

Gestion répartie de données - 2

Diffusion à grande échelle Techniques P2P

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)
<http://sardes.inrialpes.fr/~krakowia>

Passage à grande échelle

- **Qu'est-ce que le passage à grande échelle (*large scale*) ?**
 - ◆ Croissance du nombre de sites, d'objets, d'utilisateurs, etc.
 - ◆ Croissance de la taille géographique, de la complexité (nb d'interactions)
 - ❖ Une propriété souvent présente : *organisation dynamique*
- **Capacité de croissance**
 - ◆ La *capacité de croissance* (*scalability*) est l'aptitude, pour un système, à maintenir ses qualités (performances, adaptabilité, robustesse, etc.) lorsque le système passe à grande échelle
 - ◆ Exemple : à l'échelle de l'Internet, un algorithme linéaire (croissance en $O(n)$) est considéré comme inefficace ($\log(n)$ est acceptable)
- **Ce qu'il faut éviter**
 - ◆ Les structures centralisées
 - ◆ Les algorithmes qui supposent que tous les sites sont en état de marche
 - ◆ L'usage de la diffusion générale
 - ◆ L'hypothèse d'une structure statique du système

2

Un outil : les réseaux de recouvrement

- **Réseau de recouvrement (*overlay network*)**
 - ◆ Un réseau réalisé "au-dessus" d'un réseau existant
 - ◆ Objectif : réaliser un "réseau virtuel" ayant certaines propriétés spécifiées
 - ◆ Moyen : utiliser une couche d'application pour réimplanter certaines fonctions du réseau ou rajouter d'autres fonctions
 - ❖ Réimplanter le *roulage*
 - ❖ Rajouter une fonction de *stockage*
 - ◆ Avantages
 - ❖ Grande souplesse de conception (on maîtrise le niveau application)
 - ❖ Indépendance par rapport au réseau physique sous-jacent (avantages de la virtualisation)
 - Mais on peut aussi exploiter des propriétés du réseau sous-jacent (proximité physique, etc.).
 - ❖ Exemples : l'Internet lui-même (DNS, TCP, etc.), les réseaux de distribution de contenu, les systèmes P2P (cf plus loin)

3

Diffusion à grande échelle (1)

- **Limites des algorithmes "classiques" de diffusion**
 - ◆ Les algorithmes de diffusion vus jusqu'ici s'appliquent à un ensemble bien identifié de processus
 - ◆ Les contraintes fortes sur leurs spécifications font que ces algorithmes passent très mal à grande échelle
- **Nouveaux problèmes**
 - ◆ On souhaite utiliser la diffusion pour des applications dans lesquelles
 - ❖ le nombre de processus (ou sites) est inconnu et variable, ainsi que la topologie d'interconnexion
 - ❖ le nombre de sites est potentiellement très grand (taille de l'Internet)
 - ❖ les sites jouent un rôle symétrique (chacun est client et serveur)

4

Diffusion à grande échelle (2)

■ Champs d'application

- ◆ Découverte et observation de ressources
- ◆ Collecte de données (surveillance d'installations)
- ◆ Détection de pannes
- ◆ Mise à jour de bases de données à grande échelle

■ Voies d'approche

- ◆ On utilise des algorithmes **probabilistes** (un site a une probabilité élevée de recevoir l'information)
- ◆ Le principe utilisé est la **propagation aléatoire** ; analogies : propagation d'une rumeur (*gossip*) ou d'une épidémie (algorithmes dits "épidémiques")

5

Bref historique

La **diffusion épidémique** a été introduite en 1987 (Demers et al.) pour la mise à jour de bases de données.

Deux idées : diffusion à un sous-ensemble de destinataires choisis **au hasard** ; anti-entropie (**échange mutuel**), comme complément

Elle a trouvé récemment (2000) un regain d'intérêt avec le développement des systèmes de grande taille, des systèmes pair à pair et des réseaux de capteurs

Exemple récent d'application : Astrolabe

A. Demers, et al. Epidemic Algorithms for Replicated Database Maintenance, *Proc. 6th ACM Symp. on Principles of Distributed Computing*, Vancouver, pp. 1-12, 1987

R. van Renesse, K. P. Birman, W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed Systems Monitoring, Management and Data Mining. *ACM Trans. on Computer Systems*, 21(2), May 2003

6

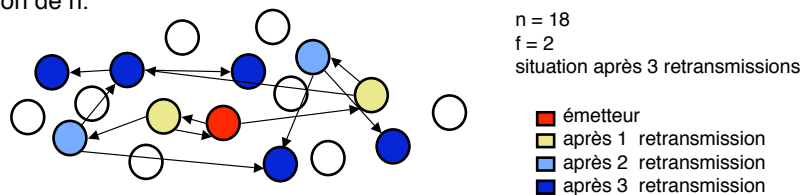
Diffusion épidémique

Principe

Chaque processus participe à la diffusion. Les paramètres sont :

- capacité b : nombre de messages qu'un processus peut stocker
- nombre de retransmissions t : nombre de phases de rediffusion
- nombre de destinataires f : à chaque phase, le processus transmet le message à f destinataires choisis aléatoirement

Les paramètres b , t , f caractérisent un algorithme de diffusion épidémique. Ils peuvent être fixés indépendamment du nombre total n de processus, ou être fonction de n .



