Conception des systèmes répartis

Les mémoires géantes à clé-valeur ou encore « l'approche NoSQL »

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

21 novembre 2012

plan

Plan

Origine

Les besoins du Web

- Gestion de données persistantes (bases de données) géantes de services Web : Google, Amazon, Facebook, etc
- SGBD relationnelles trop complexes et bloquantes : abandon du paradigme ACID
- Contourner le théorème CAP de N. Lynch et S. Gilbert (voir conjecture de E. Brewer) : seules, deux des trois contraintes suivantes peuvent être assurées en réparti :
  - Cohérence des données (vue uniforme),
  - Disponibilité,
  - Résistance au partitionnement.
- Approche simplifiée : (grand) tableau associatif dont les éléments sont des enregistrements (clé-valeur)

Plan

Principes de conception

La transparence . . .

☞ S'attaque aux niveaux supérieurs de transparence :

- transparence de la réplication,
- transparence des fautes
- transparence des changements d'échelle

Objectifs

- Priorité à la disponibilité et à la résistance au partitionnement
- Cohérence faible impliquant résolution des conflits
- Aspect Temps réel : le temps de réponse est important
- Résistance aux pointes de charge
- Adaptabilité par rapport à l'architecture répartie (scalability)

Type de mémoire

Les mots de ce genre de mémoire . . .

☞ Orientation objet : enregistrer la forme linéarisée d'objets (à la Java, C++,etc) accessible via une clé.

Les opérations

- L'insertion (l'écriture) d'une nouvelle paire :
  $\langle$clé, valeur (*de l'objet linéarisé*)$\rangle$
- La lecture d'une valeur en fournissant la clé

Exemple

- **get** (clé) : renvoie la valeur et un contexte (version)
- **put**(clé, valeur, contexte)

Plan

Les grandes lignes

Les contraintes

- Modèle de données : tableau associatif (clé,valeur) ;
- Modèle NoSQL : on ne garde que AD du ACID ;
- L'écriture doit toujours être possible ;
- Efficacité : 99.9% des requêtes doivent être traitées en moins de 300 ms ;
- Tolérance aux fautes : il y a presque toujours des sites en panne ;
- Transparence d'échelle : adaptabilité de l'architecture.

Les principes de conception

Objectif : disponibilité avant tout

- Réplication avec cohérence faible et résolution des conflits lors des lectures ;
- Approche « pair à pair » de type Schord pour la répartition des réplicas et des sites ;
- Résistance au partitionnement : les lectures et écritures peuvent continuer.

Architecture : un choix de technologies

Face au problème :

- du partitionnement : hachage des sites offrant une adaptabilité incrémentale ;
- de l'insertion sans blocage : horloges vectorielles avec résolution des conflits lors des lectures ;
- de gestion des fautes temporaires : quorums « souples » et indications sur la localisation des réplicas résistant à la perte de réplicas ;
- de redémarrage suite à des fautes permanentes : arbres de Merkle pour resynchroniser des réplicas divergents ;
- d'appartenance à l'anneau : protocole par rumeurs ;
- de détection de faute : orienté détecteur de faute.

Plan

Les systèmes pair à pair

Révolutions ?

- Premiers systèmes vraiment répartis ?
- Remise en cause du modèle client/serveur :
  tout pair est client ET serveur
- Essai de suppression de serveurs « centralisateurs »
- Quelques protocoles : Gnutella, eDonkey, Bitorrent, FreeNet,. . .
- Quelques applications célèbres : Kazaa, mlMac, eMule, Gnucleus, MLDonkey, . . .

Problème de base

La localisation efficace DU site (d'un site) qui possède la donnée (une copie de) cherchée SANS serveur de noms centralisé

Le protocole de localisation : Schord

Definition

Protocole définissant la primitive :

IP-Address lookup(Key which)

Cette primitive renvoie l'adresse IP du noeud ayant la donnée repérée par la clé fournie en paramètre.

Principes de base

- Données et sites sont identifiées par des clés de m bits.
- Une fonction de hachage attribue cet identificateur à partir des adresses IP pour les sites et de noms pour les données
- Les identificateurs sont ordonnés circulairement modulo 2^m (Schord ring)

Affectation des données aux sites

Placement

Domaine des identificateurs $[0 : 2^6[$

K_i : clé donnée, N_j : clé site

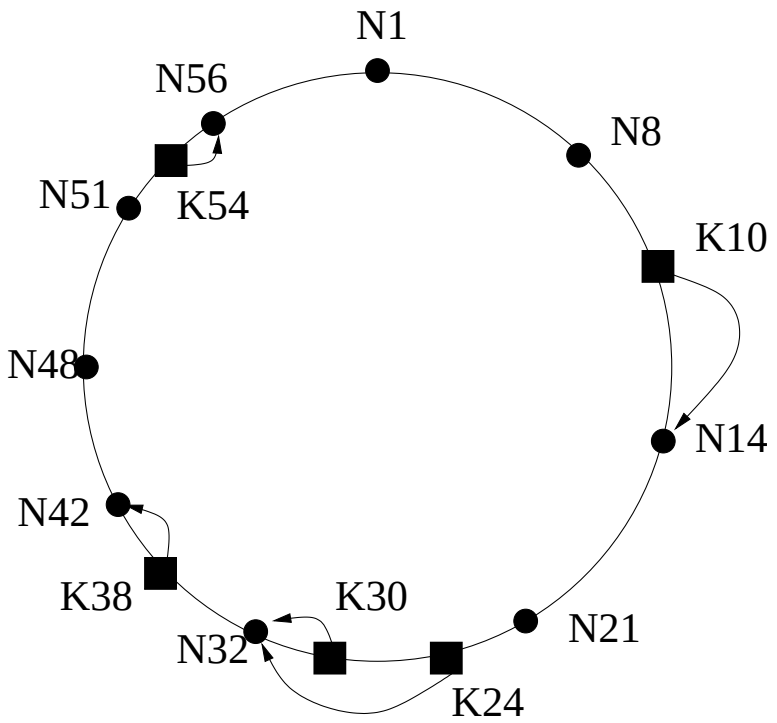
Fonction *suc* :

