VERS DE NOUVEAUX MODELES PEER-TO-PEER

Pierre François

Facultés Notre Dame de la Paix, Namur

Résumé Dans cet article, nous expliquons les faiblesses majeures des systèmes Peer-to-Peer actuels. Nous montrons en quoi ces faiblesses peuvent compromettre l'évolutivité quantitative de ces systèmes. Nous en déduisons une série d'exigences pour l'élaboration d'un système Peer-to-Peer performant et "scalable".

Nous introduisons ensuite la notion de table de hachage distribuée (DHT) en proposant un modèle général pour celle-ci. Nous déduisons le potentiel en terme de performances d'un système Peer-to-Peer basé sur ce principe.

Sur base de ce modèle général, nous discutons la pertinence d'une comparaison de performance des DHT avec les systèmes existants. Pour ce faire, nous montrons que les services proposés par les DHT dans leurs versions actuelles sont différents des services rendus par des applications telles que Gnutella ou Napster. Nous proposons une voie d'adaptation des DHT visant à rendre ces services identiques sans en compromettre la performance.

Enfin, nous présentons deux solutions existantes (Can et Chord) en montrant la manière à laquelle celles-ci répondent à notre modèle général.

1 Introduction

La multiplication incessante des machines personnelles connectées à l'Internet et l'accroissement constant des volumes de stockage de celles-ci ont contribué au succès d'applications telles que Napster et Gnutella. Au delà du battage médiatique réalisé autour des problèmes juridiques liés au partage d'oeuvres d'art digitalisées, des doutes ont été émis, ces dernières années, au sujet de la capacité de ces systèmes à s'adapter à leur évolution en terme de nombre de clients. Certains ISP (Internet Service Provider) ont également émis des craintes au sujet de la quantité de trafic engendré par les clients Gnutella sur leur réseaux. C'est dans ce contexte que de nouveaux modèles Peer-to-Peer, censés remédier à ces problèmes de performance, ont vu le jour. Ces nouveaux modèles disposent, pour la plupart, de caractéristiques communes dans leur mode de fonctionnement. Cet aspect a poussé la communauté P2P à taxer ces modèles du nom de DHT, pour "Distributed Hash Table", tant leur interface est semblable à une table de hachage dont le contenu est distribué entre plusieurs hôtes.

Il semble que le monde scientifique a répondu aux "appels de détresse" des ISP en proposant des modèles théoriques performants. Toutefois, l'applicabilité de ces modèles peut paraître compromise au vu de certains aspects critiques

des DHT. Si leurs auteurs de DHT ne semblent pas destiner leur modèle à des applications de partage de fichiers personnels (nous expliquerons pourquoi cela nous semble irréalisable), mais plutôt à des applications distribuées comme les implémentations du modèle de publish/subscribe, le "file mirroring" ou encore les systèmes de fichiers distribués, leur but est d'aboutir à des applications dont le fonctionnement serait distribué sur Internet. Cet aspect ajoute, par rapport aux applications distribuées sur un réseau local, un nombre importants de contraintes supplémentaires auxquelles les scientifiques ne semblent pas avoir porté une grande attention.

Dans cet article, nous commencerons par justifier brièvement les craintes émises au sujet de la "scalabilité" des systèmes actuels. Ensuite, nous proposerons un modèle général de table de hachage. Nous explicitons, sur base de ce modèle, les aspects des DHT qui constituent des faiblesses potentielles pour les implémentations qui en découleront. Nous mettons en évidence les aspects contraignants pour l'implémentation d'un "file-sharing system" basé sur une DHT. Nous tenterons ensuite d'identifier quelques adaptations permettant de réduire l'impact de ces contraintes. Enfin, nous présenterons le fonctionnement théorique de Can et de Chord, en expliquant la manière à laquelle ils correspondent à notre modèle de table de hachage distribuée. Cette présentation permettra de réaliser la performance et la fragilité potentielle de ce type de système.

2 Faiblesses des systèmes actuels

Dans cette section, nous explicitons les principales justifications des craintes émises par les ISP et le monde scientifique en général en critiquant les modèles sur lesquels reposent ce type de système. Pour ce faire, nous discutons du modèle de flooded requests sur lequel est basé Gnutella et le modèle de répertoire centralisé, principe de fonctionnement du système Peer-to-Peer hybride Napster.