7

Diffusion épidémique : aspects mathématiques

Les algorithmes épidémiques comportent une succession de tours. Soit n la taille de la population, $Y(r)$ le nombre d'individus touchés ("infectés") après le r -ième tour. Alors, si un individu infecté continue la propagation à **tous les tours**, on a

$$p = Y(r)/n \approx 1/(1 + n \exp(-fr)) \quad (f = \text{fanout, taille de l'ensemble de diffusion})$$

Donc la proportion d'individus touchés augmente **exponentiellement** avec le nombre de tours, et avec f .

Si un individu touché ne propage l'infection que pendant **un tour** ("infect and die"), alors la proportion p satisfait l'équation

$$p = 1 - \exp(-pf)$$

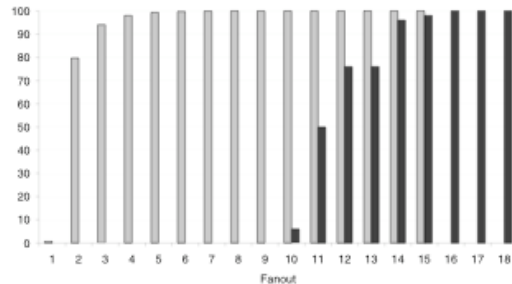
Dans ce modèle, supposons que $f = \log(n) + c$ (c est un paramètre) Alors la probabilité que **tous** les processus soient touchés à terme est

$$P = \exp(-\exp(-c))$$

Il y a un changement de comportement quand c change de signe, donc quand f dépasse $\log(n)$

8

Diffusion épidémique : comportement



■ Proportion of nodes reached in non atomic broadcast
■ Proportion of atomic broadcast
 Total number of processes : 50 000

Comportement bimodal : le comportement change à partir d'une certaine valeur du nombre f de processus destinataires d'une retransmission (*fanout*). Une diffusion est dite atomique (dans ce contexte) si elle touche l'ensemble des processus. On constate que la proportion de diffusions atomiques croît rapidement à partir d'une certaine valeur de f , et qu'elle est de 100% pour une valeur de f suffisamment élevée.

Source : P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, L. Massoulié. From Epidemics to Distributed Computing, *IEEE Computer*, 37(5), 60-67, May 2004

9

Algorithmes de diffusion épidémique : exemple (1)

```

// Initially:
events, eventIds := ∅
every T ms:
  // Update ages
  for all e ∈ events do
    e.age := e.age + 1
  for all e ∈ events: e.age > k do
    events := events \ {e}
// Gossip
gossip.events := events
Choose f random members target1, ..., targetf
for all j ∈ [1...f] do
  send(targetj, gossip)
upon receive(gossip):
  // Update events and ages
  for all e ∈ gossip.events
    if not (e.id ∈ eventIds) then
      events := events ∪ {e}
      eventIds := eventIds ∪ {e.id}
      deliver(e)
  if e' ∈ events such that
    e.id = e'.id and e'.age < e.age then
      e'.age := e.age
  // Garbage collect eventIds
  while leventIds > leventIdsm do
    remove oldest element from eventIds
  // Garbage collect events
  while levents > leventsm do
    remove oldest element from events
  
```

k = nombre max propagations
 f = "fanout"
 constantes

P. Th. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, P. Kouznetsov, Lightweight Probabilistic Broadcast, *Proc. IEEE Int. Conf. on Dependable Systems and Networks (DSN'01)*, 2001

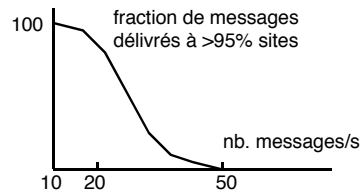
10

Algorithmes de diffusion épidémique : exemple (2)

Commentaires sur l'algorithme

Le paramètre sensible est la **taille des tampons** pouvant contenir les messages. Un processus doit pouvoir conserver les messages suffisamment longtemps pour pouvoir les retransmettre.

