# Recherche d'informations à grande échelle dans des architectures Peer-to-Peer

Bruno DEFUDE
Dept Informatique
Institut National des Télécommunications



# Plan

- Introduction
- P2P pur : GNUTELLA
- Super-peers
  - SuperPeers
  - Kazaa
- P2P « structuré »
  - Chord
  - P-Grid
  - Autres (Freenet, CAN, Pastry)
- P2P sémantique
  - SON
  - Routing Indices
  - Padoue
- Synthèse

# P2P

- Internet des débuts était un système P2P
  - N'importe quel couple d'ordinateurs pouvaient s'envoyer des paquets (pas de firewalls, pas de communications asymétriques, …)
  - Machines servent indifféremment de client ou de serveur
  - Approche coopérative
- Exemples typiques : Usenet News et DNS



# Définition P2P

- Chaque nœud participant peut être client et serveur
- Chaque nœud paye sa participation en donnant accès à une partie de ses ressources
- Propriétés :
  - Pas de coordination centralisée
  - Pas de BD centralisée
  - Aucun nœud n'a une vision globale du système
  - Comportement global émerge à partir des interactions locales
  - Tous les services et données sont accessibles de n'importe quel nœud
  - Nœuds sont autonomes
  - Nœuds et connections sont non fiables

## Classes de systèmes P2P

- P2P Hybrides (e.g Napster)
  - Index centralisé (non tolérant aux fautes)
  - Échange d'information direct
- P2P « Purs » (e.g Freenet, Gnutella)
- P2P Hiérarchiques ou « super-peers » (e.g Kazaa)
  - Mélange C/S et P2P
- P2P sémantiques (e.g SON)
  - P2P « pur » avec routage basé sur une information sémantique



## Fonctionnalités d'un système P2P

- Découverte de ressources 
- Gestion des mises à jour
- Passage à l'échelle
- Tolérance aux fautes
- sécurité

# Classes d'applications

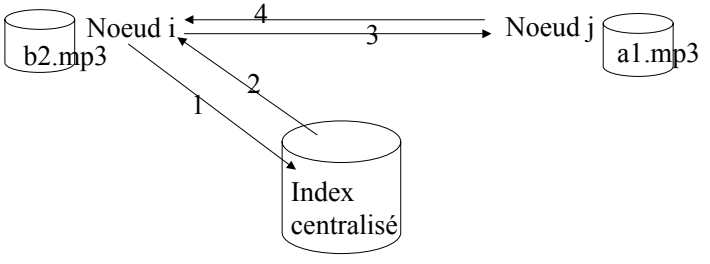
- Partage de fichiers : Napster, Gnutella, Freenet, KaZaa
- Système de stockage persistant à grande échelle : OceanStore
- Grid computing : Seti@home



# Critères de comparaison

- Recherche de ressource :
  - Topologie du réseau : ouverte (Gnutella) ou contrôlée (Chord)
  - Placement des données et méta-données : libre (Gnutella) ou dirigé (Chord)
  - Routage des messages : fonction de choix des successeurs
- Besoins applicatifs :
  - Expressivité du « langage de requêtes » : égalité (Chord), préfixe (P-Grid), SQL (?), …
  - Complétude des résultats : une réponse, toutes les réponses, les k meilleures réponses, …
  - Autonomie des nœuds : choix des ressources à stocker, choix des nœuds successeurs, …

# Napster




1: (enregistrement de i)
recherche de a1.mp3
(ajout des fichiers de i dans index)

2 : retour des noeuds possédant a1.mp3

3 : demande directe de i à j pour télécharger a1.mp3

4: téléchargement
de a1.mp3 et ajout dans la base de ressources partagées



# Bilan Napster

- Fonctionne bien à l'échelle d'Internet
- Accès en P2P mais recherche centralisée
- Serveur doit être bien dimensionné et tolérant aux fautes (cluster avec 100, 1000, ... noeuds)
- Sensible aux partitionnements du réseau (serveur inatteignable) et aux attaques

# Plan

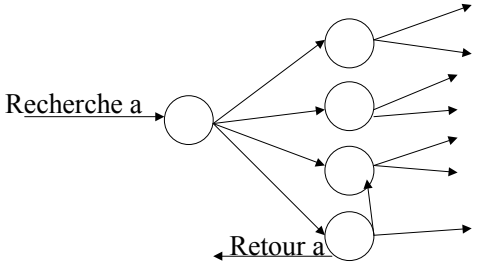
- P2P pur : GNUTELLA
- Super-peers
  - SuperPeers
  - Kazaa
- P2P « structuré »
  - Chord
  - P-Grid
  - Autres (Freenet, CAN, Pastry)
- P2P sémantique
  - SON
  - Routing Indices
  - Padoue
- Synthèse



# P2P « pur »

GNUTELLA

# Gnutella



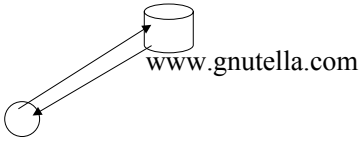

Chaque noeud propage la requête à k voisins (4)
Nombre de propagation limité (7)
Détection de cycles



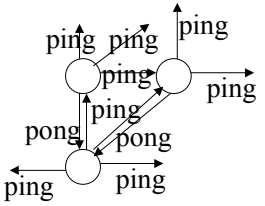
# Gnutella : types de messages

| **Type** | **Description** | **Information** |
|---|---|---|
| Ping | Annonce disponibilité et lance recherche nouveaux pairs | vide |
| Pong | Réponse à un ping | Adresse IP + No port; nombre et taille de fichiers partagés |
| Query | Requête | Bande passante minimum demandée; critère de recherche |
| QueryHit | Réponse à Query si on possède la ressource | Adresse IP + No port et bande passante; nombre de réponses + descripteurs réponses |
| Push | Demande de téléchargement pour pairs derrière un firewall | Identifiant du pair; index du fichier demandé; adresse IP et No port où envoyer le fichier |

# Gnutella (ajout d'un noeud)



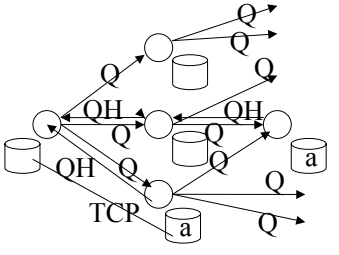

Initialisation de la
table des noeuds connus
(extérieur au protocole)




