Notes de cours (ENS Lyon, M1)

Chapitre 2 : Réseaux Pair à Pair

Table des matières

2 Réseaux pair à pair 25

2.1 Introduction et définitions 25
2.1.1 Définition 25
2.1.2 Historique 25
2.1.3 Problématique et objectifs 26
2.1.4 Classification 26
2.2 Système partiellement décentralisé : modèle hybride 28
2.2.1 Napster 28
2.2.2 Kazaa 29
2.2.3 Bilan : avantages et limites 29
2.3 Totalement décentralisé : modèle pur 30
2.3.1 Connexion au réseau 30
2.3.2 Recherche de ressource 30
2.3.3 Avantages et limites 31
2.4 Systèmes structurés 32
2.4.1 Le principe 32
2.4.2 Recherche d'une ressource dans Chord 33
2.4.3 Répartition des clés : notion de hachage consistant 35
2.4.4 Ajout de noeuds et stabilisation 35
2.4.5 Départ de noeuds et tolérance aux pannes 36
2.4.6 Conclusion 37
2.5 Vers un partage équitable 38
2.5.1 Téléchargement multiple 38
2.5.2 BitTorrent et le partage équitable 38
2.6 Mécanisme de recherche persistante 40
2.6.1 Publication/abonnement 40
2.6.2 Meghdoot 41
2.6.3 Sub-2-Sub 43

2 Réseaux pair à pair

2.1 Introduction et définitions

2.1.1 Définition

Les systèmes pair à pair (P2P, de l'anglais "peer-to-peer") sont composés d'un ensemble d'entités partageant un ensemble de ressources, et jouant à la fois le rôle de serveur et de client. Chaque noeud peut ainsi télécharger des ressources à partir d'un autre noeud, tout en fournissant des ressources à un troisième noeud.

Immense essor de tels systèmes, phénomène de société avec d'importants impacts en termes commerciaux (droits, taxes) et moraux (contenu des données échangées).

Permet une utilisation maximale de la puissance du réseau, une élimination des coûts d'infrastructure, et une exploitation du fort potentiel inactif en bordure de l'Internet. Retient l'attention de la recherche, des développeurs et des investisseurs.

Les pairs du réseau peuvent être de nature hétérogène : PC, PDA, Téléphone portable...

Nature dynamique : les pairs peuvent aller et venir.

2.1.2 Historique

Internet = P2P : libre échange des données. Emergence du P2P = renaissance du modèle originel d'Internet, au niveau applicatif.

Le P2P est devenu très populaire avec *Napster*, en 1999. Logiciel pour le partage de musique en ligne. Logiciel avec le plus grand taux de croissance de tous temps. Accès internet pour les américains : téléchargement de musique 24h/24 7j/7. Problème de copyright pour les compagnies de disque : poursuite judiciaire en 2000. Accord en 2001 : Napster devait payer 26 millions de dollars aux compositeurs et éditeurs, ainsi qu'un pourcentage de l'argent obtenu par un service payant (depuis 2003).

Après les problèmes rencontrés par Napster, émergence de nombreux autres programmes P2P, parmi lesquels Gnutella et Kazaa.

Meilleure compréhension des technologies P2P, et applications utilisant une approche plus décentralisée, pour rendre un contrôle de police plus difficile. Difficulté d'élaborer des protocoles efficaces du fait du manque d'élément central.

Problèmes de copyright : pas de moyen de contrôle, mais les nouvelles applications P2P ont retenu la leçon Napster et argumentent qu'ils ne sont pas responsables des activités illégales utilisant leur logiciel. Il y a de nombreuses utilisations légales qui assurent le futur du P2P, mais il est dur d'interdire l'échange de fichiers illégaux (musique, films...).

Moyens d'attaque : pairs des compagnies de disque qui attaquent le réseau en créant des données bidon, ou en surchargeant le réseau. Autre technique : repérage des gros utilisateurs.

Le P2P est devenu un peu plus que du partage de fichiers, avec également des projets pour utiliser la puissance de calcul des pairs plutôt que leur collection de MP3 (SETI@home – Search for Extra-Terrestrial Intelligence, Groove). Manière forte et équitable de collaborer en vue d'accroître le potentiel du réseau.

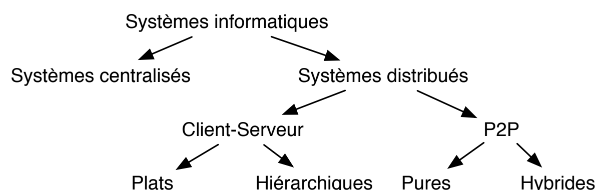
2.1.3 Problématique et objectifs

Problématique du P2P : bien définir la relation entre un éventuel *réseau virtuel – overlay network* qui connecte les membres présents dans le système et les mécanismes de routage, afin d'assurer certaines propriétés :
- Recherche rapide d'une donnée
- Mise à jour rapide du réseau lors du départ et de l'arrivée d'un membre dans le réseau
- Publication rapide d'une donnée
- Sécurité et anonymat des transactions, pour empêcher le pistage d'une requête

Les applications sont de nature différente, et donc les objectifs variés :
- Partage et réduction des coûts entre les différents pairs
- Fiabilité (pas d'élément centralisé)
- Passage à l'échelle (éviter les goulots d'étranglement)
- Agrégation des ressources (puissance de calcul, espace de stockage)
- Accroissement de l'autonomie du système, chacun a la responsabilité de partager ses ressources
- Anonymat

2.1.4 Classification

P2P : branche des systèmes distribués dans la classification des systèmes informatiques :



Trois distinctions dans les réseaux pair à pair, suivant les choix de conception : présence ou non d'une connexion virtuelle entre les membres présents dans le système (*réseau virtuel – overlay network*), et mécanisme de recherche au sein de ce réseau.

1. Modèle hybride : Systèmes partiellement décentralisés (Napster et Kazaa) : pas de réseau virtuel mais utilisation d'index plus ou moins répartis pour fournir directement l'adresse IP d'un membre disposant de la donnée recherchée.