2.1 Le modèle "flooded requests"

En se penchant sur le trafic généré par les clients Gnutella et sur le principe de fonctionnement du mécanisme de recherche dont disposent ces clients, on peut se rendre compte que l'accroissement du trafic induit par l'arrivée de nouveaux noeuds, et donc de nouveaux émetteurs de requêtes, est trop important pour que l'on puisse considérer le système Gnutella comme "scalable".

Lorsqu'un hôte connecté au système effectue une requête de recherche, il l'envoie à tous les hôtes qui lui sont voisins. Ceux-ci propagent à leur tour cette requête selon la même méthode. Ainsi, lorsqu'un hôte reçoit une requête, il la transmet à tous les hôtes auxquels il est connecté sauf à celui qui lui a envoyé la requête. Si, dans le graphe représentant la topologie d'un réseau gnutella, il existe plusieurs chemins entre deux noeuds, on peut constater qu'une requête passant par un de ces noeuds empruntera tous les chemins possibles pour arriver à l'autre noeud.

Fig. 1. Propagation d'une requête selon le modèle de "flooded requests".

Un mécanisme fonctionnant sur base d'un champ "Time to live" similaire au champ TTL des paquets IP est utilisé. Lors de chaque retransmission, la valeur de ce champ est décrémentée et un paquet ayant une valeur de champ TTL nulle n'est plus retransmis. Ce mécanisme constitue un moyen de limitation de portée des requêtes afin de ne pas saturer le réseau. On peut remarquer que cela implique le non déterminisme du système.

Bien que ce mécanisme de TTL constitue un moyen de protection contre le bouclage des requêtes (c'est d'ailleurs la fonction principale du champ TTL des paquets IP), une méthode plus efficace, en terme de consommation de bande passante, a été implantée dans Gnutella. La méthode est très simple : lorsqu'un noeud émet initie une requête, il l'identifie grâce à une valeur numérique. Lors de chaque traitement de requête par un noeud, celui-ci enregistre l'identifiant de la requête dans une cache. Avant de retransmettre une requête à ses pairs, le noeud vérifie simplement, sur base de cette cache, qu'il ne l'a pas déjà retransmise auparavant. Si la requête a déjà été retransmise, elle est ignorée.

Nous pouvons remarquer que cette solution est mieux adaptée pour Gnutella étant donné que la réception d'une requête déjà reçue ne constitue pas un indice d'erreur de configuration du système (comme la réception d'un paquet déjà retransmis sur un réseau IP). En effet, on peut comprendre, au vu de la méthode de propagation des requêtes, que ce genre de phénomène est assez fréquent et totalement prévu au sein de ce type de modèle.

Notons que l'algorithme d'identification des requêtes est tel que la probabilité que deux noeuds émettent une requête avec le même identifiant est extrêmement faible. Cet aspect combiné avec le champ TTL des requêtes assure le non bouclage de celles-ci et ce, avec une efficacité reconnue.

Le nombre de clients joignables croît géométriquement avec la valeur du champ TTL spécifiée dans les paquets Gnutella. Le nombre de connexions que maintient un client jouit de la même influence sur l'accessibilité. Tout utilisateur cherchant à maximiser ses capacités de recherche dispose donc de deux éléments puissants lui permettant d'arriver à ses fins. Si le nombre d'hôtes joignables aug-

mente géométriquement avec ces deux paramètres, le trafic engendré augmente de la même façon.

Jordan Ritter, dans [1], illustre ces observations en supposant que tous les utilisateurs disposent des mêmes paramètres et qu'une requête nécessite un paquet IP de 83 bytes. Le premier tableau montre l'importance du nombre de connexions et du TTL des requêtes pour l'accessibilité des hôtes. Le second montre l'impact de ces paramètres sur le trafic engendré par un requête. Notons que Jordan Ritter est un des principaux créateurs du système Napster. Ses dires semblent toutefois confirmés par la plupart des recherches sur le sujet comme [2].

	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$
$N=2$	2	4	6	8	10	12	14	16
$N=3$	3	9	21	45	93	189	381	765
$N=4$	4	16	52	160	484	1.456	4.372	13.120
$N=5$	5	25	105	425	1.705	6.825	27.305	109.225
$N=6$	6	36	186	936	4.686	23.436	117.186	585.936
$N=7$	7	49	301	1.813	10.885	65.317	391.909	2.351.461
$N=8$	8	64	456	3.200	22.408	156.864	1.098.056	7.686.400

Tab. 1. Nombre maximum d'utilisateurs joignables (T=TTL N=Nombre de connexions)

