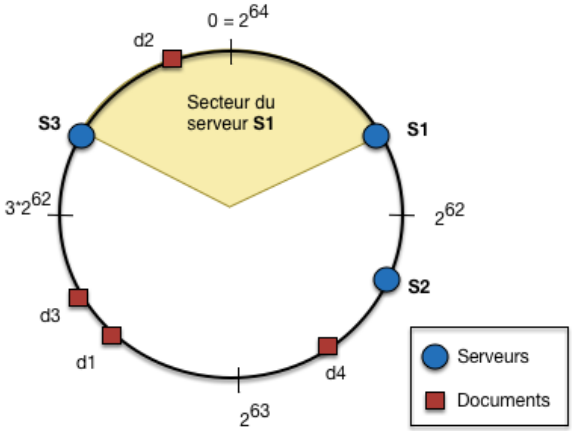
Serveurs et document sont un emplacement dans A défini par h. **Il n'y a aucune chance qu'un serveur ait la même valeur de hachage qu'un serveur, d'où**

Régle d'affectation des documents aux serveurs

Si S et S' sont deux serveurs adjacents sur l'anneau, toutes les clés de l'intervalle $]h(S), h(S')]$ appartiennent à S'.

Illustration

Serveurs **et** documents sont placés sur l'anneau.

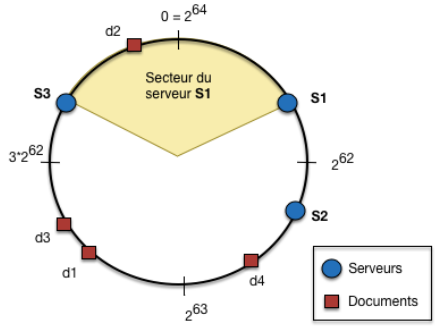


Notez le déséquilibre des fragments (pour l'instant).

La table de hachage

Elle établit une correspondance entre le découpage de l'anneau en arcs de cercle, et l'association de chaque arc à un serveur.

Intervalle	Serveur
$]c, a]$	S1
$]a, b]$	S2
$]b, c]$	S3



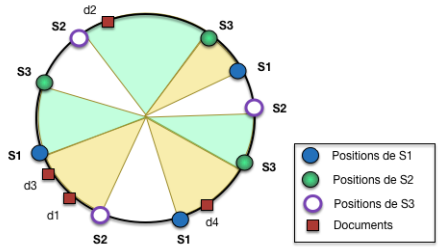
Quelques détails vraiment utiles

Comment fonctionne la tolérance aux pannes ? Comment fonctionne l'équilibrage ?

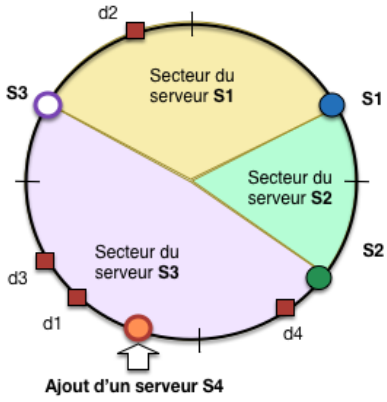
Panne ⇒ géré par la réplication (surprise !) ; par exemple on copie sur la machine suivante dans l'anneau, etc.

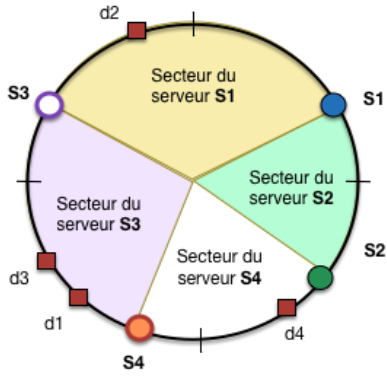
Equilibrage ⇒ un même serveur (physique) est distribué en plusieurs points (virtuels) sur l'anneau.

- plus il y a de points, plus le serveur recevra de données ;
- en cas de panne, réoganisation répartie plus justement.



Ajout et suppression de serveurs



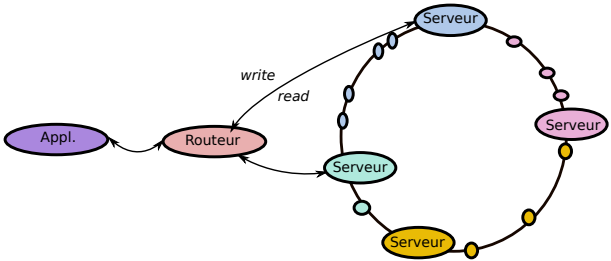


Une réorganisation **locale** est suffisante (S_3 transmet à S_4).

Mais où est donc la table de hachage ?

Qui nous dit quelle est la liste des serveurs et leur intervalle ?

Il nous faut un **routeur**.



Autres solutions : (i) chaque nœud est un routeur (acceptable quand on n'ajoute/retire pas de serveurs tout le temps), (ii) le routeur est intégré au client.