2. Modèle pur : Systèmes totalement décentralisés (Gnutella) : réseau virtuel non structuré, sur lequel la recherche s'effectue par inondation.
3. Systèmes reposant sur un réseau virtuel structuré correspondant à une table de hachage distribuée (DHT pour *distributed hash table*), sur lequel une stratégie de routage spécifique à la topologie DHT est appliquée (Chord, Pastry). Algorithmes intéressants assurant une complexité logarithmique en fonction du nombre de pairs présents dans le réseau.

Caractéristiques des réseaux P2P :
- Localisation des fichiers dans un environnement distribué
- Méta-données ou index du réseau P2P
- Libre circulation des fichiers entre les pairs
- Capacité de connexion variable suivant les pairs
- Hétérogénéité du réseau suivant les pairs
- Echanges d'information non sécurisés
- Certains pairs peuvent être non fiables
- Aucune vue globale du système

Traditionnellement, échange de ressources et de services entre ordinateurs : techniques client/serveur. Un ordinateur dominant, le serveur, connecté à plusieurs clients qui ont moins de contrôle. Les clients peuvent communiquer entre eux uniquement à travers le serveur.

P2P : chaque pair peut se comporter à la fois comme client et comme serveur : contrôle décentralisé.

Comparaison avec le modèle client/serveur :

P2P	Client/Serveur
Auto-organisé	Management centralisé
Evolution dynamique	Configuration statique
Découverte des pairs	Consultation de tables
Flux distribué	Flux centralisé
Symétrie du réseau	Asymétrie du réseau
Adressage dynamique (appli.)	Adressage statique (IP)
Noeuds autonomes	Noeuds dépendants
Attaques difficiles	Attaques plus simples

Nous abordons dans ce chapitre les différents types de réseaux P2P et les algorithmes mis en oeuvre dans chaque cas.

2.2 Système partiellement décentralisé : modèle hybride

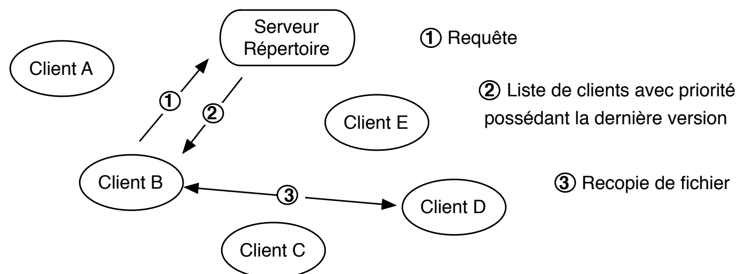
Certains “serveurs” sont contactés pour obtenir des méta-informations (identité du pair sur lequel sont stockés les informations), mais les données restent distribuées sur les pairs et les échanges de données se font directement d’un pair à un autre.

2.2.1 Napster

Fondé en 1999 par Shawn Fanning et Sean Parker.

P2P hybride :
- serveur central qui conserve l’information sur les pairs et répond aux requêtes pour cette information.
- pairs responsables pour héberger l’information car le serveur central ne stocke pas de fichiers
- les pairs doivent dire au serveur quels fichiers ils veulent partager et quels fichiers ils demandent.
- les adresses IP des autres pairs sont obtenues

Plusieurs répertoires centralisés décrivant les fichiers présents dans Napster, et chaque pair s’enregistre sur ce répertoire lorsqu’il rejoint le réseau. Les répertoires connaissent les adresses IP, et le nom des fichiers qu’un pair veut partager.



Principe :
1. Chaque utilisateur dispose du logiciel Napster. A l’exécution du logiciel, Napster recherche une connexion internet.
2. Si une connexion est détectée, une connexion entre l’utilisateur et un des serveurs centraux de Napster est établie.
3. Le serveur central maintient un répertoire des ordinateurs clients connectés et stocke les informations sur ces utilisateurs (notamment les fichiers en partage).
4. Si un client désire un fichier, il passe une requête au serveur centralisé auquel il est connecté.
5. Le serveur regarde s’il peut répondre à la requête.
6. Le serveur renvoie à l’utilisateur une liste des réponses éventuelles : adresse IP, nom d’utilisateur, taille du fichier, ...

7. L'utilisateur choisit le fichier qu'il veut et établit une connexion directe avec l'hôte du fichier, en lui envoyant sa propre adresse IP et le nom du fichier demandé.
8. Transfert du fichier entre les deux ordinateurs, puis connexion interrompue à la fin du transfert.

2.2.2 Kazaa

Même principe, mais absence de serveurs fixes. Utilisation des pairs avec une connexion Internet rapide et processeur puissant : SuperPairs. Ces noeuds particuliers du réseau se comportent comme les serveurs Napster : connaissance des fichiers présents chez les autres utilisateurs, et adresse de ces fichiers. Chaque SuperPair réfère une recherche aux autres SuperPairs.

La recherche est donc plus rapide, on ne cherche que dans les fichiers indexés par le SuperPair auquel on est connecté.

Ce modèle SuperPair est une variante du modèle hybride.

2.2.3 Bilan : avantages et limites

Avantages d'un système hybride :
- Présence d'un serveur central : facile à administrer, et donc facile à contrôler
- Evite les recherches coûteuses sur le réseau : pas de routage et planification de la gestion des utilisateurs
- Tolérance aux fautes en sondant régulièrement les pairs connectés et en maintenant un état cohérent
- Service novateur qui généralise le P2P

Les limites :
- Pas d'anonymat partout car chaque pair est connu du serveur et des pairs sur lesquels il télécharge.
- Limites habituelles d'un serveur central : problème de disponibilité, de passage à l'échelle (saturation de la bande passante et du nombre de processus). Cas de Napster : facile à fermer.
- Certains pairs peuvent mentir sur leur débit pour ne pas être sollicités.

2.3 Totalement décentralisé : modèle pur

Gnutella : protocole de fichiers partagés, développé en mars 2000 par Justin Frankel et Tom Pepper.