	$T=1$	$T=2$	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$
$N=2$	166	332	498	664	830	996	1.162	1.328
$N=3$	249	747	1.743	3.735	7.719	15.687	31.623	63.495
$N=4$	332	1.328	4.316	13.280	40.172	120.848	362.876	1.088.960
$N=5$	415	2.075	8.715	35.275	141.515	566.475	2.266.315	9.065.675
$N=6$	498	2.988	15.438	77.688	388.938	1.945.188	9.726.438	48.632.688
$N=7$	581	4.067	24.983	150.479	903.455	5.421.311	32.528.447	195.171.263
$N=8$	664	5.312	37.848	265.600	1.859.864	13.019.712	91.138.648	637.971.200

Tab. 2. Trafic généré (en Bytes) (T=TTL N=Nombre de connexions)

Chaque hôte dispose donc d'un pouvoir sensible sur le fonctionnement du système et sur sa capacité à supporter l'arrivée de nouveaux noeuds. En effet, si les clients ont tendance à maximiser le champ TTL et le nombre de connexions qu'ils peuvent établir, on peut imaginer que le système risque, à terme, de s'étouffer dans la masse de trafic qu'il engendre.

En outre, ce type de mécanisme ne permet pas de fournir de borne raisonnable sur le temps de résolution d'une requête ou sur le nombre de pairs nécessaires à l'atteinte de la ressource recherchée. Le système est indéterministe en ce sens qu'un hôte n'est pas assuré qu'une réponse positive à une requête lui sera retournée, même si un hôte tiers présent dans le système est en mesure de

lui fournir. Cela signifie, dans le cas particulier de Gnutella, qu'un noeud effectuant une requête n'est pas sûr de recevoir une réponse même si un ou plusieurs noeuds présents à ce moment dans le système disposent de fichiers répondant aux critères spécifiés dans la requête.

2.2 Le modèle de répertoire centralisé

Napster constitue le plus connu des répertoires centralisés, son comportement est décrit dans [3]. Le débat sur l'opportunité de la classification de ce modèle dans les modèles P2P nous conduit à considérer ce modèle comme un système P2P hybride. Selon ce modèle, les membres du système se connectent à un répertoire commun où ils enregistrent les ressources dont ils disposent. Lors d'une demande d'accès à une ressource, un noeud contacte le répertoire et fournit les critères correspondant à la ressource recherchée. Le répertoire parcourt les enregistrements qu'il contient et élit le ou les pairs susceptibles de répondre le mieux aux critères de recherche. Le noeud demandeur peut ainsi contacter de manière directe le ou les noeuds qui partagent la ressource.

Le fait que toute l'information concernant les ressources soit concentrée en un point semble constituer un défaut majeur du modèle en terme de "scalabilité". En effet, le serveur qui répertorie les ressources doit supporter une bande passante plus importante au fur et à mesure que le nombre de requêtes par intervalle de temps croît. De plus, la capacité de stockage de ce même serveur doit également augmenter de façon linéaire par rapport à l'augmentation du nombre de ressources partagées. Ceci tend à rendre plus ardue l'évolution quantitative de tels systèmes.

3 Une solution, les "Distributed Hash Tables"

Les craintes émises dans la section précédente ont motivé l'élaboration de modèles de systèmes Peer-to-Peer susceptibles de fournir un service ne souffrant pas des défauts supposés des applications actuelles. Dans cette section, nous présentons quelques exigences reconnues comme fondamentales en vue de l'élaboration de nouveaux systèmes Peer-to-Peer.

Partant de la constatation que la plupart des prototypes proposés actuellement peuvent être vus comme des implémentations distribuées d'une table de hachage, nous en proposons ensuite un modèle général.

Enfin, nous expliquons pourquoi nous pensons que les performances de ce type de système (dans leurs versions actuelles) ne peuvent être comparées avec les performances de systèmes tels que Gnutella.

3.1 Besoins

Le développement de systèmes Peer-to-Peer déterministes, efficaces en terme de trafic engendré et robustes vis à vis de l'évolution du nombre d'hôtes présents

semble donc intéressant au vu de l'engouement incessant pour les systèmes Peer-to-Peer et des faiblesses des implémentations actuelles.

Ces systèmes doivent en outre être conçus de manière à limiter l'influence d'un noeud sur leur comportement global. Il ne faut en aucun cas qu'un noeud puisse bloquer le système, de manière volontaire ou non.