Si la fréquence d'émission augmente (la taille des tampons restant fixe), les tampons se remplissent plus vite et les messages sont éliminés avant d'être retransmis suffisamment de fois pour être largement diffusés



D'où l'idée d'un algorithme adaptatif : régler la fréquence d'émission en fonction des ressources et de la charge

L. Rodrigues, S. Handurukande, J. Pereira, R. Guerraoui, A.-M. Kermarrec, Adaptive Gossip-based Broadcast, *Proc. IEEE Int. Conf. on Dependable Systems and Networks (DSN'03)*, 2003

11

Conclusion sur la diffusion épidémique (1)

Champ d'application

- Systèmes de grande taille, avec évolution dynamique
- Connaissance mutuelle limitée entre les membres
- Rôles symétriques (pair à pair)

Avantages de la diffusion épidémique

- Facilité de réalisation et de mise en place
- Robustesse (vis-à-vis des modifications de l'environnement)
- Grande tolérance aux défaillances
- Pas de reconfiguration pour la reprise
- Réalisation possible d'une large gamme de comportements
- Ajustement via un petit nombre de paramètres

Limitations

- Nécessite des ressources en mémoire
- Pas de garanties strictes (seulement probabilistes)

12

Conclusion sur la diffusion épidémique (2)

Expérience acquise

Expérience encore réduite, pour le moment

Quelques problèmes ouverts

Connaissance mutuelle des membres

Gestion des tampons de stockage de messages

ces deux aspects conditionnent notamment le passage à l'échelle

Structuration du réseau et utilisation de ses caractéristiques

Diffusion sélective (filtrage)

13

Systèmes "pair à pair" (*peer to peer*, P2P)

■ Idée de base

- ◆ utiliser la capacité globale de l'ensemble des **clients** potentiels comme support des fichiers
- ◆ ne garder aux "serveurs" que le rôle de répertoire (et fonctions de gestion, distribution de logiciel), et non plus la fonction de stockage - à la limite faire disparaître la notion de serveur séparé
 - ❖ chaque nœud devient alors client et serveur

■ Réalisations

- ◆ des systèmes d'échange de fichiers dans une communauté large
 - ❖ **Napster, Gnutella, Freenet, etc.**
- ◆ des systèmes à base de groupes restreints (dynamiques)
 - ❖ **Groove**
- ◆ des algorithmes efficaces de hachage distribué : **Chord, CAN, etc.**
- ◆ extension vers le calcul sur grilles (mise en commun de ressources de calcul)

14

Systèmes P2P : classification (1)

■ Selon le degré de centralisation

- ◆ **Contrôle centralisé**
 - ❖ **Un serveur central permet la mise en correspondance des pairs**
 - Avantage : structure simple
 - Inconvénients : maîtrise centrale, capacité de croissance limitée, point unique de défaillance
- ◆ **Totalement décentralisé**
 - ❖ **Tous les nœuds jouent un rôle symétrique (clients et serveurs)**
 - Avantage : pas de maîtrise centrale (confidentialité) ; disponibilité
 - Inconvénients : peu gérable ; performances non garanties
- ◆ **Partiellement décentralisé**
 - ❖ **Un ensemble (dynamiquement variable) de nœuds joue un rôle privilégié**
 - Avantages et inconvénients : essaient de concilier les 2 précédents

15

Systèmes P2P : classification (2)

■ Selon l'organisation du réseau (virtuel)

- ◆ **Non structuré**
 - ❖ **Le placement des données n'est pas lié à la topologie du réseau**
 - Avantage : adapté à un ensemble de nœuds très dynamique
 - Inconvénient (si pas de contrôle central) : recherche aléatoire (donc aussi temps de réponse) ; gaspillage de bande passante -> capacité de croissance ?
- ◆ **Structuré**
 - ❖ **Les données sont placées en des points prédéterminés pour rendre la recherche déterministe**
 - Avantage : recherche rapide, temps d'accès borné
 - Inconvénients : peu adapté à une population dynamique
- ◆ **Faiblement structuré**
 - ❖ **Solution intermédiaire ; recherche partiellement déterministe**

16

Systèmes P2P : classification (3)