"Gnutella is a decentralized system. There's no single server that tells you all the information of who's got what. So there's no single point in the continuum that you can force to shut down, if you take half of the computers that use Gnutella off the system, the other half will still work just fine." [source http ://www.cnn.com]

Réseau complètement décentralisé, modèle pur de P2P. Chaque élément joue à la fois le rôle de client et de serveur : Servent = **Serv**eur + Cli**ent**. Les messages sont des informations émises par les servents à travers le réseau Gnutella.

Utilisation d'un réseau virtuel non structuré, et recherche par inondation sur ce réseau.

2.3.1 Connexion au réseau

Pour rejoindre le réseau, un client doit connaître au moins un noeud déjà présent sur le réseau (un servent), et à partir de cette connexion, il peut, par le moyen de ping, trouver les adresses d'autres servents.

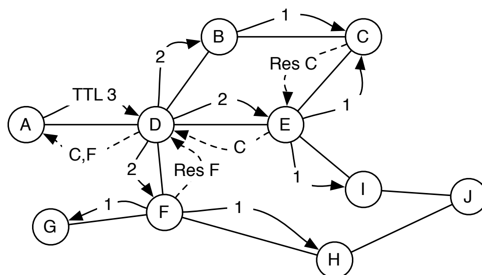
Pour trouver le premier servent et pouvoir rejoindre le réseau Gnutella, on utilise les *Gwebcache*. Certains servents Gnutella fonctionnent également comme des serveurs web. Ils sont ainsi répertoriés par les moteurs de recherche, et on peut obtenir une liste de servents connectés à Gnutella.

Chaque servent maintient une connexion avec un certain nombre d'autres servents, normalement environ 5. Un réseau logique se forme donc sur le réseau global. Un servent qui reçoit un ping doit répondre avec un (ou plusieurs) pong, indiquant son adresse IP et son numéro de port, ainsi que le nombre et la quantité de données partagées, ce qui permet de créer le réseau logique.

2.3.2 Recherche de ressource

Pour chercher une ressource sur le réseau, un servent envoie une requête à chaque servent auxquels il est connecté. Ces derniers retransmettent le message (inondation). Lorsque la ressource est trouvée, son adresse est propagée le long du chemin inverse. Le nombre de noeuds interrogés est contrôlé par un compteur TTL (*Time To Live*), qui indique le nombre de fois que l'on peut retransmettre le message.

Exemple sur le réseau suivant :



Pour l'échange de fichiers, le servent qui a émis la requête se connecte directement au servent qui possède la ressource. Utilisation d'un message *Push* si les données sont derrière un firewall. Avec Gnutella, le téléchargement est impossible si les deux servents sont derrière un firewall.

2.3.3 Avantages et limites

Avantages :
- L'administration est simple, car mutualisée
- La topologie évolue au gré des départs et arrivées de servent
- Disponibilité du réseau
- Très dur à attaquer, on ne peut pas l'arrêter

Cependant,
- Le réseau est rapidement inondé par des ping-pong, ne passe pas bien à l'échelle
- Supporte mal la montée en charge du nombre d'utilisateurs
- Manque de fiabilité dans les requêtes : avec TTL=7, seulement 25% des requêtes aboutissent.
- Recherche de complexité exponentielle.
- Routage par inondation : définitivement pas une méthode optimale, et il n'a pas fallu longtemps avant que d'autres algorithmes plus efficaces émergent.
- Pas spécialement d'anonymat : il est possible de savoir qui possède une certaine ressource par une simple recherche.

Orientation possible vers les superPairs, comme dans une autre version de Gnutella (version 6) et dans Kazaa, ce qui rend le réseau un peu moins robuste mais améliore les performances. C'est un mélange modèle pur/ modèle hybride.

Tables de hachage distribuées pour améliorer le routage.

2.4 Systèmes structurés

Pour résoudre un peu tous les problèmes qui se sont posés dans les systèmes présentés précédemment, je présente maintenant les techniques de systèmes pair à pair basés sur des tables de hachage distribuées.

L'idée est de ne pas avoir de contrôle central ni de rapport hiérarchique, mais de pouvoir rechercher de façon efficace les données et d'être résistant aux défaillances et à la volatilité des pairs.

Ces systèmes fournissent un routage efficace d'un point à un autre du système (pas besoin d'inonder le système), et sont basés sur une notion de connaissance locale : un noeud n'a pas de connaissance globale, mais il peut se rapprocher de la donnée recherchée.

Nous présentons le principe général, en illustrant les grandes idées sur le protocole *Chord*.

2.4.1 Le principe

On utilise une fonction de hachage h, que l'on peut appliquer à la fois aux adresses IP des pairs et aux chaînes ASCII identifiant une ressource disponible sur le réseau. Le résultat est un nombre aléatoire codé sur m bits. m est choisi suffisamment grand pour que la probabilité d'une collision soit suffisamment petite. Typiquement $m = 160$, et on dispose donc de 2^{160} identificateurs.

La fonction de hachage couramment utilisée, et notamment dans *Chord* : SHA-1. Fonction utilisée en cryptographie. Détails dans le Tannenbaum.

L'**identifiant** d'un noeud est obtenu en hachant l'adresse IP, et la **clé** d'une ressource est obtenue en hachant son nom.

Le principe consiste ensuite à stocker, de façon totalement distribuée, l'information $(nom, adresseIP)$ pour chaque ressource, qui associe au nom d'une ressource l'adresse IP du pair qui possède la ressource.

La plupart des identifiants de noeuds ne correspondent pas à des noeuds réels. On définit donc une fonction $successeur(k)$ qui retourne l'identifiant du premier noeud réel $\geq k$ modulo 2^m.