Si nous excluons la possibilité d'utiliser des services reposant sur des répertoires centralisés ou sur une diffusion massive de messages (Broadcast), nous devons penser les systèmes Peer-to-Peer comme des infrastructures où l'information sur la localisation des ressources est distribuée de telle façon que chaque pair du système ne doive maintenir qu'une faible quantité d'information de routage. La recherche d'une ressource s'exécute alors de façon incrémentale par le transfert de proche en proche de la requête correspondante en dirigeant progressivement celle-ci vers le ou les pairs les mieux "informés" pour y répondre.

Le recours à la distribution de l'information de routage entraîne l'incohérence ponctuelle du système lorsque son environnement change (lors des départs et arrivées de pairs dans le système). Les pairs d'un tel système doivent donc continuellement s'adapter aux modifications de leur environnement. Il s'avère dès lors également souhaitable qu'un nombre restreint de noeuds voient leur information de routage influencée par une telle modification de l'environnement.

3.2 Un modèle général de Distributed Hash Table

Diverses solutions ont été apportées dans le but de remédier aux différents inconvénients des systèmes P2P actuels. Les auteurs de Chord [4], CAN [5], Pastry [6] et Tapestry [7] font reposer l'intérêt de leurs systèmes sur la décentralisation, la "scalabilité" et la performance des requêtes de recherche en terme de nombre d'hôtes contactés et de trafic engendré. La complexité (en terme de nombre de sauts sur le réseau Peer-to-Peer) de résolution d'une requête est de l'ordre de $\log(N)$, N représentant le nombre de noeuds dans le système. Le volume d'information de routage maintenu par les pairs de tels systèmes est également de l'ordre de $\log(N)$.

Une analyse de ces différents modèles permet de constater qu'ils fonctionnent tous selon des principes similaires. Dans [8], les auteurs identifient déjà ces principes en classant ces mécanismes dans l'ensemble des implémentations du "Document routing model". Il semble cependant que la communauté P2P taxe ces nouveaux modèles du nom de DHT (Distributed Hash Table). Effectivement, il convient de constater que l'interface de ces systèmes offre des fonctionnalités similaires aux tables de hachage classiques.

Nous tentons donc d'identifier ces principes communs en proposant un modèle général de DHT. Les 5 points suivants décrivent un ensemble de mécanismes permettant à un groupe d'ordinateurs connectés via un réseau de rendre un service identique à celui d'une table de hachage; le contenu de celle-ci étant distribué parmis les constituants du groupe, appelés hôtes ou noeuds.

Identification des noeuds et des ressources

Chaque hôte se voit affecté un identifiant numérique calculé par exemple sur base d'une fonction de hachage appliquée à son adresse IP. A chaque document ou ressource partagée correspond également un identifiant numérique (sur base d'une fonction de hachage appliquée à son contenu ou à son nom). Les valeurs des identifiants de ressource et d'hôte se situent dans le même espace désigné par le symbole V.

Pour un état donné du système, on distingue un sous-ensemble de V, noté N, correspondant à l'ensemble des noeuds participant au système.

Répartition des responsabilités des pairs

Lorsqu'un noeud est présent dans le système, il se voit attribuer la responsabilité d'un certain nombre de ressources. Chaque implémentation du modèle doit définir la manière à laquelle les ressources sont affectées aux différents noeuds du système. Nous pouvons donc affirmer que chaque implémentation du modèle décrit une relation R_{key} injective et surjective, dépendante de l'état du système, qui associe un identifiant de noeud et un identifiant de ressource et est vérifiée lorsque le noeud est effectivement responsable de la ressource.

Cette relation étant injective et surjective, elle admet un inverse fonctionnel et partout défini r_{key} de signature $r_{key} : V \rightarrow N$

Organisation de l'information de routage

Pour des raisons de "scalabilité", tout noeud d'un système répondant au modèle ne doit maintenir qu'une vision partielle de la topologie du réseau P2P auquel il participe. Ainsi, chaque noeud connait un sous-ensemble des noeuds présents dans le système. Chaque implémentation du modèle doit donc définir une relation (R_{node}) qui associe un identifiant de noeud à un autre identifiant de noeud et est vérifiée lorsque le second fait partie de la vision partielle du premier de la topologie du réseau. Cette relation est partout définie, irréflexive et telle que le graphe (N, R_{node}) est connexe.

Résolution d'un "lookup"

Un hôte doit pouvoir effectuer une requête d'accès à un document ou à une ressource partagée dans le système. Pour ce faire, il doit connaitre la valeur de la clé correspondant à cette ressource[1]. Cette requête de recherche est appelée *lookup*. Le résultat d'un lookup dans le système, pour une clé k, est une référence vers le noeud responsable de clé k, en l'occurence X tel que $XR_{key}k$.