- $suc(N_i) = N_j$ tel que : $\nexists N_k, i < k < j$
- $suc(K_i) = N_j$ tel que : $\exists k : suc(N_k) = N_j \wedge k < i < j$

Exemple

$suc(N21) = N32$

$suc(K30) = N32$



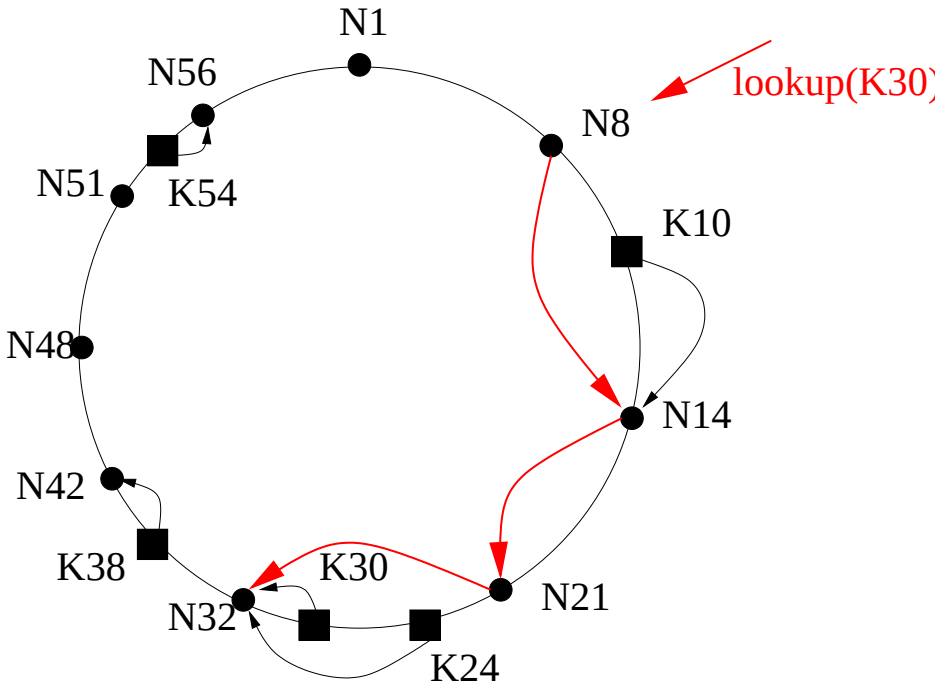
Algorithme de la primitive lookup

Algorithme simple

- Chaque site connaît son successeur
- Appel par RPC

Exemple

N8.lookup(K30)



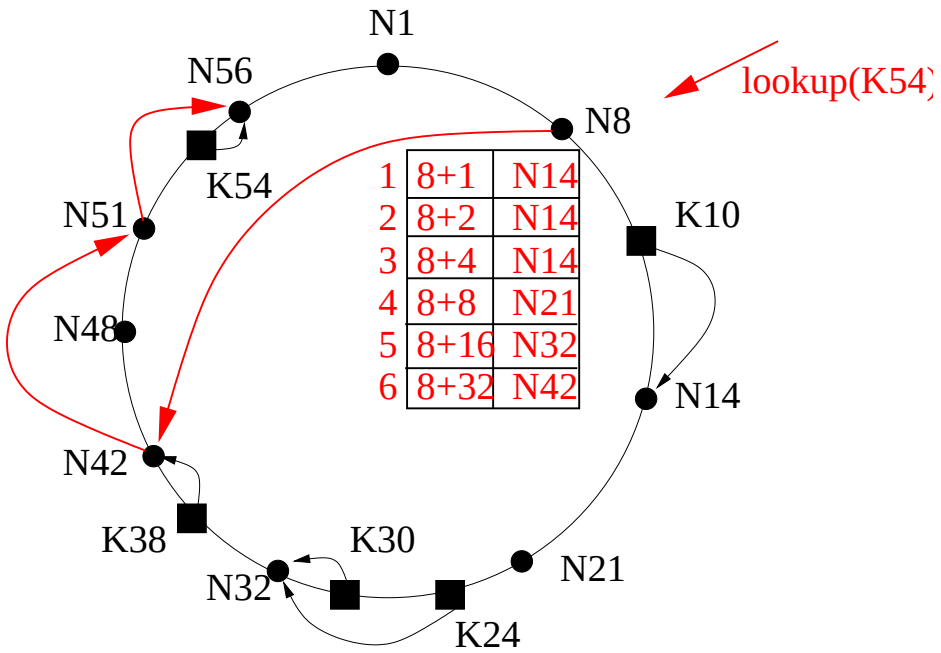
Algorithme de la primitive lookup

Algorithme optimisé

- Chaque site a une table d'index
- Réduit le nombre d'appels RPC

Exemple

N8.lookup(K54)



Avantages et inconvénients...

Quelques difficultés

- Insertion/retrait de sites $\Rightarrow$ déplacement de données
- Nécessité d'une fonction de hachage cohérente : elle répartit équitablement les données et les sites.

Quelques avantages

- Vrai algorithme réparti : pas de serveur central
- Passage à l'échelle (tables de routage)

Plan

Conclusion

Des « sur-couches »

- Cassandra (Facebook : gestion de la messagerie)
- BigTable (Google)

Extensions proposées

- Modèle de données plus complexe ;