Compléter la table
des noeuds connus



# Gnutella (recherche)



# Les principes sous-jacents

- Principe d'égalité entre les nœuds
  - Même capacité (puissance, bande passante, ...)
  - Même comportement (également client et serveur) et bon comportement (pas de « mensonge »)
- Principe de requêtes « populaires »
  - Requêtes concernent principalement peu de ressources
  - Ressources très demandées sont très répliquées
- Principe de topologie du réseau
  - Graphe minimisant le nombre de chemins entre deux nœuds
  - Longueur du chemin minimum entre deux nœuds quelconque est faible (5 à 8)



# Quid des principes en réalité?

- Principe d'égalité entre les nœuds
  - A- [Saroiu et al. 01] montrent un écart de 1 à 3 dans la bande passante disponible
  - B- [Adar, Huberman 00] montrent que 70% des utilisateurs ne partagent aucun fichier (ils n'en ont pas ou bien ils n'intéressent personne) et que 50% des résultats sont produits par 1% des nœuds
  - A peut perturber le réseau et produire des partitionnements (trop forte charge demandée à des nœuds connectés via des modems e.g)
  - B implique que d'une part ceux qui partagent n'y ont pas intérêt (pas de réciprocité) et d'autre part que le réseau est sensible aux fautes et aux attaques
  - Études montrent également que certains nœuds sous-évaluent leur bande passante disponible pour éviter d'être choisis

# Quid des principes (2)

- Principe des requêtes « populaires »
  - Les 100 requêtes les plus fréquentes sont distribuées uniformément
  - Les autres suivent une distribution de type Zipf
  - Les techniques de cache de résultats s'appliquent bien et peuvent apporter une amélioration notable
- Principe de topologie du réseau
  - Plusieurs études montrent que le graphe sous-jacent de Gnutella est de type « small-world » et que le degré des nœuds suit une distribution « power law »
  - Le principe de diffusion de Gnutella ne s'adapte pas à SW (beaucoup de messages redondants)



# Observations sur la bande passante Gnutella

- Sur une période de un mois
  - Requête = 560 bits (y compris headers TCP/IP)
  - Requêtes 25% du trafic, pings 50% et reste 25%
  - En moyenne un pair est connecté activement à 3 autres
- Limite de la dégradation à 10 requêtes / seconde
  - 10 req. X 560 bits X 4 X 3 connections = 67200 b/s
  - Au-dessus de la capacité des modems
- Gnutella ne passe pas l'échelle

## Bilan de Gnutella

- Complètement décentralisé
- Très tolérant aux fautes
- S'adapte bien à la dynamique du réseau
- Simple, robuste et passe l'échelle (pour le moment)
- Gros consommateur de bande passante
- Pas de garantie de succès, ni d'estimation de la durée des requêtes
- Pas de sécurité, ni de réputation
- Problème du « free riding »



## Super-peers

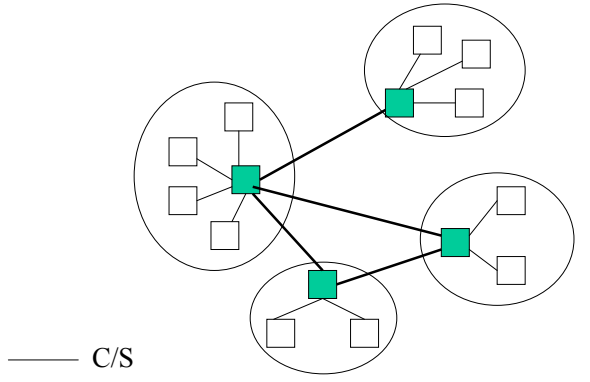
Client/serveur + P2P

# Super-peers

- Éviter les problèmes dus à l'hétérogénéité de la bande passante des nœuds
- Tous les nœuds ne sont plus égaux
  - Nœuds avec bonne bande passante sont organisés en P2P : les super-peers
  - Nœuds avec faible bande passante sont rattachés en mode client/serveur à un super-peer (cluster)
  - Super-peers disposent d'un index des ressources de leur cluster
- Utilisé dans KaZaa



# Exemple de super-peer

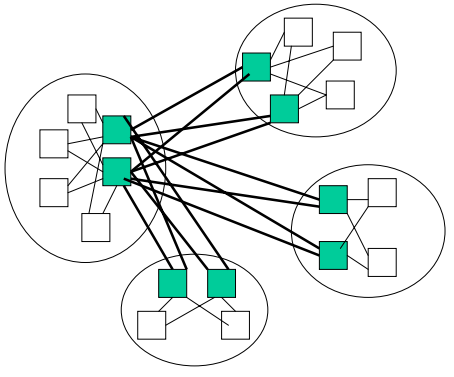


# Super-peer redondant

- Super-peer introduit de la sensibilité aux défaillances
- Amélioration possible, choisir k super-peers (partenaires) au lieu d'un dans un cluster
- Chaque partenaire est connecté à chaque client et possède un index de toutes leurs ressources
- Clients envoient leurs requêtes aux partenaires selon un principe de « round-robin »
- Les voisins d'un partenaire distribuent également leurs requêtes équitablement
- Fait baisser la charge du partenaire d'un facteur k
- Augmente le coût d'entrée d'un nouveau client d'un facteur k
- Augmente le nombre de connections ouvertes de $k^2$



# Exemple de super-peer redondant



# Règles de conception d'un super-peer [Yang 2003]

- R1 : augmenter la taille d'un cluster augmente la charge individuelle (bande passante) mais diminue la charge agrégée
  - Peu de gros clusters : sensibilité aux défaillances, attaques, ...
- R2 : la redondance de super-peer est favorable
  - Amène la bonne charge agrégée des gros clusters avec une faible charge individuelle et une bonne tolérance aux fautes



# Règles de conception d'un super-peer

- R3 : maximiser le nombre de connections des super-peers (diminue le nombre de sauts pour atteindre les résultats). Stratégie doit être choisie par tous les super-peers
- R4 : minimiser le TTL (Time To Live)

# KaZaa

- Système en vogue pour le partage de fichiers sur Internet
- Détails du fonctionnement sont non publics
- Utilise une technique de super-peer
- S'intéresse également au problème de la fiabilité des informations fournis par les nœuds (principe de vote)