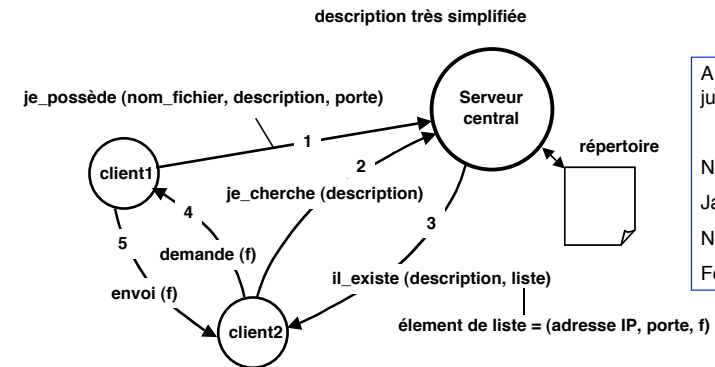
	•Non structurés	•Faiblement structurés	•Structurés
•Contrôle centralisé	•Napster		
•Partiellement décentralisé	•Kazaa, Gnutella		
•Totalemment décentralisé	•Gnutella (initialement)	•Freenet	•Chord, Can, Tapestry

Gnutella : <http://www.gnutella.com>
 Freenet : <http://www.freenetproject.org>
 Chord : <http://www.pdos.lcs.mit.edu/chord/>

Un article de synthèse : A survey of peer to peer file sharing technologies
<http://www.eltrun.aueb.gr/whitepapers/#3rd>
 Le tableau ci-dessus en est extrait

17

Exemple de système pair à pair : Napster



A existé de janvier 1999 à juillet 2001

Nombre d'utilisateurs :
 Jan. 2000 : 1 000 000
 Nov. 2000 : 23 000 000
 Fev. 2001 : 50 000 000

si porte = 0, le détenteur du fichier est derrière un pare-feu. Le fichier sera envoyé en mode *push* (son détenteur est averti par le serveur)

Optimisation : le client mesure la qualité de la connexion (ping) pour plusieurs sites de la liste et choisit celui qui a les meilleures performances à ce moment

18

Exemple de système pair à pair : Gnutella

■ Structure

- ◆ Contrôle décentralisé, réseau non structuré
- ◆ Chaque nœud est à la fois client, serveur et routeur

■ Protocole

- ◆ Construit sur IP - 4 types de messages
- ◆ **Ping** : Un hôte sonde le réseau pour acquérir des correspondants
 - ❖ utilisé lors d'une connexion initiale ou d'une réinsertion
- ◆ **Pong** : Réponse à **Ping** - indique (adr. IP, porte), nb et taille des fichiers de celui qui répond
- ◆ **Query** : Recherche - indique clé de recherche (par ex. mots clés), et débit mini requis
- ◆ **Query Hit** : Réponse à **Query**- indique (adr. IP, porte), débit, nb de fichiers trouvés.
- ◆ Quand un hôte a reçu **Query Hit**, il peut directement communiquer avec le correspondant indiqué.

19

Gnutella : technique de recherche

■ Principe de base

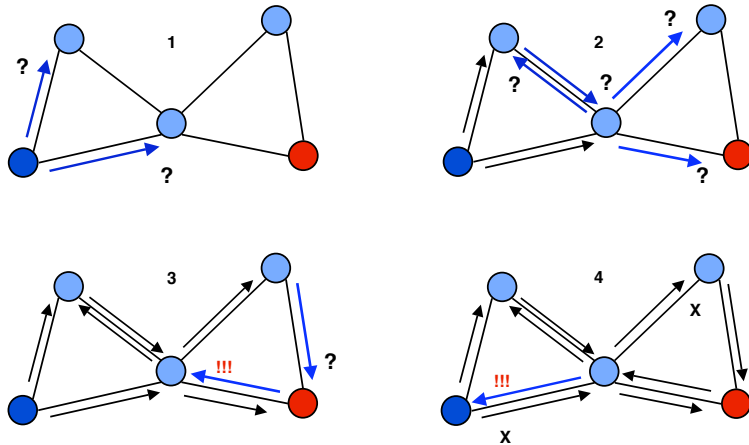
- ◆ Fonctionne par inondation (*flooding*)
- ◆ Une requête (**Query**) est transmise à un certain nombre de voisins, qui eux-mêmes la retransmettent à leurs voisins, etc.
- ◆ Les requêtes ont une identification unique, pour éviter les retransmissions en boucle
- ◆ Le nombre de retransmissions (TTL) est limité
- ◆ Malgré ces précautions, il y a beaucoup de messages superflus...