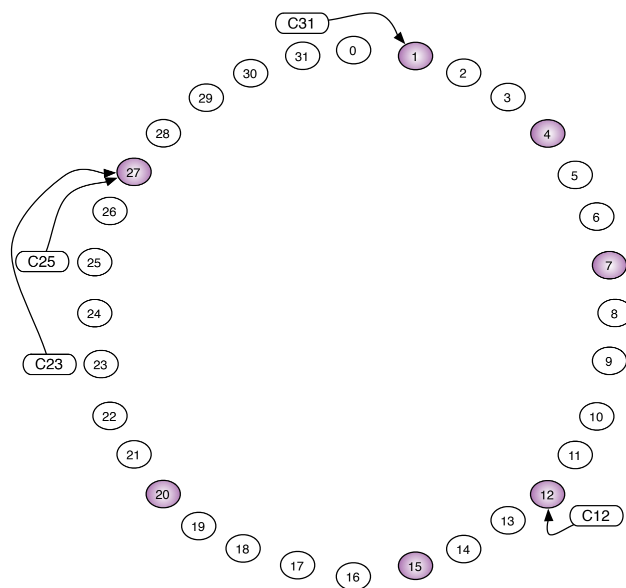
L'information sur la ressource nom est alors stockée sur le noeud d'identifiant $successeur(h(nom))$. L'index est ainsi globalement réparti aléatoirement sur plusieurs noeuds.

Dans Chord, la topologie adoptée est un anneau, et cela peut changer d'un système pair à pair à un autre. Ainsi, par exemple, CAN se base sur un tore à d dimensions, et Tapestry a une topologie dédiée. L'idée de la topologie est d'avoir

- un faible degré pour une mise à jour rapide malgré la volatilité des noeuds
- un faible diamètre pour une recherche performante

Il faut donc trouver un compromis, et Chord obtient à la fois un degré et un diamètre en $\log N$ comme nous allons voir.

Exemple d'anneau de Chord, avec $m = 5$. En blanc, les noeuds virtuels et en grisé, les noeuds réels. Les clés de ressources sont allouées aux successeurs représentés par une flèche.



2.4.2 Recherche d'une ressource dans Chord

Recherche d'une donnée *nom* : le noeud demandeur hache le nom pour obtenir la clé c de la ressource, et en contactant $successeur(c)$, il peut obtenir l'adresse IP du noeud possédant la donnée recherchée. Le problème consiste à contacter $successeur(c)$.

Méthode simple mais peu efficace : chaque noeud conserve l'adresse IP de son successeur réel sur le cercle (le successeur de 1 est 4, le successeur de 4 est 7, ...). Le noeud demandeur peut alors envoyer un paquet de requête contenant son adresse IP et la clé de la donnée qu'il recherche. Le paquet circule dans le cercle jusqu'à arriver au successeur de la clé recherchée. Ce noeud regarde alors s'il a des informations correspondant à la clé, et les renvoie directement au noeud demandeur vu qu'il a son adresse IP. Coût de la recherche en N (N étant le nombre de participants).

Amélioration en conservant à la fois le successeur et le prédécesseur... Mais même avec deux sens possibles pour la recherche, cette méthode est très inefficace dans les grands systèmes P2P.

... faire un exemple sur l'anneau au tableau...

Utilisation de **tables de repérage** (*table finger*) : on conserve plus d'information à chaque noeud pour accélérer la recherche.

- Chaque noeud k dispose d'une table de routage à m entrées.
- La i^e entrée ($i = 0..m-1$) contient l'identité du premier noeud réel qui se situe au moins 2^i après le noeud courant k, obtenu par : $successeur\big(k + 2^i \text{ (modulo } 2^m)\big)$.

– Une entrée contient l'identifiant et l'adresse IP du noeud concerné ($identifiant[i]$ et $adresse[i]$ pour l'entrée i).

...Construction des tables de repérage sur l'exemple...
Au noeud 1 : $1+1 \rightarrow 4$, $1+2 \rightarrow 4$, $1+4 \rightarrow 7$, $1+8 \rightarrow 12$, $1+16 \rightarrow 20$ et ainsi de suite pour les autres noeuds.

...Exemples de recherche au moyen des tables de repérage...

1. Recherche de 23 depuis le noeud 20 : clé entre moi et mon successeur donc je cherche le résultat directement sur le noeud 27 dont je connais l'adresse IP
2. Recherche de 14 depuis le noeud 1 : 14 n'est pas entre 1 et 4 donc on examine la table de repérage. Entrée située juste avant 14 : le noeud 12. On se retrouve dans le cas 1.
3. Recherche de 16 depuis 1 : même départ mais 16 n'est pas situé entre 12 et son successeur donc 12 procède comme dans l'exemple 2 pour transmettre la requête. Et ainsi de suite.

On se rapproche ainsi de plus en plus du noeud recherché (si le noeud est loin de moi, je sais que le noeud plus proche possède plus d'information sur les noeuds proche de lui, et donc sur l'objectif). A chaque itération, on diminue par deux la distance entre le noeud qui recherche et l'objectif.

L'adresse IP du noeud qui possède l'information sur la clé recherchée est ensuite retransmise au noeud qui effectuait la requête en suivant le chemin inverse.

Algorithmes :

– Recherche dans la table de repérage du noeud k le plus grand prédécesseur de id :
  Pour i de $m-1$ à 0, si $identifiant[i] \in [k, id]$ alors renvoyer $identifiant[i]$.
  Renvoyer k (si l'on n'a pas trouvé d'identifiant dans la table).
– Recherche du successeur de id par le noeud k :
  Si $id \in]n, successeur(n)]$ alors retourner $successeur(n)$.
  Sinon, on recherche le plus grand prédécesseur de id (noté n') à l'aide de l'algo précédent et on cherche le successeur de id sur le noeud n'.

Chaque noeud stocke seulement de l'information sur un petit nombre d'autres noeuds, et la plupart sont assez proches en termes d'identifiant de noeud.

Propriétés

Réseau à N noeuds :

– Avec une forte probabilité il faut contacter $O(\log N)$ noeuds pour trouver un successeur ($O(\log N)$ messages).
  La complexité en pratique est d'environ $\frac{1}{2} \log N$.
– Chaque noeud conserve des informations sur $O(\log N)$ autres noeuds.
– Lorsqu'un noeud arrive ou part, la mise à jour des informations requiert $O(\log^2 N)$ messages.