Pour résoudre un *lookup*, un hôte commence par chercher, parmis les hôtes qu'il connait, celui dont l'identifiant tend le plus à vérifier la relation R_{key} avec la clé correspondant à la ressource recherchée. Il transfère la requête à ce noeud, qui effectue la même opération. La recherche se propage ainsi de noeud en noeud et se termine lorsqu'elle parvient à l'hôte effectivement responsable de la clé[2]. Le

[1] Ceci pose d'ailleurs problème pour le développement d'applications désireuses de profiter des avantages de ce modèle.

[2] En pratique, la recherche peut se terminer lorsqu'un noeud peut décider de façon certaine de l'identité de ce responsable.

noeud émetteur de la requête est alors averti de l'identité du noeud responsable de la clé.

Plus formellement, le résultat souhaité pour un lookup peut être décrit au moyen d'une fonction de la forme $Lookup : N \times V \rightarrow N^+$ telle que $Lookup(n, k)$ désigne un chemin de longueur minimale de n à $r_{key}(k)$ dans le graphe (N, R_{node}).

Pour être complètement défini, le système doit également comprendre une fonction de résolution de "nexthop" σ de signature $N \times V \rightarrow N$ satisfaisant la propriété ci-dessous.

$$\forall n \in N, \forall k \in V,\ lookup(n, k) = (n_1, ..., n_p) \Rightarrow \forall i : 1 \leq i \leq p : n_i = \sigma(n_{i-1}, k)$$

Remarquons que l'efficacité d'un système peut notamment être évaluée en analysant la tendance des relations R_{key} et R_{node} à minimiser la longeur des chemins empruntés par les requêtes.

Gestion des départs et arrivées de noeuds

Lors des départs et arrivées de noeuds, le système s'adapte et réaffecte les responsabilités des noeuds pour se maintenir dans un état de cohérence. Pour entrer dans le système, un noeud doit simplement disposer d'un accès à un noeud déjà présent dans celui-ci[3].

3.3 Efficacité réelle et marketing

Identifiants aléatoires et proximité des hôtes L'utilisation de valeurs "aléatoires" pour les identifiants de noeuds introduit un inconvénient dont toutes les implémentations risquent de souffrir. En effet, cela implique que la relation "voisin dans le paysage du système P2P" n'a aucun lien avec une quelconque proximité dans la couche réseau sur lequel fonctionne le système. On ne peut donc pas rendre deux noeuds voisins dans le système pour profiter de leur proximité sur le réseau. On peut donc rencontrer des situations où, pour trouver une ressource qui se situe sur une machine proche, un noeud effectue une recherche qui effectuera un trajet important sur le réseau. Les "auteurs" de DHT semblent vouloir contrer cet inconvénient en ajoutant des mécanismes ou en modifiant la méthode d'affectation des identifiants numériques afin de rendre leur implémentation plus attentive à la topologie du réseau sur lequel elle repose. Nous verrons ainsi que les protocoles CAN et Pastry, même s'ils sont des implémentations de DHT, disposent de propriétés les rendant sensibles à la proximité des hôtes.

Localité des ressources Certains auteurs n'hésitent pas à qualifier leur DHT de solution extrêmement performante, en comparaison avec Gnutella. Ainsi, dans [6], on peut lire :

> "Pastry, along with Tapestry, Chord and Can, represent a second generation of peer-to-peer routing and location schemes that were inspired

[3] N'importe lequel.

> by the pioneering work of systems like FreeNet and Gnutella. Unlike that earlier work, they guarantee a definite answer to a query in a bounded number of network hops, while retaining the scalability of FreeNet and the self-organizing properties of both FreeNet and Gnutella. "

Il semble cependant que les buts avoués des DHT dans leur définitions actuelles ne correspondent pas aux services rendus par Gnutella. Gnutella offre un partage de fichiers entre une multitude de particuliers situés sur un réseau "hétérogène", alors que les applications promises des DHT relèvent souvent de la distribution de contenu dans un réseau d'entreprise. Nous allons tenter d'en expliquer la raison principale.

Un aspect critique des DHT réside dans l'imposition de la localité des ressources. Pour illustrer l'impact de cette contrainte, nous pouvons envisager une implémentation de DHT visant à rendre le service le plus populaire des applications P2P : le "file sharing".