■ Améliorations

- ◆ Remplacer inondation par envoi aléatoire, effectué en parallèle (diminue le nombre de messages)
- ◆ Utiliser des "Super Serveurs" (un nœud ayant beaucoup de connexions et une puissance importante). Ces Super Serveurs changent dynamiquement et sont connectés entre eux

20

Gnutella : exemple de recherche



21

Gnutella : quelques problèmes

- **Problème de croissance**
 - ◆ Le TTL (limite au nombre de retransmissions) est nécessaire pour limiter le nombre de messages mais il limite aussi l'“horizon”
- **Problèmes des “parasites”**
 - ◆ Il est possible d'utiliser Gnutella comme “consommateur uniquement”, sans apporter de fichiers (*freeloading*)
 - ◆ Remède : interdire les connexions via HTTP (navigateur) ; donner priorité aux clients qui apportent beaucoup
- **Problème des connexions interrompues**
 - ◆ En particulier clients/serveurs ayant des connexions lentes (modem)
 - ◆ Remède : mieux annoncer charge courante ; privilégier les connexions via super-serveurs
 - ◆ Fonctionnement encore imparfait (une fraction élevée des chargements est interrompue)

22

Kazaa

- **Un système très utilisé, mais très mal connu**
 - ◆ Fraction importante du trafic de l'Internet (40 à 50% ???)
 - ◆ Code propriétaire, organisation cachée, distribuée, et non identifiée
- **Architecture**
 - ◆ Essaie de combiner les avantages de Napster et de Gnutella
 - ◆ Organisation hiérarchique (“Super- nœuds” (SN), environ 30 000), connectés entre eux (30 à 150 connexion par SN)
 - ◆ Chaque SN est responsable de 100 à 150 nœuds (fils)
 - ◆ Chaque SN fonctionne comme un Napster local (recueille le contenu et les adresses de ses “descendants”)
- **Fonctionnement**
 - ◆ Une requête est transmise à un SN
 - ❖ S'il peut la satisfaire, OK (un nombre x de réponses est requis)
 - ❖ Sinon, il la transmet à un SN voisin, et ainsi de suite
 - ❖ Performances (latence) non garanties
 - ❖ Bonne robustesse, redondance des SN
 - ◆ Connexion initiale : fournit une liste de SN

23

Exemple de système pair à pair : Freenet

- **Système expérimental**
 - ◆ Déploiement initial en cours : <http://www.freenetproject.org>
- **Structure**
 - ◆ Totalement décentralisé, faiblement structuré
 - ◆ Les nœuds ont une table de routage (contrairement à Gnutella)
- **Propriétés**
 - ◆ Meilleure capacité de croissance que Gnutella
 - ◆ Anonymat des clients et des serveurs
- **Protocole**
 - ◆ 4 types de messages
 - ◆ **Data request** - clé
 - ◆ **Data Reply** - Fichier
 - ◆ **Data failed** - Adresse, cause de l'échec
 - ◆ **Data insert** - clé, Fichier

24

Freenet : principe d'organisation

■ Stockage des données

- ◆ Un fichier est stocké en plusieurs exemplaires, à des emplacements déterminés par sa clé
- ◆ Chaque nœud possède une table de routage contenant les adresses d'un ensemble d'autres nœuds et une liste (indicative) des clés détenues par chacun de ces nœuds
- ◆ La recherche se fait de proche en proche : si un nœud ne détient pas le fichier cherché, il transmet la requête à un successeur : celui qui possède (dans la table de routage) la clé "la plus proche" de la clé recherchée

■ Clés

- ◆ 2 sortes de clés
 - ❖ De bas niveau, pour le stockage - créées par hash-code sur contenu
 - ❖ De haut niveau, pour l'usage humain - pointent vers clés de bas niveau
 - ❖ Les clés de haut niveau sont créées à partir de descriptifs des fichiers à rechercher (ex : mots-clés)

25

Freenet : fonctionnement des clés

■ Clés de bas niveau (*Content Hash Key* : CHK)

- ◆ Utilisées pour stocker les fichiers
- ◆ Clé = SHA-1 (contenu du fichier)

■ Clés de haut niveau (*Signed Subspace Key* - SSK)

- ◆ Utilisées pour stocker des descripteurs (listes de CHK)
- ◆ Une SSK est relative à un espace de noms (sous-ensemble de fichiers).
 - ❖ Cet espace a un propriétaire et est protégé par un couple (clé publique, clé privée)
- ◆ H1=SHA(clé publique) ; H2=SHA(descriptif) ;
- ◆ SSK=SHA(H1 concaténé à H2)
- ◆ Le fichier est signé avec la clé privée (fichier + {SHA(contenu)}_{clé privée})
- ◆ Un client peut recréer une SSK avec le couple (clé publique, descriptif), mais seul le propriétaire peut créer un fichier

26

Freenet : stockage

■ Insertion d'un fichier

- ◆ Le propriétaire envoie à un nœud la clé (CHK ou SSK) et un TTL (nombre maximum de sauts, également nb max de copies)
- ◆ Le nœud vérifie si la clé existe déjà
 - ❖ Si oui, collision (le fichier ou le descripteur est déjà présent)
 - ❖ Si non, le nœud retransmet au nœud possédant la clé la plus proche dans la table de routage
 - ❖ Si le TTL est atteint sans collision, un message OK est renvoyé
 - ❖ Si OK, le propriétaire envoie le fichier, qui est stocké sur tous les nœuds du parcours précédent