– Hachage consistant : avec une forte probabilité, le départ ou l'arrivée d'un noeud oblige à relocaliser seulement une fraction en $O(1/N)$ des clés.

Preuves dans les papiers Chord. On revient un peu plus en détail sur la notion de hachage consistant et sur les effets liés à la volatilité des noeuds.

2.4.3 Répartition des clés : notion de hachage consistant

L'attribution des clés au successeur permet aux noeuds d'arriver et de quitter le système avec un dérangement minimal.

Lorsque k arrive dans le système, ce noeud récupère certaines des clés localisées auparavant sur $successeur(k)$. Lorsque k quitte le système, toutes ses clés sont relocalisées sur $successeur(k)$. Et il n'y a pas d'autres modifications à effectuer. (Faire tourner l'exemple).

On peut alors prouver les propriétés suivantes pour un système composé de N noeuds et K clés. Avec une très forte probabilité,

– Chaque noeud est responsable d'au plus $(1+\epsilon)\frac{K}{N}$ clés
– Lorsqu'un noeud rejoint ou quitte le système, $O(\frac{K}{N})$ clés sont relocalisées, et seulement vers/depuis le noeud qui arrive/part.

Pour la topologie Chord, $\epsilon = O(\log N)$, et on peut également rendre ϵ arbitrairement petit en créant sur chaque noeud $O(\log N)$ *noeuds virtuels* possédant chacun leur propre identificateur.

Cf papiers "consistent hashing" pour les détails.

Remarque : j'ai utilisé le terme "avec une très forte probabilité". Ceci est lié à la fonction de hachage et suppose une répartition aléatoire des clés. On pourrait imaginer un adversaire qui choisit des clés donnant toutes la même valeur...

Utilisation de la fonction de hachage SHA-1 : pour générer des clés en collision, il faudrait en quelque sorte inverser, ou décrypter SHA-1. Ceci est un problème difficile, donc les propriétés énoncées sont vérifiées en suivant cette assomption.

2.4.4 Ajout de noeuds et stabilisation

Arrivée d'un noeud k. Ce nouveau noeud doit contacter un noeud existant et lui demander de rechercher l'adresse IP de $successeur(k)$ (processus de recherche expliqué auparavant). k contacte ensuite le noeud trouvé, lui demande quel est son prédécesseur, et demande à son successeur et prédécesseur de l'insérer entre eux sur le cercle. De plus, le noeud successeur lui transmet les clés de l'intervalle $]pred, k]$.

Il faut également s'assurer que les tables de repérage sont à jour pour conserver une recherche rapide. Cela s'effectue en exécutant sur chaque noeud un processus en arrière plan qui recalcule périodiquement chaque entrée de repérage en utilisant la fonction de recherche de successeur d'une clé.

Pour s'assurer de la stabilité du système, on vérifie également périodiquement le successeur immédiat de chaque noeud en demandant à son successeur quel est son prédécesseur, et on s'assure que le prédécesseur est à jour également.

Ces mises à jour périodiques permettent de stabiliser le système. Dès que les pointeurs successeur seront à jour, la fonction de recherche renverra le bon résultat, et on peut montrer qu'au bout d'un certain temps après la dernière arrivée de nouveau noeuds, les pointeurs successeurs forment un anneau de tous les noeuds dans le système, même si les noeuds arrivent pendant les opérations de stabilisation.

Après l'arrivée de nouveaux noeuds, efficacité de la recherche ?

- Si toutes les entrées des tables de repérage sont raisonnablement correctes, succès en $O(\log N)$.
- Si tous les pointeurs successeurs sont corrects, la recherche aboutit mais peut prendre plus de temps. En effet, la recherche dans la table de repérage ne connaît pas les nouveaux noeuds et va donc un peu sous-estimer le saut. Cependant, on termine avec une recherche linéaire à l'aide des pointeurs successeur. Avec le processus de stabilisation, les recherches sont en $O(\log N)$ avec une forte probabilité.
- Possibilité d'échec si les noeuds dans la région considérée ont des pointeurs incorrects ou bien si les clés n'ont pas encore migré d'un noeud à l'autre.

Le processus de stabilisation fonctionne également dans le cas d'arrivée simultanée de plusieurs noeuds.

2.4.5 Départ de noeuds et tolérance aux pannes

Lors du départ d'un noeud du système, il remet ses clés à son successeur et il informe également son prédécesseur pour que celui-ci puisse changer son pointeur successeur. Le processus de stabilisation met à jour les tables de repérage comme pour le cas de l'ajout.

Le problème est un peu plus corsé lorsqu'un noeud tombe en panne. Si k tombe en panne, les noeuds contenant k dans leur table de repérage doivent pouvoir retrouver le successeur de k. Notamment, le prédécesseur de k n'a plus de successeur valide.

L'idée consiste à conserver, en plus de son successeur immédiat, une liste des r successeurs directs. Cela permet à un noeud de sauter par dessus $r-1$ noeuds contigus défaillants et de continuer sa participation sur le cercle. Alors, la probabilité que r noeuds soient simultanément en panne est p^r, p étant la probabilité qu'un noeud soit en panne.

On a alors les propriétés suivantes (preuves dans les articles de recherche). Soit $r = O(\log N)$ et un système initialement stabilisé. Si chaque noeud tombe en panne avec une probabilité $\frac{1}{2}$, alors

- avec une forte probabilité la recherche renvoie le plus proche successeur en vie de la clé recherchée ;
- le temps d'exécution pour effectuer cette recherche est en $O(\log N)$.

L'utilisation de liste de successeurs permet aussi à un logiciel de plus haut niveau de dupliquer les données. Les applications généralement dupliquent les informations sur k noeuds suivant la clé (pour éviter la perte d'information lors de panne), et la liste des successeurs permet de savoir quand des noeuds

partent et arrivent, pour pouvoir générer au besoin des nouveaux répliquas de l'information.

2.4.6 Conclusion

Validation expérimentale du protocole Chord : très bons résultats qui concordent avec les théorèmes énoncés.