# Principes de KaZaa

- Un nouvel utilisateur doit ouvrir un compte et déclarer ses répertoires de fichiers partagés et s'il accepte d'être super-peer;
- En fonction de sa bande passante (évaluée par KaZaa) le système le choisit comme super-peer ou non (la charge CPU induite est limitée à 10%);
- Un super-peer est connecté en P2P aux autres super-peers (selon un protocole à la Gnutella ?);
- Un nœud envoie ses requêtes à son super-peer (accessoirement il transmet aussi un index de ses fichiers partagés)
- Un super-peer dispose d'un index de ses ressources propres et de celles des nœuds de son cluster
- Un système de vote permet de noter les nœuds (qualité des ressources et ratio accès/servis) : utilisé pour ordonner les files d'attentes

# P2P « Structuré » (DHT)

Distributed Hash Table :
Chord, P-Grid



# Chord [Stoica et al. 2001]

- Table de hachage distribuée
- Les nœuds sont répartis sur un anneau
- Les ressources sont réparties sur les différents nœuds de l'anneau
- Structure dynamique
  - Ajout/retrait de nœud
  - Panne d'un nœud
- Peut être utilisée pour construire des applications au-dessus (DNS, …)

# Consistent Hashing

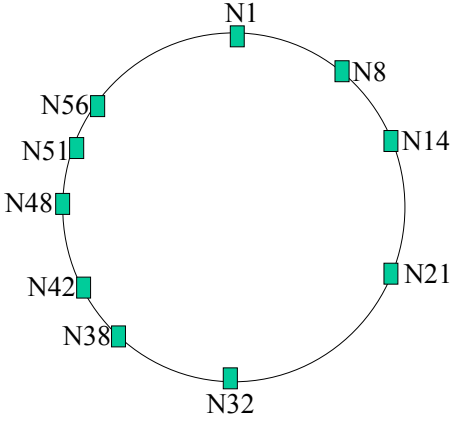
- Chord alloue les ressources aux nœuds en utilisant une fonction de hachage
- Consistent Hashing garantit avec une probabilité élevée
  - Ressources sont distribuées uniformément sur l'ensemble des nœuds
  - L'ajout/retrait d'un Nème nœud n'oblige à déplacer que O(1/N) ressources



# Structure en anneau

Chaque nœud est alloué sur l'anneau en fonction de hash(IP)

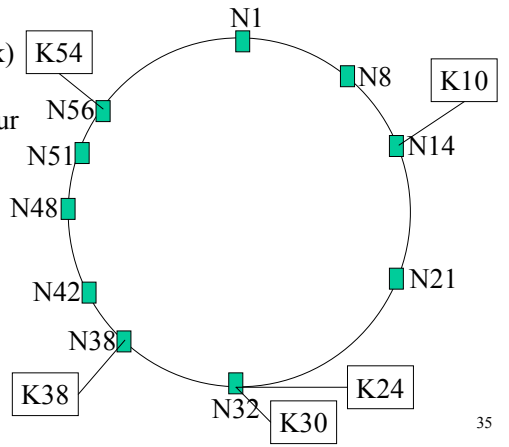
Au plus $2^m$ nœuds



# Placement des ressources

Hash(ressource)=k

k placé sur successeur(k)
successeur(k)=nœud immédiatement supérieur (ou égal) à k

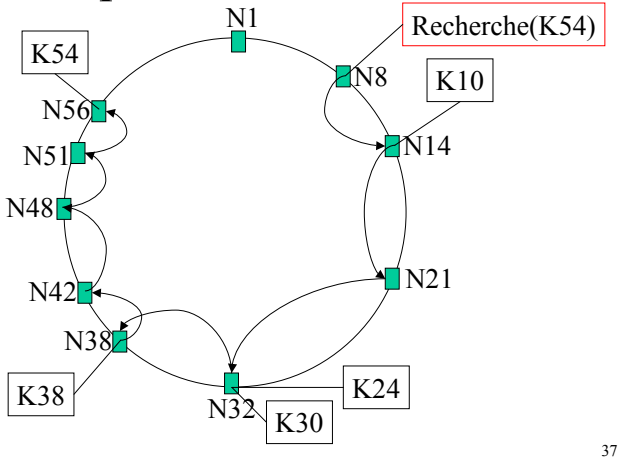





# Recherche (naive) d'une ressource

- Sur le nœud i on reçoit la requête : recherche(k)
- Si i possède k, il retourne k
- Sinon, il propage la requête à son successeur (chaque nœud doit stocker l'identification de son successeur)
- Le résultat suit le chemin dans l'ordre inverse
- Recherche linéaire en nombre de nœuds

# Exemple de recherche naive






# Amélioration de la recherche

- Avoir une table de routage plus complète
- Pour chaque nœud i :
  - Succ[k]=premier nœud sur l'anneau qui vérifie $(i + 2^{k-1}) \bmod 2^m$, $1 <= k <= m$
  - Successeur=succ[1]
  - m entrées dans la table
  - Prédecesseur (utilisé pour la maintenance dynamique du réseau)
  - Donne adresse IP et No de port du nœud

# Exemple de recherche

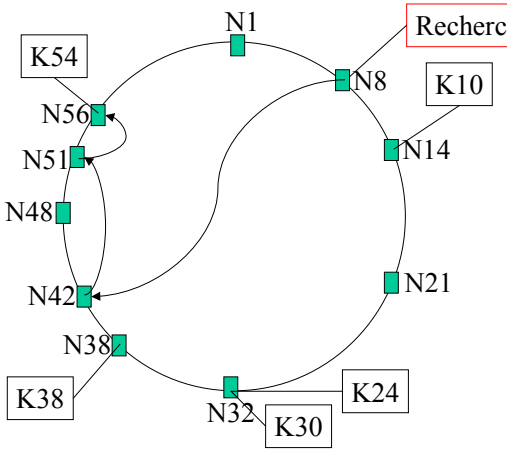



Table routage N8

| | |
|---|---|
| N8+1 | N14 |
| N8+2 | N14 |
| N8+4 | N14 |
| N8+8 | N21 |
| N8+16 | N32 |
| N8+32 | N42 |



# Algorithme de recherche

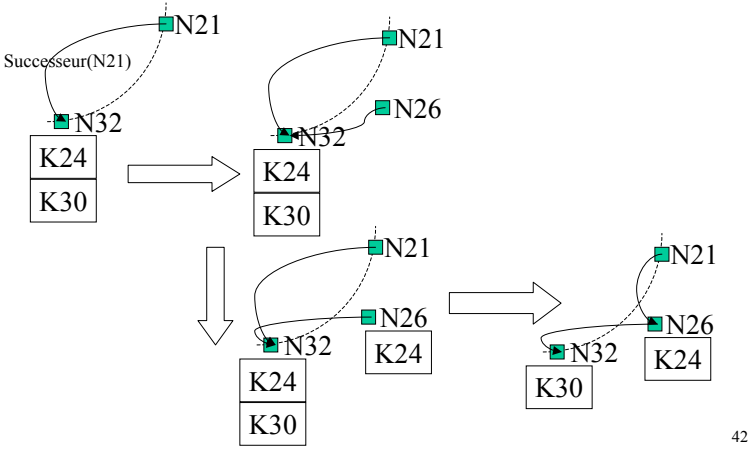
- Rechercher si la clé existe localement. Si oui on renvoie la valeur associée sinon
- Rechercher dans table routage nœud avec plus grande valeur inférieure ou égale à la clé cherchée
- Transmettre la requête au nœud sélectionné et appliquer récursivement
- Nombre de sauts moyen : $O(\log_2(N))$