27

Freenet : recherche

■ Recherche d'un fichier à partir de sa clé

- ◆ Envoi (clé + TTL) à un nœud
 - ❖ Si la clé est présente, le nœud renvoie le fichier
 - ❖ Si la clé n'est pas présente, le nœud transmet la clé à celui possédant la clé la plus proche dans la table de routage
- ◆ On continue jusqu'à ce que le fichier soit trouvé ou le TTL atteint
 - ❖ En cas de "cul de sac" (pas de successeur), on revient en arrière
 - ❖ Idem en cas de détection de boucle

28

Systèmes structurés

■ Principe

- ◆ Table de hachage répartie (DHT : *Distributed Hash Table*)
- ◆ L'emplacement du stockage est entièrement déterminé
- ◆ Une seule opération de base : clé => emplacement

■ Exemples

- ◆ Chord, Can, Tapestry
- ◆ Utilisent des tables de routage, avec un algorithme pour une recherche efficace (coût de recherche : $\log N$ pour N nœuds)

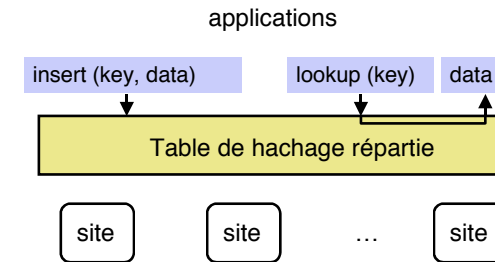
■ Caractéristiques

- ◆ Bonne capacité de croissance
- ◆ Mais peu dynamique (insertion, suppression, déplacement coûteux)
- ◆ Doivent faire leurs preuves sur applications en vraie grandeur

29

Tables de hachage réparties (DHT)

Chaque site gère une partie de l'espace de hachage : il est responsable d'un certain intervalle de clés
pas de connaissance globale centralisée
pas de point unique de défaillance
passage à l'échelle
répartition uniforme des ressources



30

Un exemple de DHT : Chord

Algorithme de base : trouver le nœud responsable d'une clé donnée
fonction : $key \rightarrow node$

Les clés et les nœuds reçoivent un **identificateur** de m bits
identificateur nœud = hachage de son adresse IP
identificateur clé = hachage de la clé

La fonction de hachage doit avoir les "bonnes" propriétés (faible risque de collision, inversion impossible). Exemple : SHA-1

Les identificateurs sont ordonnés modulo 2^m (ordre circulaire)

Correspondance entre nœud et clé : la clé k est associée au premier nœud dont l'identificateur est égal ou supérieur à celui de la clé k

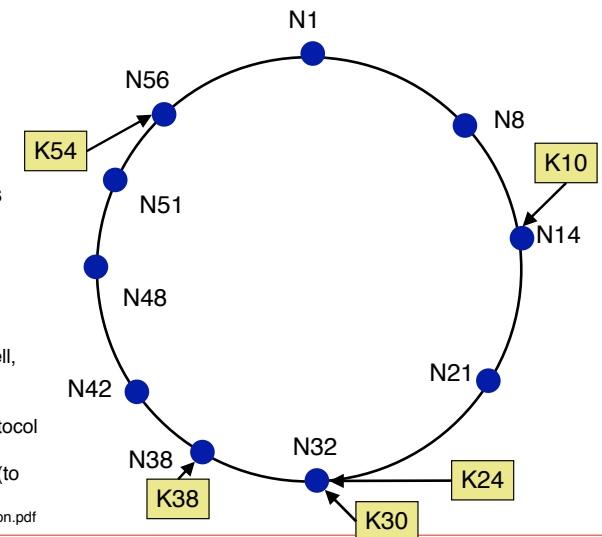
Informations : <http://www.pdos.lcs.mit.edu/chord/>

31

Chord : exemple

$m = 6$, $2^m = 64$
10 nœuds, 5 clés

L'association évolue dynamiquement en cas d'addition ou de retrait des nœuds et des clés



I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications, *IEEE-ACM Trans. On Networking* (to appear)
<http://pdos.csail.mit.edu/chord/papers/paper-ton.pdf>

32

Chord : propriétés

Soit N le nombre de nœuds et K le nombre de clés. On peut montrer que, avec une probabilité élevée :

Chaque nœud est responsable d'au plus $(1 + O(\log N))K/N$ clés

Quand un nœud rejoint ou quitte le réseau, les modifications touchent $O(K/N)$ clés (associées au nœud considéré)