Avantages de Chord : simplicité, possible de prouver la correction et les performances du protocole même dans le cas d'arrivées et de départs simultanés. Bonnes propriétés de passage à l'échelle, recouvre bien des défaillances, et répond la plupart du temps correctement aux requêtes même dans une phase de récupération de défaillance et de stabilisation. Si une requête n'aboutit pas, on réessaie après un petit temps de pause pour permettre au système de se stabiliser.

Autres protocoles qu'il peut être intéressant de regarder (informations sur le web) :

- CAN : idée similaire à Chord mais la topologie diffère : tore à d dimensions. Chaque noeud conserve alors $O(d)$ informations, et le coût de la recherche est en $O(dN^{\frac{1}{d}})$. Pour $d = \log N$ les performances sont similaires à Chord, mais d n'évolue pas avec la taille du réseau (et N est amené à varier).
- Tapestry/Pastry : autre topologie
- Freenet : accent sur les problèmes d'anonymat
- BitTorrent : partage équitable – pendant le téléchargement d'un fichier, un noeud doit mettre à disposition ce qu'il a déjà téléchargé pour évité de surcharger le noeud possédant initialement le fichier.

Certains systèmes P2P ont été développés suite à l'essor de l'ADSL et aux problèmes de partage équitable des données. Je développe un peu ce problème maintenant.

2.5 Vers un partage équitable

Développement de l'ADSL : débit asymétrique, le débit de sortie des clients est 10 à 20 fois plus faible que leur débit de téléchargement. Ce faible débit freine l'ensemble du réseau, car indépendamment de sa vitesse de connexion, un ordinateur sera limité par le débit imposé par le pair qui envoie les données.

La notion de téléchargement multiple a été introduite pour compenser cette asymétrie.

2.5.1 Téléchargement multiple

eDonkey, eMule...

Possibilité de télécharger un fichier à partir de plusieurs sources. Un fichier est découpé en petites tranches, il peut être téléchargé morceau par morceau à partir de plusieurs pairs. Solution partielle : ne peut pas permettre à tous les pairs en même temps d'augmenter leur vitesse de téléchargement, car au total les capacités du réseau restent déséquilibrées.

Ce fractionnement des données permet également d'augmenter la durée de vie d'un fichier dans le réseau : persistance des fichiers dans le temps. Si on télécharge un fichier que possède un pair dans son intégralité, lorsque le pair se déconnecte, on doit trouver une autre source et recommencer le téléchargement depuis le début. En revanche, avec le téléchargement multiple, un fichier peut même être disponible sur le réseau alors que personne ne le possède dans son intégralité.

Problème de coopération : problématique essentielle. Nombreux utilisateurs se servent du réseau uniquement pour acquérir de nouvelles données, mais ne partagent pas leurs données. Pour pallier à ce type de problèmes, protocoles de type BitTorrent.

2.5.2 BitTorrent et le partage équitable

Les bouts de fichiers déjà téléchargés sont partagés lorsqu'un utilisateur est en cours de téléchargement du fichier. Cependant, dans la plupart des cas, à la fin du téléchargement, l'utilisateur se déconnecte du réseau P2P ou bien arrête de partager le fichier qu'il vient de récupérer.

BitTorrent vise à inciter à la coopération entre les pairs en suivant un principe simple : il faut partager des morceaux d'un fichier déjà acquis pour pouvoir en obtenir de nouveaux plus rapidement. BitTorrent se concentre sur le téléchargement d'un fichier (l'indexation des fichiers est prise en charge par un autre protocole).

Chaque pair BitTorrent, en tant que serveur, donne en priorité aux clients qui lui ont récemment envoyé des morceaux du fichier. Ainsi un pair, en tant que client, a tout intérêt à donner ce qu'il possède pour recevoir le plus souvent possible des morceaux de fichier.

BitTorrent est utilisé pour distribuer certaines versions de Linux. Il est également utilisé par des sociétés éditrices de jeux, pour la diffusion de leurs logiciels et versions d'essai.

1. Chaque client se connecte à une centaine d'autre, et échange avec eux la liste des blocs qu'il possède (*bitmap*). Un noeud spécial, le *tracker*, est chargé de mettre les clients intéressés par le téléchargement du fichier en relation.
2. Un client répète en continu le processus suivant jusqu'à ce que le programme soit arrêté par l'utilisateur (une fois le fichier récupéré, le client continue de le redistribuer).
   (a) Chaque client demande un bloc à chacun des autres clients (les blocs qui paraissent plus rare sont demandés de préférence).
   (b) Chaque client ne sert qu'une partie des demandes qui lui sont adressées selon le principe d'*un prêté pour un rendu* : les trois clients qui lui ont le plus donné dans les 10 dernières secondes sont servis de préférence, ainsi qu'une quatrième demande sélectionnée au hasard.

Heuristique d'*un prêté pour un rendu* dans le choix de redistribution du fichier : heuristique connue en théorie des jeux pour résoudre le dilemme du "téléchargeur égoïste".

Conclusion

On a ainsi vu comment le P2P sert au partage de fichiers, utilisant divers techniques algorithmiques pour l'accès au fichier (premier problème : trouver le fichier) et également en utilisant sur un même fichier des techniques de partage équitable (assurer la persistance des fichiers dans le temps et un téléchargement rapide et fiable).

Cependant, **Le pair à pair, ce n'est pas que l'échange de fichiers**, il existe de nombreuses autres applications. Pour n'en citer que quelques unes :
– Partage de puissance de calcul et espace mémoire
– Logiciels de messagerie instantanée / téléphonie IP
– Listes de diffusion : mécanisme de recherche persistante

On développe maintenant les techniques algorithmiques visant à développer le mécanisme de recherche persistante.

2.6 Mécanisme de recherche persistante

Dans les systèmes étudiés précédemment, on s'est concentré sur une recherche *instantanée* : les objets sont stockés et les messages qui circulent sont les requêtes.

On va s'intéresser ici aux recherches persistantes : les requêtes sont stockées, et les messages sont des publications sur les objets. On dispose alors de plusieurs types de recherche : exactes ou par plages de valeurs.