# Ajout d'un nœud

- Repose sur la combinaison d'opérations élémentaires
  - N.join(n') : nœud n annonce au nœud n' qu'il rentre dans le réseau et lui demande de lui fournir son successeur
  - N.stabilize() : lancé périodiquement, permet à n et à son successeur de vérifier qu'ils forment un couple correct (il n'y a pas de nouveaux nœuds entre les 2)
  - N.majentrées() : lancé périodiquement, permet d'initialiser la table de routage pour les nouveaux nœuds ou de la mettre à jour pour les nœuds existants
  - N.testpredecesseur() : lancé périodiquement, vérifie que le prédecesseur est toujours là



# Exemple d'ajout



## Propriétés de l'algorithme d'ajout

- L'algorithme garantit que n'importe quelle séquence de join entrelacée avec des stabilize converge vers un état stable (tous les nœuds restent atteignables)
- Même en présence d'un ajout (limité) de nœud, le coût de la recherche reste en O(log(N))
- Une recherche peut échouer si le réseau n'est pas complètement stabilisé (il faut relancer la recherche un peu plus tard)



## Sensibilité aux pannes

- Algorithme de recherche repose sur la notion de successeur (sensible à la défaillance de celui-ci)
- Une solution consiste à gérer une liste de r successeurs

# Evaluation Chord

- Type de recherche : égalité
- Coût de la recherche : O(Log(n))
- Coût de la mise à jour : O(Log(n))
- Coût de l'ajout d'un nœud : $O(Log_2(n))$
- Pas d'autonomie de stockage et de routage



# Bilan Chord

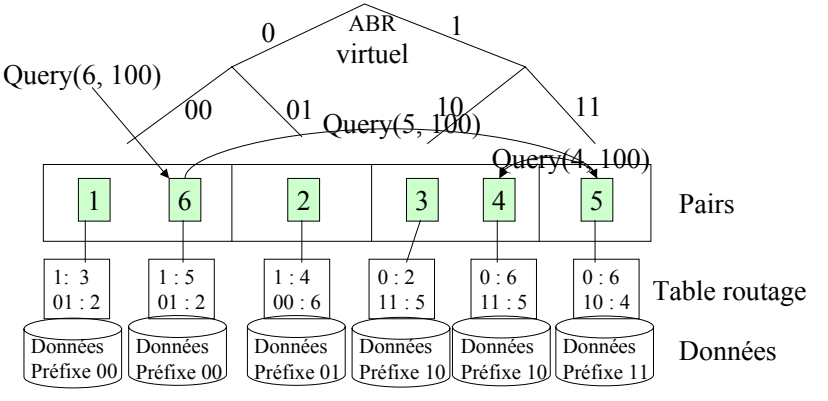
- Algorithme assez simple, avec de bonnes propriétés démontrables
- Résultats expérimentaux confirment
- Problème de latence :
  - Fonction de recherche minimise le nombre de sauts, mais tous les sauts n'ont pas forcément le même prix (traversée transatlantique e.g)
  - Besoin d'utiliser de l'information sur la distance entre les nœuds (on choisira parmi les successeurs possibles celui à distance minimale) -> Global Network Positionning

# P-Grid [Aberer 01]

- Table de hachage distribuée
- Nœuds se répartissent les ressources selon une structure d'arbre binaire de recherche
- Repose sur une fonction de hachage préservant les préfixes et distribuant uniformément les ressources sur l'arbre binaire
- Structure auto-adaptative, avec réplication
- Permet les recherches basées sur les préfixes (pas seulement égalité stricte)



# Exemple d'un P-Grid



## Recherche dans un P-Grid

- Nœuds ne sont pas placés selon une fonction de hachage (seulement les ressources), mais sont responsables de ressources ayant le même préfixe
- Requête r soumise sur nœud i
  - Nœud satisfait le préfixe de la ressource : OK
  - Sinon, recherche dans table routage, nœud satisfaisant le « plus proche » préfixe : envoi récursif de la requête sur ce nœud



## Réplication, tolérance aux fautes

- Structure naturellement répliquée (plusieurs nœuds responsables du même préfixe)
- Tolérance aux fautes :
  - on maintient dans les tables de routage non pas un pair par préfixe mais plusieurs
  - On en choisit un dans la liste et si cela ne marche pas on itère sur les suivants

## Construction du P-Grid

- Initialement tous les nœuds sont responsables de tout l'espace de recherche
- Lorsque deux nœuds se rencontrent (lors d'une recherche ou bien sur un « ping »), ils vont se répartir l'espace de recherche et se référencer mutuellement (différents cas, selon qu'ils aient un préfixe commun ou pas)
- Simulations montrent que l'algorithme converge quasi-indépendamment du nombre de nœuds



## Fonction de hachage

- Doit respecter la relation de préfixe sur les noms de fichiers :
  - Si f1 préfixe f2 alors clé(f1) préfixe clé(f2)
- Doit assurer une bonne distribution dans l'ABR (pour qu'il puisse être équilibré et avoir un même nombre de ressources par nœud)
- Échantillonnage des noms de fichiers dans Gnutella : construction d'un arbre lexicographique sur les noms qui est ensuite projeté sur l'ABR

## Evaluation de P-Grid

- Type de recherche : préfixe et intervalle
- Coût de recherche et de mise à jour : O(Log(n))
- Construction : difficile à évaluer, simulation semble montrer que c'est efficace
- Pas d'autonomie de routage
- Faible autonomie de stockage (choix d'un nœud qui gère le bon préfixe)