De par sa nature, une telle application impose donc le contenu informationnel stocké localement par chaque pair. En effet, chaque ressource (en l'occurence chaque fichier) est référencée dans le système grâce à une clé. Chaque pair du système est également identifié par une valeur numérique obtenue par une opération mathématique. Ces deux aspects d'identification numérique combinés avec la définition par le système de la relation R_{key} impose donc une localisation arbitraire des ressources proposées par le système. L'évolution des volumes de stockage et de la bande passante disponibles chez les particuliers peut nous laisser imaginer que l'utilisateur futur des systèmes de partage de fichiers pourrait être prêt à céder une partie de ses ressources à la "communauté", permettant ainsi le fonctionnement de "notre" application.

Cette application imaginaire nous amène cependant à nous poser certaines questions.

- Nous pouvons arguer sans crainte que les ressources matérielles des pairs sont potentiellement très diversifiées. Comment réagir alors si la relation R_{key} impose la responsabilité de fichiers de taille conséquente à un utilisateur doté de ressources limitées ou d'une bande passante faible ?
- Le départ d'un hôte du système doit être comblé par le transfert des ressources dont il est responsable vers un hôte dont l'identité dépend de la relation R_{key}. Peut-on sérieusement envisager ce transfert dans un contexte d'utilisation par le particulier ? Retenons également que le départ d'un hôte n'est pas forcément volontaire, ceci remettant évidement en question la possibilité réelle d'effectuer ce transfert.
- Une requête de recherche dans le système correspond à un lookup paramétré par la clé identifiante de la ressource souhaitée. Cette aspect est-il compatible avec la méthode de recherche la plus utilisée (parce qu'elle est la plus adaptée) par les utilisateurs, à savoir la spécification de sous-parties du nom de la ressource ? Quels moyens peuvent être mis en place pour fournir à l'utilisateur la connaissance de la clé identifiante de le ressource qu'il recherche ?

– Enfin, comment répondre à la question de la responsabilité juridique des données stockées par chaque utilisateur dans un tel contexte ? Le monde du Peer-to-Peer n'a sans doute pas besoin de nouveaux conflits avec la justice.

Ces quelques remarques nous font penser que les services proposés par les DHT sont de nature différente des services rendu par Gnutella ou Napster. Posons nous alors la question de savoir quel crédit accorder à une comparaison des performances des DHT (dans leur définition actuelle) avec Gnutella.

Nous pouvons proposer quelques pistes pour lever certaines contraintes liées à la nature des DHT et donc permettre de les adapter à la recherche de documents telle qu'elle est considérée par les utilisateurs. Nous pouvons redéfinir le sens attaché à la relation R_{key} de telle façon qu'un pair ne soit pas responsable du maintien des ressources correspondant aux clés avec lesquelles la relation R_{key} est vérifiée mais bien responsable du maintien de **l'information sur la localisation** de ces ressources. Ainsi, lorsqu'un hôte rentre dans le système, il s'enregistre auprès du responsable de la clé correspondant à chaque ressource qu'il détient localement. Lorsqu'une requête de résolution d'une clé parvient au responsable effectif de cette clé, celui-ci répond en envoyant la liste des hôtes qui se sont enregistrés pour cette clé. Une solution similaire a été proposée pour Kademlia, une DHT décrite dans [9].

4 Implémentations du modèle

Dans cette section, nous présentons deux implémentations du modèle de DHT. Pour ce faire, nous décrivons la manière à laquelle ces implémentations respectent les principes décrits dans notre modèle général de DHT.

4.1 CAN

CAN (Content Adressable Network) est décrit dans [5]. Sa principale caractéristique est la considération de l'espace des identifiants comme une zone géographique à plusieurs dimensions dont chaque noeud est responsable d'un sous-espace.

Identification des noeuds et des ressources

La valeur de hachage de l'adresse IP d'un noeud est interprétée comme les coordonnées d'un point dans un espace cartésien E de dimension dim fixée. A chaque ressource correspond une clé k calculée également au moyen d'une fonction de hachage et interpretée de la même façon comme les coordonnées d'un point de E. L'espace V du modèle général correspond donc ici à E.

Répartition des responsabilités des pairs

Chaque noeud entrant dans le système se voit affecter une "zone géographique", correspondant à un sous ensemble des points de l'espace E. L'identifiant

du noeud représente un point de cette zone géographique. A tout moment, l'entièreté de l'espace E est affecté aux noeuds du système et l'intersection de deux zones géographiques détenues par deux noeuds différents est vide.