2.6.1 Publication/abonnement

Système de notification asynchrone, et découplage des publieurs et des abonnés.

- Les abonnés enregistrent leurs intérêts
- Les publieurs publient des événements

Publieurs $\rightarrow$ [système publication/abonnement] $\rightarrow$ Abonnés

Problème de passage à l'échelle, pour faire la correspondance entre les publieurs et les abonnés : nécessité d'un bon paradigme de communication.

Classification des systèmes :

- basés sur les sujets (*topic-based*)
  Exemple : sujet=vente-appart.
  Cela revient à un multicast (transmission simultanée à plusieurs noeuds) au niveau applicatif
- basés sur les attributs ou le contenu (*attribute-based, content-based*) : filtre le contenu du message.
  Exemple1 : (ville=Lyon) (type=T2)
  Exemple2 : (ville=Lyon || Grenoble) && (type=T3 || prix $< 200,000$ EUR)

Utilisation de *réseaux structurés* pour réaliser un système publication-abonnement :

- Basé sur les sujets : associer une clé au nom d'un groupe. Exemple système *Scribe*. L'ID d'un sujet est donc associé à un noeud du réseau, et lorsqu'on s'abonne à un sujet, on cherche une route vers ce noeud. Le noeud correspondant à un sujet peut alors router vers tous les abonnés en utilisant un arbre de multicast (racine= le noeud correspondant au sujet). Scribe est basé sur Pastry.
- Basé sur les contenus : on doit effectuer une correspondance entre l'espace des attributs et l'espace des identificateurs. Un attribut par pair : risque d'avoir trop de clés pour un pair si attribut très populaire. Un couple (attribut,valeur) par pair : pas toujours possible (champ des valeurs). Il faut faire un compromis efficacité/expressivité.

On va étudier maintenant deux systèmes. Meghdoot est un réseau structuré basé sur CAN. Sub-2-Sub est, à l'opposé, faiblement structuré.

2.6.2 Meghdoot

Système de publication/abonnement basé sur le contenu, et mis en oeuvre au dessus du réseau P2P CAN. Système $S = \{A_1, ..., A_n\}$ où A_i est un attribut. Attribut A_i : {nom : type, L_i, H_i} (un nom, un type et des bornes L_i pour le min et H_i pour le max). Le type peut être entier, réel, caractère, chaîne de caractères, ...

Abonnement : conjonction de prédicats sur un ou plusieurs attributs. Opérateurs (=, $<$, $>$) et constantes (ou plages de valeur).
Ex abonnement : $s = (A_1 \geq v_1) \wedge (v_2 \leq A_2 \leq v_3)$

Evenement : ensemble d'égalité sur les attributs.
$e = \{A_1 = c_1, ..., A_n = c_n\}$.
Un événement e correspond à un abonnement s si chaque prédicat de s est vérifié par e.

Le système doit :
- stocker les abonnements
- router les événements aux abonnements concernés

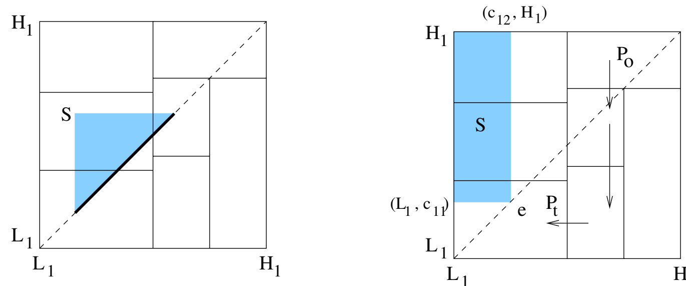
On structure l'espace en un espace cartésien de dimension $2n$ (pour n attributs). Attribut A_i de domaine $[L_i, H_i]$: correspond aux dimensions $2i-1$ et $2i$. (rectangle de dimensions $[L_i, H_i]$).

CAN : similaire dans le principe à Chord, découpage en zones et un noeud appartient à une zone donnée. (zone= correspond à plusieurs noeuds virtuels).

Un abonnement $s = (l_1 \leq A_1 \leq h_1) \wedge ... \wedge (l_n \leq A_n \leq h_n)$. s est associé au point $[l_1, h_1, ..., l_n, h_n]$. Lorsqu'un utilisateur P_O souscrit un abonnement, il envoie ses informations au pair correspondant à la zone de cet abonnement, P_e (routage de proche en proche). P_e conserve les informations sur P_O (adresse IP, adresse e-mail...) et les coordonnées de l'abonnement.

L'événement e est associé également à un point $[c_1, c_1, c_2, c_2, ..., c_n, c_n]$, et s est affecté par e si pour tout i $l_i \leq c_i \leq h_i$.

Exemple : projection d'un abonnement sur l'axe des événements, et zone des abonnements concernés par un abonnement.

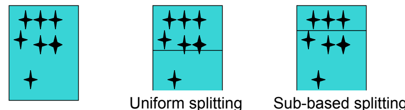


Lors de l'introduction d'un événement dans le système par un pair P_0, P_0 route l'événement vers le pair P_t qui gère la zone du point de l'événement. L'événement est alors propagé depuis P_t vers tous les pairs qui sont dans la région touchée par l'événement.

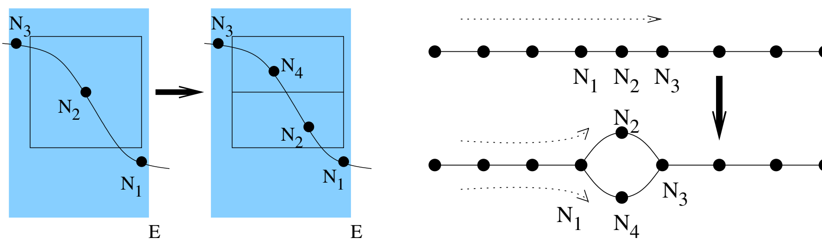
L'espace logique est divisé en zones (comme dans CAN), et un pair est associé par zone et gère cette zone. Table de hachage distribuée (DHT) multidimensionnelle (cf CAN). Connaissance des coordonnées de sa propre zone ainsi que des zones voisines, et connaissance des adresses IP des noeuds gérant les zones voisines (information utilisée pour le routage).