## Bilan de P-Grid

- Structure assez simple, tolérante aux fautes, avec passage à l'échelle
- Évite de structurer les nœuds selon une fonction de hachage
- Peut intégrer facilement des optimisations sur la latence (choix d'un pair)
- N'est pas restreint aux recherches basées sur l'égalité stricte

# Autres approches de DHT

- Freenet [Clarke 01]
- CAN [Ratsanamy 01] : espace à n-dimensions
- Pastry [Rowstron 01] : hypercube
- JXTA [Gong 01] : environnement de programmation multi-plateformes, en Java pour interopérabilité des systèmes P2P



# Freenet [Clarke 01]

- Système pour permettre la publication, la réplication et la recherche de données (« lutter contre la censure »)
- Assure l'anonymat des producteurs et des consommateurs
  - Impossible de déterminer l'origine ou la destination des données (chaîne de proxies)
  - Difficile pour un nœud de savoir quelles données il stocke (tout est crypté)
- Requêtes sont routées sur le nœud le plus proche
- Réplication dynamique des données (lors d'une recherche et d'une insertion)

## Bilan Freenet

- Type de recherche : égalité
- Coût de recherche et insertion : O(Log(n))
- Réplication intrinsèque
- Autonomie de stockage
- Autonomie partielle du routage (dépendance entre clés stockées et requêtes traitées)



## P2P « sémantique »

Routing Indices, SON, Padoue

# P2P « sémantique »

- Ajouter de l'information aux tables de routage
- Généraliste vs spécifique (à une application)
- Information doit pouvoir être maintenue dynamiquement
- Équilibre entre taille et précision



# Quelle information ?

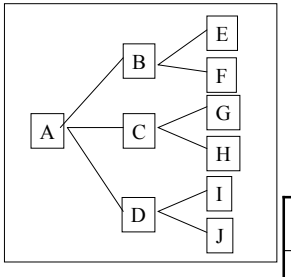
- Sur le contenu des nœuds (index)
  - Suppose que les nœuds aient un contenu homogène (pas de placement aléatoire)
  - Routage par comparaison requête-index
  - Même information partagée par tous, ou bien information construite localement
  - Exemple « routing indices », SON
- Sur les requêtes
  - Suppose qu'il y ait peu de requêtes différentes
  - Routage par comparaison requête-requête
- Sur les utilisateurs
  - Suppose que les utilisateurs aient toujours le même besoin
  - Routage par comparaison utilisateur-utilisateur
- Sur un mixte des trois solutions
  - Exemple de Padoue

# « Routing Indices » [Crespo 02]

- Introduire de l'information sur le contenu des nœuds (index)
- Analogue aux systèmes d'index répartis et hiérarchisés pour moteurs de recherche sur Internet
- Trouver l'équilibre entre la taille de l'index et le gain



# Exemple de « RI »




RI pour nœud A

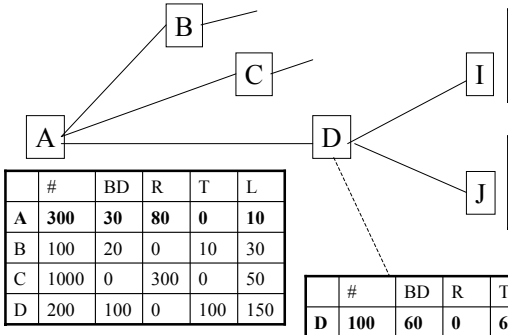
| Chemin | Nb documents | BD | Réseaux | Théorie | langages |
|---|---|---|---|---|---|
| B | 100 | 20 | 0 | 10 | 30 |
| C | 1000 | 0 | 300 | 0 | 50 |
| D | 200 | 100 | 0 | 100 | 150 |

# Utilisation de l'index

- Soit Q une requête, conjonction de plusieurs termes de recherche ($t_Q^1$, …$t_Q^k$)
- Proximité(Q, chemin) = Nbdedocuments X $\prod_i$ (RI($t_Q^i$)/Nbdedocuments)
- Q émise sur A = ('BD', 'langages')
- Proximité(Q, B) = 100 X 20/100 X 30/100 = 6
- Proximité(Q, C) = 1000 X 0/1000 X 50/1000 = 0
- Proximité(Q, D) = 200 X 100/200 X 150/200 = 75
- Permet d'ordonner les nœuds successeurs



# Exemple complet de RI




| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **A** | **300** | **30** | **80** | **0** | **10** |
| B | 100 | 20 | 0 | 10 | 30 |
| C | 1000 | 0 | 300 | 0 | 50 |
| D | 200 | 100 | 0 | 100 | 150 |

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **I** | **50** | **25** | **0** | **15** | **50** |
| D | 1550 | 125 | 380 | 95 | 190 |

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **J** | **50** | **15** | **0** | **25** | **25** |
| D | 1550 | 135 | 380 | 85 | 215 |

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **D** | **100** | **60** | **0** | **60** | **75** |
| A | 1400 | 50 | 380 | 10 | 90 |
| I | 50 | 25 | 0 | 15 | 50 |
| J | 50 | 15 | 0 | 25 | 25 |

## Algorithme de recherche

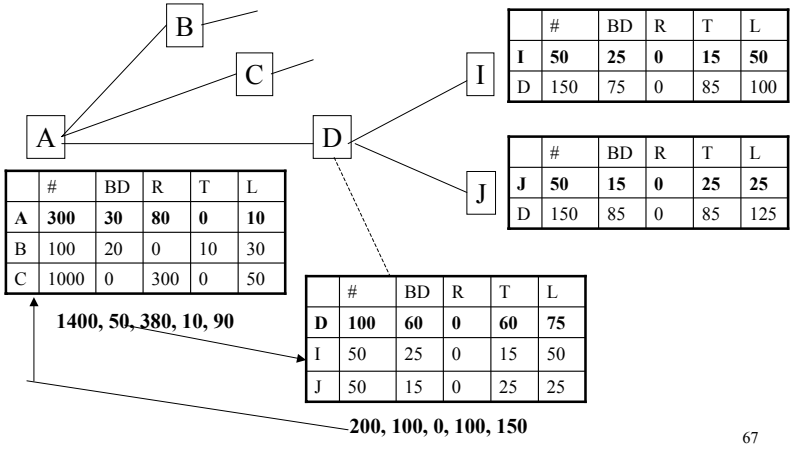
- Résoudre Q localement. Si suffisamment de résultats OK, sinon
- Évaluer proximité des successeurs
- Tant qu'il n'y a pas assez de résultats
  - Prendre le successeur S le plus proche
  - Recherche(Q, S)