Cette zone est $dim-rectangulaire$. La relation de responsabilité R_{key} associe un noeud N à une clé k et est vérifiée lorsque k appartient à la zone géographique affectée à N.

Organisation de l'information de routage

Deux noeuds vérifient la relation de "connaissance" R_{node} s'ils sont responsables de zones géographiques adjacentes. Dans ce modèle, on a donc :

$$\text{Pour tout couple de noeud } (A, B) \; : \; A \, R_{node} \, B \; \Rightarrow \; B \, R_{node} \, A$$

Résolution d'un "lookup"

Lorsqu'un "lookup" pour une clé k est traité par un hôte, celui-ci transmet la requête vers le noeud dont les coordonnées géographiques sont les plus proches des coordonnées que représente k. De cette façon, la requête est bien transférée vers le noeud tendant le plus à vérifier la relation R_{key} avec k.

La fonction de résolution de nexthop σ correspond donc à la découverte, parmis l'ensemble des identifiants des hôtes responsables des zones adjacentes, de l'identifiant tel que la distance euclidienne entre celui-ci et k est minimale.

Le noeud responsable est trouvé lorsque la requête aboutit à un noeud dont la responsabilité couvre le point défini dans la requête. Ce noeud vérifie donc la relation R_{key} avec la clé recherchée.

Gestion des départs et arrivées de noeuds

Lors d'une arrivée d'un nouveau noeud dans le système, le contact d'entrée du noeud effectue un *lookup* avec, comme valeur de clé, les coordonnées du noeud entrant calculées par hachage. Le noeud responsable apprend de cette façon l'arrivée d'un pair et divise sa zone géographique en deux zones. Cette division se fait de manière à minimiser la somme en valeur absolue des différences entre les dimensions de chaque zone. Le but de cette méthode est d'éviter qu'un noeud puisse être responsable de zones très étendues sur une seule dimension et minime sur une autre, afin de réduire le nombre de zones adjacentes. A deux dimensions, ce mécanisme tend à rendre les zones les plus carées possibles. Le noeud restreint sa responsabilité à une de ces deux parties et donne la responsabilité de l'autre moitié au nouveau noeud. Les responsables des zones adjacentes à ces deux moitiés sont prévenus de la modification afin que ceux-ci mettent à jour leur table de routage.

Lors d'un départ, un mécanisme distribué permet aux différents voisins de se répartir la zone libre. Ceci se déroule de manière à ce que les noeuds responsables des plus petites zones en prennent une part plus importante.

L'avantage majeur de cette solution se situe dans la "paramétrisabilité" de la localisation des noeuds dans l'espace. En effet, la fonction de hachage peut-être utilisée pour obtenir les valeurs des coordonées d'un point de E restreint à un nombre fixé de dimension i. Ainsi, les $dim - i$ valeurs restantes peuvent être déterminées sur base d'un ensemble de caractéristiques du noeud.

On peut, par exemple, utiliser la valeur de hachage pour calculer les $dim - 1$ premières coordonnées du point et l'adresse IP du noeud pour calculer la dernière. Ce calcul se faisant de manière à rapprocher, sur cette dimension, des machines proches sur le réseau. La proximité de deux machines sur le réseau aurait donc une influence sur leurs proximité, dans l'espace cartésien, des zones dont ils sont responsables. L'intérêt de cette méthode réside dans le fait que les noeuds communiquent essentiellement avec leurs voisins.

4.2 Chord

Chord est notamment décrit dans [4]. Il répond au modèle de la manière suivante :

Identification des noeuds et des ressources

Chaque noeud et chaque ressource reçoit un identifiant numérique obtenu par une fonction de hachage (SHA-1) décrite dans [10]. Cette fonction s'applique, pour un noeud, à son addresse IP. Pour une ressource, elle s'applique à son nom ou à son contenu. SHA-1 est supposée rendre minime la probabilité que deux valeurs de hachage obtenues à partir d'adresse IP différentes soient identiques. Elle est supposée répartir ces valeurs de manière uniforme dans un espace fixé. Pour Chord, cet espace est un ensemble de valeurs numériques dont la taille est égale au nombre maximal de noeuds présents dans le système.