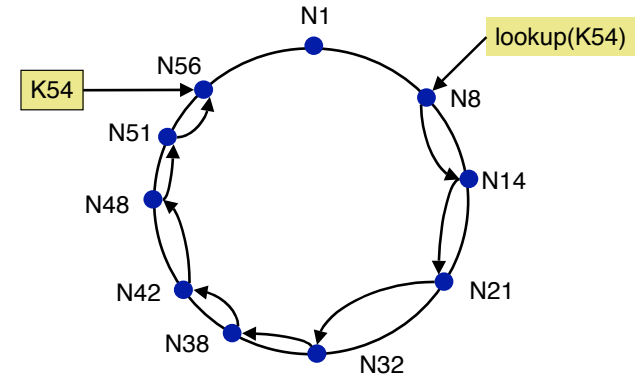
Le terme de "probabilité élevée" signifie que ces résultats sont valables pour une distribution de la fonction de hachage sans anomalie majeures (pas de concentration de collisions).

33

Chord : recherche d'une clé (1)

Méthode simple, peu rapide : recherche séquentielle

Chaque nœud doit connaître son successeur. Étant donné une clé, on parcourt la suite des nœuds jusqu'à trouver une paire de nœuds qui "encadre" la clé. Le dernier est le nœud cherché.

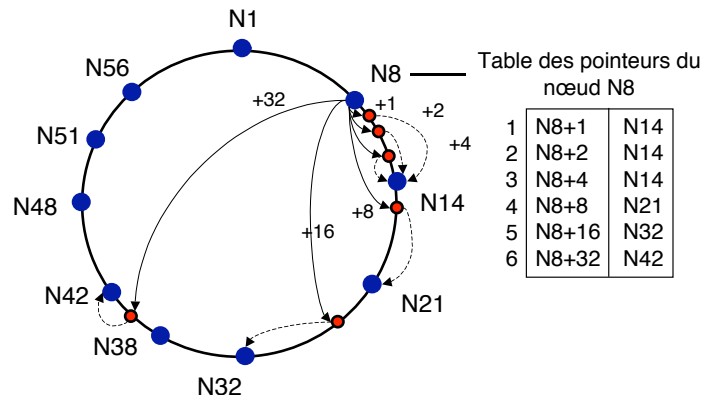


34

Chord : recherche d'une clé (2)

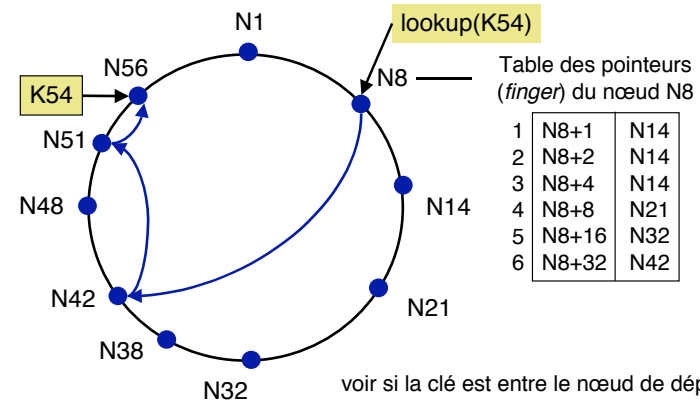
Chaque nœud a une table des pointeurs (*finger*)

$\text{finger}[k] = \text{premier nœud qui suit } (n + 2^{k-1}) \bmod 2^m, 1 \leq k \leq m$



35

Chord : recherche d'une clé (3)



voir si la clé est entre le nœud de départ et son successeur
sinon chercher dans la table des pointeurs le nœud le plus proche précédant la clé (ici 42)
rappeler la procédure de recherche à partir de ce nœud

36

Chord : performances de la recherche

Avec une probabilité élevée (i.e. si la fonction de hachage réalise une dispersion aléatoire des clés et nœuds), le nombre de nœuds à interroger pour la recherche du nœud successeur d'une clé dans un réseau de N nœuds est $O(\log N)$

Le nombre d'entrées dans les tables de pointeurs peut être réduit à $O(\log N)$ (au lieu de m , nombre de bits des clés)

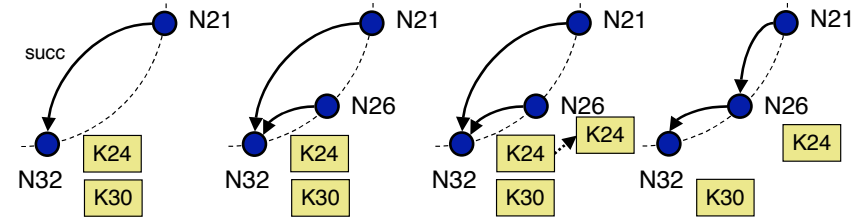
Le nombre moyen de messages pour réaliser une recherche est de l'ordre de $0,5 \log N$