## Performances de la recherche

- Par rapport à Gnutella diminue le nombre de messages
- Exploration restreinte aux nœuds ayant la plus grande probabilité de succès
- Pas d'information sur le nombre de sauts nécessaires (améliorations possibles avec d'autres RI)
- Pas de garantie d'avoir tous les résultats
- Plutôt orienté recherche des k meilleurs résultats

# Création d'un RI




| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **A** | **300** | **30** | **80** | **0** | **10** |
| B | 100 | 20 | 0 | 10 | 30 |
| C | 1000 | 0 | 300 | 0 | 50 |

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **I** | **50** | **25** | **0** | **15** | **50** |
| D | 150 | 75 | 0 | 85 | 100 |

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **J** | **50** | **15** | **0** | **25** | **25** |
| D | 150 | 85 | 0 | 85 | 125 |

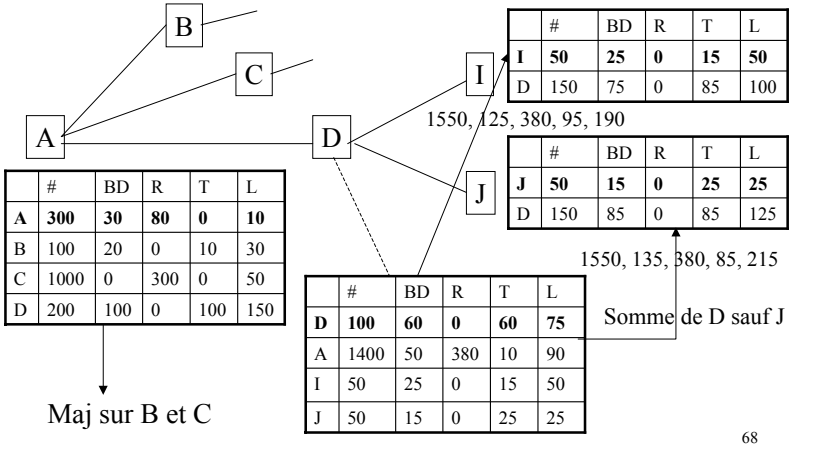
| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **D** | **100** | **60** | **0** | **60** | **75** |
| I | 50 | 25 | 0 | 15 | 50 |
| J | 50 | 15 | 0 | 25 | 25 |

**1400, 50, 380, 10, 90**

**200, 100, 0, 100, 150**



# Création d'un RI (2)




| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **A** | **300** | **30** | **80** | **0** | **10** |
| B | 100 | 20 | 0 | 10 | 30 |
| C | 1000 | 0 | 300 | 0 | 50 |
| D | 200 | 100 | 0 | 100 | 150 |

Maj sur B et C

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **I** | **50** | **25** | **0** | **15** | **50** |
| D | 150 | 75 | 0 | 85 | 100 |

1550, 125, 380, 95, 190

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **J** | **50** | **15** | **0** | **25** | **25** |
| D | 150 | 85 | 0 | 85 | 125 |

1550, 135, 380, 85, 215

| | # | BD | R | T | L |
|---|---|---|---|---|---|
| **D** | **100** | **60** | **0** | **60** | **75** |
| A | 1400 | 50 | 380 | 10 | 90 |
| I | 50 | 25 | 0 | 15 | 50 |
| J | 50 | 15 | 0 | 25 | 25 |

Somme de D sauf J

# Maj d'un RI

- Analogue au processus de création
- Pour diminuer le coût, on peut accumuler les maj et les traiter par lots
- Suppression d'un nœud suit le même principe



# Bilan RI

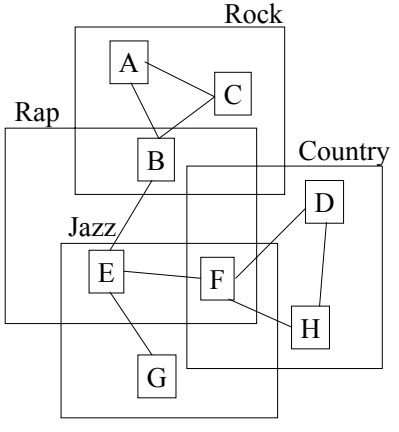
- Structure d'indexation assez simple, mais maj génère beaucoup de messages
- Fonctionne bien pour obtenir les meilleurs résultats, mais pas forcément tous les résultats
- Pour traiter des graphes généraux, il faut intégrer la gestion des cycles (détection ou prévention)
- S'applique à des langages types mots-clés mais pas généralisable à des langages plus complexes
- RI ne donne pas d'indications sur le nombre de sauts (d'autres RI sont proposées)

# Semantic Overlay Networks

- [Crespo, Garcia-Molina 03]
- Classifier l'ensemble des nœuds via une classification « sémantique » (e.g genres de musique)
- Un même nœud peut se trouver dans plusieurs classes
- Selon la requête on sélectionne le ou les SON susceptibles d'y répondre au mieux



# Exemple de SON



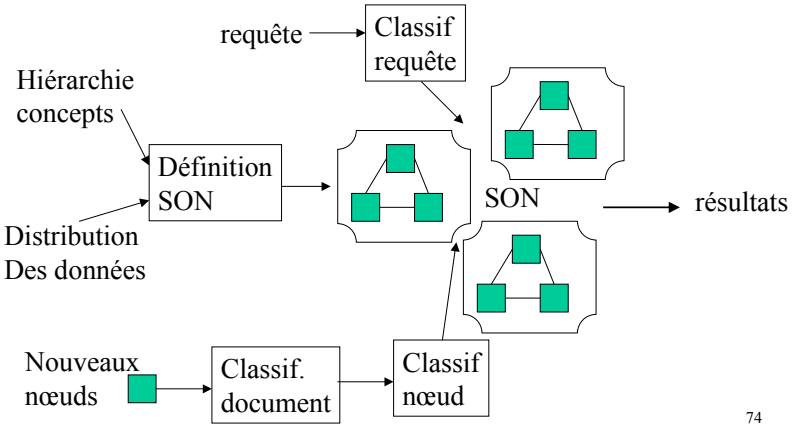

$n_i$
$\{(n_i, n_j, l_k)\}$
(A, B, 'Rock')