Gestion des pairs et équilibrage de charge

Un pair gère les abonnements associés à sa zone, et propage les événements. La charge d'une zone est donc proportionnelle au nombre d'abonnements stockés dans la zone. Pour équilibrer la charge, découpe des zones dépendante du nombre d'abonnements. C'est un meilleur équilibrage de charge que la découpe uniforme, qui donnait de bons résultats pour la recherche d'objets comme dans CAN, Chord... (le hachage des noms des objets donne un résultat homogène, contrairement aux abonnements).



On peut aussi agir sur les événements pour équilibrer la charge : création de chemins de propagation alternatifs lorsqu'un noeud est surchargé par les événements qu'il doit router. Pour cela, on réplique une zone, et le routage se fait suivant un algorithme de Round Robin : on choisit une fois la zone originale, une fois la zone répliquée... Plus efficace que le partitionnement d'une zone, qui ne réduira pas le problème de la charge du noeud concerné.



Arrivée d'un pair dans le système : l'affecter à une zone chargée (garder et mettre à jour une liste des pairs surchargés en chaque noeud...). Choisir la réplication ou le partitionnement suivant le type de charge de la zone choisie.

Départ d'un pair : regarde si il est répliqué, si oui informe les autres pairs de sa zone de son départ. Sinon demande à ses voisins qui veut bien reprendre sa zone.

Panne. Messages réguliers échangés entre voisins, notamment pour mettre à jour les charges. Lorsqu'un noeud découvre que son voisin est en panne, il récupère sa zone (ou la confie à un de ses voisins). Problème de perte des abonnements du noeud défaillant, mais technique de sauvegarde des abonnements

dans la zone symétrique (avec $l_i > h_i$) vu que l'on n'utilise que la moitié de l'espace en fait.

Conclusion Meghdoot

- Bonnes propriétés de passage à l'échelle, les techniques d'équilibrage de charge sont très efficaces en pratique.
- Cependant, problèmes inhérents aux réseaux structurés : peu de flexibilité, et attribution de zones de l'espace de nommage.

Tous les détails dans le papier sur la page web. On présente maintenant une approche alternative, Sub-2-Sub.

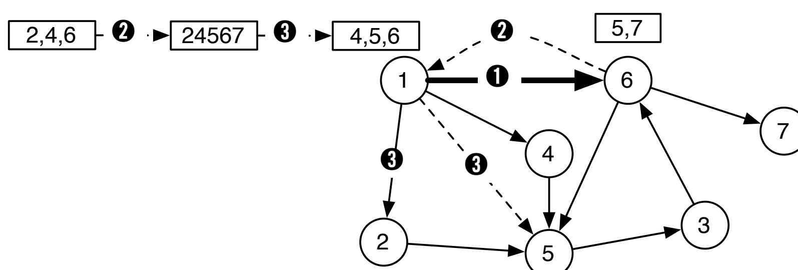
2.6.3 Sub-2-Sub

Comme Meghdoot, Sub-2-Sub est un système de notification asynchrone d'événements basé sur le contenu. Ce système supporte des requêtes exactes et par plage de valeurs, et vise à pallier l'inadéquation des réseaux P2P structurés pour le cas publication/abonnement.

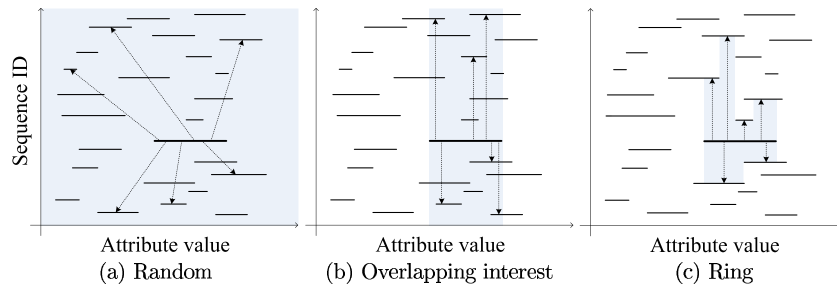
Réseau P2P non structuré de construction *épidémique* dans un espace d'attributs multidimensionnel. Le réseau est faiblement structuré, mais cherche à regrouper automatiquement les abonnements similaires. Il supporte les abonnements multi-attributs.

Algorithme épidémique : périodiquement chaque pair échange de l'information avec un de ses voisins "logiques" :

1. sélection de la cible parmi les voisins (ici, parmi ses voisins, 1 choisit 6)
2. communication : échange des informations (1 récupère la liste des voisins de 6 et les combine à la liste de ses voisins)
3. traitement de l'information reçue : le pair recalcule l'espace de ses voisins en utilisant des facteurs de sélection (1 garde comme voisins 4,5,6 uniquement).



Les voisins sont sélectionnés selon différents critères, et classifiés dans différents groupes. On associe à chaque noeud un ID tiré aléatoirement, ce qui permet de représenter schématiquement le graphe des noeuds, représentés par leur ID et la valeur de leurs attributs.



(a) Random (b) Overlapping interest (c) Ring

On dispose alors de :

- Liens aléatoires (comme pour les algorithmes épidémiques classiques) : liens vers des noeuds choisis aléatoirement, pour éviter la déconnexion de certains noeuds du système.
- Liens d'intérêts : voisins choisis parmi tous ceux qui possèdent une même valeur d'attribut. Ces liens sont utilisés pour accélérer la dissémination d'un événement.
- Liens de l'anneau : les plus proches voisins pour toutes les valeurs d'attribut d'un noeud donné. Dissémination d'un événement de proche en proche.

Algorithmes de dissémination des événements, création d'un abonnement, ajout/disparition de pairs, ... Mais nous ne rentrons pas dans les détails ici, et travaux de recherche en cours. Plus d'informations dans le papier correspondant.