37

Chord : arrivée et départ de nœuds (1)

Les tables des pointeurs doivent être tenues à jour pour tenir compte des nœuds qui rejoignent ou qui quittent le réseau. Cela se fait en tâche de fond. Chaque nœud interroge périodiquement son successeur pour déterminer le prédécesseur de ce dernier, ce qui permet de détecter une insertion

Quand un nœud rejoint le réseau, il calcule sa clé (par hachage) et s'adresse à un nœud quelconque pour déterminer son successeur (procédure de recherche de clé). Puis la procédure ci-dessus permet de mettre à jour les pointeurs successeur et prédécesseur de chaque nœud



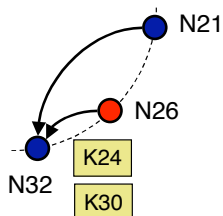
38

Chord : arrivée et départ de nœuds (2)

Lors d'une insertion, il faut mettre à jour les tables des pointeurs du nœud qui rejoint le réseau. des autres nœuds

Pour créer une table des pointeurs pour le nouveau nœud : travail fait par le **prédécesseur** de ce nouveau nœud, qui calcule les entrées pour toutes les distances 2^i

Exemple : insertion du nœud N26 : c'est N21 qui va faire le travail



N21 cherche N26+1 (N27) -> N32
 N21 cherche N26+2 (N28) -> N32
 N21 cherche N26+4 (N30) -> N32
 N21 cherche N26+8 (N34) -> N38
 N21 cherche N26+16 (N42) -> N42
 N21 cherche N26+32 (N58) -> N1

Table des pointeurs (finger) du nœud N26

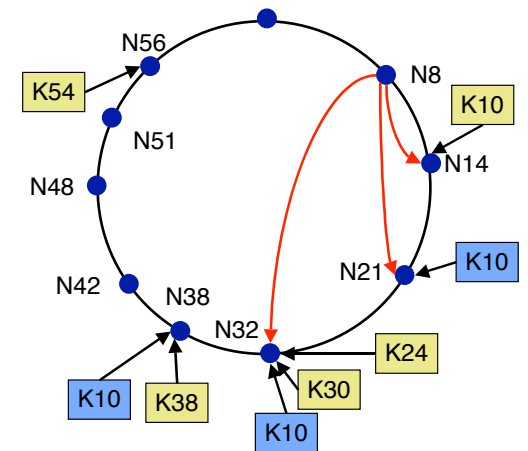
1	N26+1	N32
2	N26+2	N32
3	N26+4	N32
4	N26+8	N38
5	N26+16	N42
6	N26+32	N1

39

Chord : techniques de redondance

Pour améliorer la tolérance aux fautes :

- Chaque nœud connaît non seulement son successeur, mais ses r **successeurs**
- Chaque clé est dupliquée sur r **nœuds successifs**



40

Conclusion sur Chord

Technique de structuration de l'espace des informations utilisable dans un système P2P

Performances en log N (N nombre de nœuds du réseau), donc bonne capacité de croissance.
Dépend néanmoins des qualités de la fonction de hachage

Permet l'insertion et le départ de nœuds en cours de fonctionnement

Résiste bien aux défaillances de nœuds

Il reste à valider ces principes sur de grandes applications réelles

41

Conclusion sur systèmes P2P

■ Systèmes non structurés

- ◆ Ex : Gnutella
- ◆ Problème : capacité de croissance, anonymat

■ Systèmes structurés

- ◆ Freenet (faiblement), Chord etc. (fortement)
- ◆ Problème : insertion dynamique

■ Pour tous les systèmes

- ◆ Difficile d'assurer la qualité de service (qui la garantit ?)
 - ❖ Sécurité
 - ❖ Disponibilité - différente de celle du client-serveur (une partie du réseau est en général en état de marche)
- ◆ Pas de standard pour le moment

■ Domaine de recherche très actif

42

Références sur systèmes P2P

Gnutella : <http://www.gnutella.com>

Freenet : <http://www.freenetproject.org>

Chord : <http://www.pdos.lcs.mit.edu/chord/>

Un article de synthèse : A survey of peer to peer file sharing technologies
http://www.eltrun.aueb.gr/whitepapers/p2p_2002.gr

Une présentation détaillée : P2P systems (Keith W. Ross, Dan Rubenstein)
<http://cis.poly.edu/~ross/papers/P2PtutorialInfocom.pdf>

43