# Problématique des SON

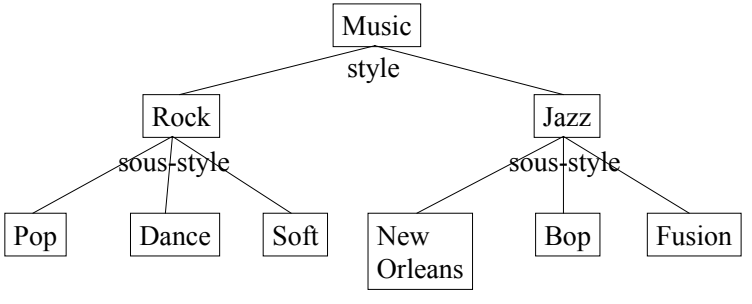
- SON avec un seul label est un P2P classique
- Une fois choisi le label en fonction de la requête -> P2P classique
- Comment définir le SON



# Processus de génération d'un SON



# Exemple de classification



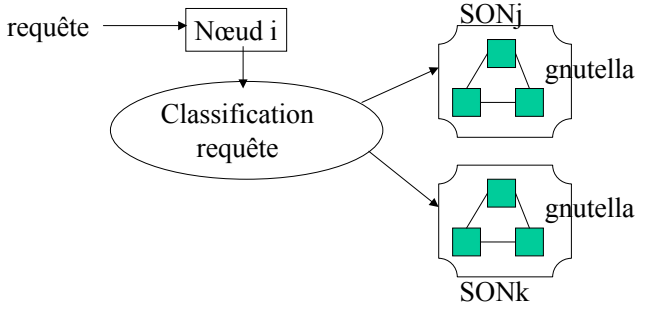

un concept = un SON



# Association des nœuds aux SON

- Repose sur un classifieur de documents
  - Si un document du nœud correspond : favorise le rappel mais augmente le coût de la recherche
  - Si k documents du nœud correspondent : diminue le rappel mais diminue le coût de la recherche
- Résolution d'une requête
  - Cherche le(s) concept(s) correspondant à la requête
  - Propage la requête dans le SON + ascendants + descendants

# Principe de recherche






# Choix de la classification

- « bonne » hiérarchie de classification
  - Classes dont les documents appartiennent à un petit nombre de nœuds
  - Nœuds se trouvent dans peu de classes
  - Classifieurs faciles à construire et les plus fiables possible

# Bilan SON

- Repose sur la classification (amélioration possible avec « layered SONs »)
- Lié à un domaine précis
- Favorise la précision mais pas le rappel
- Peut se paralléliser pour obtenir rapidement des réponses (requête lancée dans chaque SON sélectionné par le classifieur)
- Résultats expérimentaux montrent une amélioration notable en nombre de messages / gnutella



# PADOUE

- Projet ACI GRID MR avec INRIA Rocquencourt, LIP6, LIRMM, Cemagref
- Partage de données et de programmes entre chercheurs dans le domaine de l'environnement
- Réseau à grande échelle, spécialisé, avec des utilisateurs spécialistes : P2P sémantique

# Principes P2P Padoue

- Mélanger différents niveaux d'informations
  - Réseau global : analogue à SON
  - Utilisateurs : nœuds dont ils sont satisfaits
  - Requêtes : utiliser les mêmes nœuds que ceux ayant traité avec succès des requêtes analogues
- Utiliser du relevance feedback (RF)
- Propager le RF vers tous les nœuds par lesquels la requête a transité

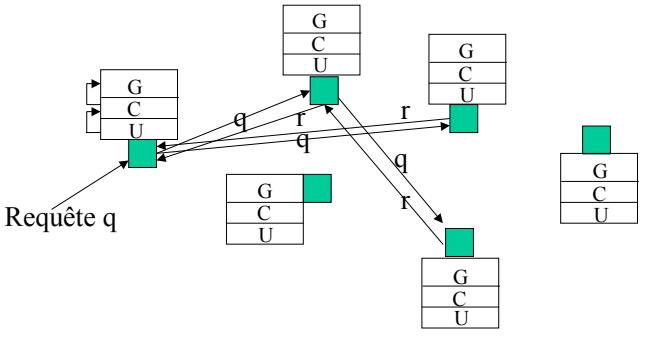


# Informations sémantiques de Padoue

| | | |
|---|---|---|
| global | Type SON | |
| communauté | $\{(c_j, \{(\text{critère}_i, \{n_j\})\})\}$ $c_{autres}$ | Agrégé pour tous Utilisateurs de la communauté |
| utilisateur | $\{(u_k, \{(\text{critère}_i, \{n_j\})\})\}$ | |

Taille fixe gérée en LRU

# Recherche dans Padoue

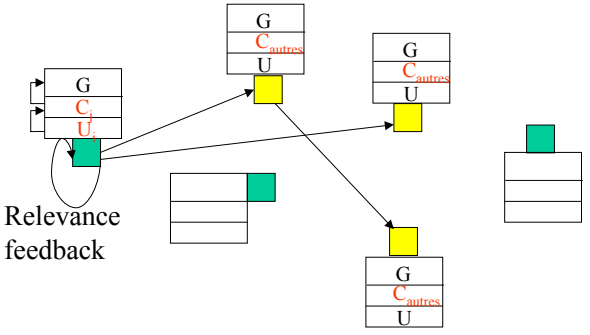





# Construction du réseau

- Ajout d'un nœud : il doit s'adresser à un nœud connu du réseau et il récupère l'information globale
- Maj des informations de routage : relevance feedback de l'utilisateur va permettre de maj l'information utilisateur + communauté du nœud de rattachement de l'utilisateur

# Propagation de RF






# Bilan Padoue

- Utilisation conjointe de différents types d'informations
- Informations statiques et dynamiques
- Besoin d'un minimum d'utilisation pour avoir de bonnes tables de routage
- Suppose que les sites soient spécialisés et que les utilisateurs posent des requêtes semblables
- Adapté à un domaine spécialisé

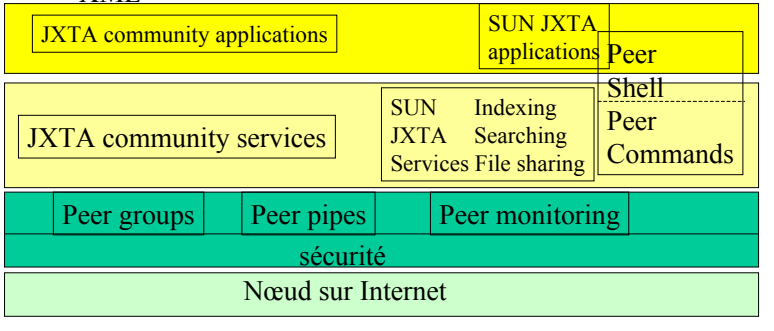
# Synthèse

Choix P2P, problèmes ouverts



# Projet JXTA (Sun)

- Plate-forme « neutre » de programmation réseau pour systèmes P2P
- 3 niveaux et 6 protocoles (découverte, routage, …) basés sur XML



# Besoins applicatifs

| | Type recherche | résultats | Garantie | Autonomie stockage | Autonomie routage |
|---|---|---|---|---|---|
| Gnutella | filtre | Tous? | Non | Oui | Oui |
| Chord | égalité | Un | oui | non | non |
| P-Grid | préfixe | Un | oui | Faible | non |
| Freenet | égalité | Un | Non | Oui | O/N |
| Super-peer | filtre | K meilleurs | non | Oui | Oui pour SP, non pour autres |
| sémantique | filtre | K meilleurs | non | Oui | oui |



# Comparatif performances

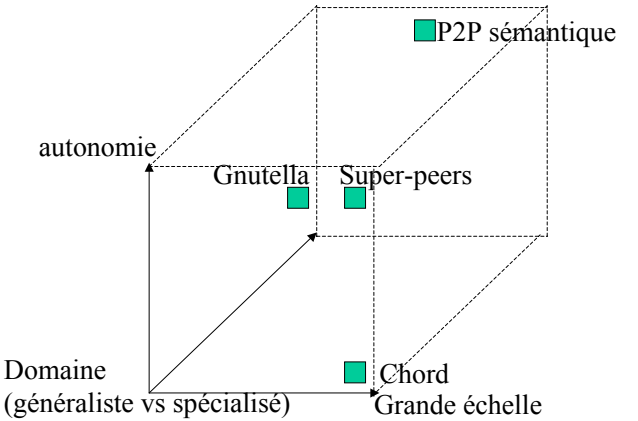
| | Modèle recherche | Coût recherche (messages) | ajout noeud | MAJ table routage |
|---|---|---|---|---|
| Gnutella | largeur | $2*\Sigma_{i=0}^{TTL} C*(C-1)^i$ | Init. routage | Périodique (ping) |
| Chord | Arbre binaire recherche | O(Log(n)) | maj routage, échange ressources | Périodique (stabilize) |
| P-Grid | Arbre binaire préfixe | O(Log(n)) | maj routage, échange ressources | Rencontres |
| Freenet | profondeur | O(Log(n)) ? | Init. Routage | Pendant recherche et insertion |

# Comparatif sémantique

| | Type de Connaissances | statique | dynamique | apprentissage |
|---|---|---|---|---|
| SON | Classification domaine | oui | Non | classifieur |
| Routing Indices | Indexation contenu | non | oui | Comparaison requête / Indice |
| Padoue | Classification domaine + requêtes passées + utilisateurs | oui | oui | - Requête<br>- Relevance feedback<br>- utilisateur |



# Taxonomie



# Problèmes ouverts

- Sécurité
- Réputation
- Contrôler le « free riding »
- P2P sémantique
  - On en est au début!
  - Mixer différents types de connaissances
  - Connaissances dynamiques (apprentissage)



# Sécurité

- Disponibilité (résistance aux attaques DoS)
- Authentification des ressources (si plusieurs réponses différentes pour la même ressource)
  - Plus vieille ressource
  - Réputation (« experts »)
  - Vote
- Anonymat
  - Auteur : quels utilisateurs ont créé quels documents,
  - Serveur : sur quels noeuds est stocké un document donné?
  - Lecteur : quels utilisateurs accèdent quels documents?
  - Document : quels documents sont stockés sur un noeud donné?

# Bibliographie


- K. Aberer et al. « Improving Data Access in P2P Systems », IEEE Internet Computing, January 2002
- K. Aberer « P-Grid: A Self-Organizing Access Structure for P2P Information Systems », Proc. COOPIS, 2001
- K. Aberer, M. Hauswirth, « Peer-to-Peer Information Systems: Concepts and Models, state-of-the-art, and Future Systems », Tutorial IEEE ICDE, 2002
- E. Adar, B. Huberman « Free Riding on Gnutella », Technical Report, Xerox PARC, septembre 2000
- I. Clarke et al. « Freenet: A Distributed Anonymous Information Storage and Retrieval System », LNCS 2009, Springer Verlag, 2001
- A. Crespo, H. Garcia-Molina « Routing Indices for Peer-to-Peer Systems », Proc. ICDCS 2002
- A. Crespo, H. Garcia-Molina « Semantic Overlay Networks for P2P Systems », soumis à publication




# Bibliographie (2)


- N. Daswani, H. Garcia-Molina, B. Yang « Open Problems in Data-Sharing Peer-to-Peer Systems », Proc. ICDT Conf, 2003
- L. Gong « JXTA: A Network Programming Environment », IEEE Internet Computing, 5(3), mai-juin 2001
- Gnutella : http://www.gnutella.com
- M.A. Jovanovic et al « Scalability Issues in Large Peer-to-Peer Networks – A Case Study of Gnutella », Research report, Univ. Cincinnati, 2001
- J. Kleinberg « The Small-World Phenomenon: An Algorithmic Perspective, Technical Report 99-1776, Cornell Univ., 1999
- J. Kubiatowicz et al « OceanStore: An Architecture for Global-Scale Persistent Storage », Proc. ASPLOS, 2000
- S. Ratsanamy et al. « A Scalable Content Addressable Network », Proc. ACM SIGCOMM 2001

# Bibliographie (3)


- A. Rowstron, P. Dreschel « Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems », Proc. IFIP/ACM Conf. On Distributed Systems Platforms, 2001
- S. Saroiu, P. Gummadi, S. Gribble « A Measurement Study of Peer-to-Peer Sharing Systems », Proc. Multimedia Computing and Networking, 2002
- K. Sripanidkulchai « The Popularity of Gnutella Queries and its Implications on Scalability », février 2001
- I. Stoica et al. « Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications », Proc. ACM SIGCOMM 2001
- B. Yang, H. Garcia-Molina « Improving Search in Peer-to-Peer Systems », Proc. 28th Conf. On Distributed Computing Systems, 2002
- B. Yang, H. Garcia-Molina « Comparing Hybrid Peer-to-Peer Systems », Proc. VLDB Conference, 2002
- B. Yang, H. Garcia-Molina « Designing a Super-Peer Network », Proc. ICDE Conf., 2003