A un instant donné, un noeud N vérifie la relation R_{key} avec une ressource Res si :

$$Res \in]Predecesseur(N), N]^4$$

On définit le successeur d'un noeud d'identifiant N comme le noeud dont l'identifiant numérique $Succ$ suit N dans l'espace des identifiants. La notion de prédécesseur utilisée plus haut peut être définie grâce à cette notion de successeur. En effet, le prédécesseur d'un noeud N est le noeud $Pred$ dont le successeur est N. Il faut remarquer que la relation de succession s'exprime en considérant l'aspect "cyclique" de l'espace des identifiants. Ainsi, pour $T = 64$, l'intervalle $[60, 3]$ existe et est constitué par les noeuds $\{60, 61, 62, 63, 0, 1, 2, 3\}$ Ainsi, le successeur du noeud 63 est 1 à condition qu'aucun noeud présent dans le système ne porte l'identifiant 0.

Dans la figure 2, nous illustrons l'affectation des responsabilités de ressources notées Kx, dont la valeur de clé est x. On remarque l'aspect cyclique de l'espace des identifiants en observant que le noeud $N1$ d'identifiant 1 est le responsable de la ressource d'identifiant 64. En effet, cette valeur de clé est comprise entre 62 et 1, valeurs des identifiants des noeuds les plus proches, de part et d'autre de 1.

[4] La relation d'appartenance et la notion d'intervalle porte sur les identifiants numériques de Res et de N.

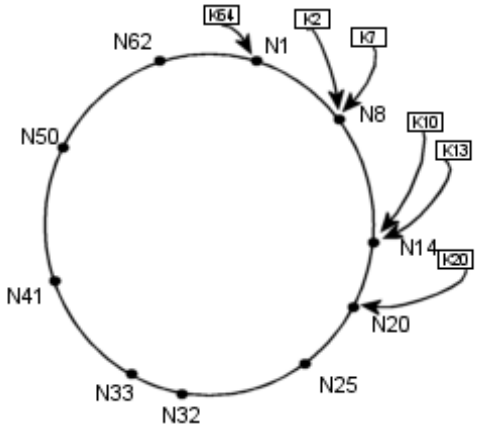

Fig. 2. Les noeuds sont responsables des ressources dont la valeur de clé est comprise entre leur identifiant et l'identifiant de leur prédécesseur.

Organisation de l'information de routage

Un noeud d'identifiant N connait les pairs responsables des clés faisant partie de l'ensemble suivant :

$$\{X = (N + 2^{i-1})\ mod\ T\,, 1 \leq i \leq log_2(T)\}$$

T étant la taille de l'espace des valeurs d'identifiant de noeud.

On a donc la propriété suivante pour Chord :

$$A\, R_{node}\, B \Leftrightarrow \exists\, i \,|\, 1 \leq i \leq log_2(T) : B\, R_{key}\, ((A + 2^{i-1})\ mod\ T)$$

Exemple : si le nombre maximum de noeuds présents dans le système est 64 alors les connaissances du noeud dont l'identifiant est 1 sont les noeuds responsables des clés de l'ensemble $\{2, 3, 5, 9, 17, 33\}$. Ces clés sont obtenues en additionnant la valeur de son identifiant (1) avec le nombre (2^{i-1}) pour la $i^{ème}$ entrée de sa table de routage. Dans la figure 3, ces responsables sont :

Noeud	*Ecart*	*Clé*	*Responsable*
$N1$	1	2	$N8$
$N1$	2	3	$N8$
$N1$	4	5	$N8$
$N1$	8	9	$N14$
$N1$	16	17	$N20$
$N1$	32	33	$N33$

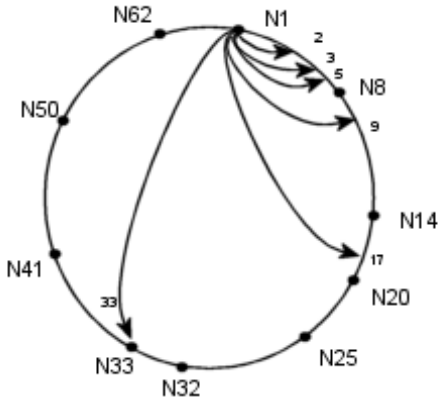

Fig. 3. Ensemble des connaissances de $N1$.

Résolution d'un "lookup"

Lors d'un "lookup", un noeud transmet la requête au noeud dont la valeur de l'identifiant précède la clé recherchée et s'en approche le plus. Lorsqu'un noeud se trouve mieux placé que toutes ses connaissances, il désigne son successeur comme étant le responsable effectif de la clé. En effet, si il est lui-même le plus proche noeud qui précède la clé, alors son successeur se situe après cette clé. On peut en déduire que la clé a une valeur comprise entre lui même et son successeur. Ainsi, le successeur est bien le responsable de la clé.

On notera *Closest_preceding_node* l'opération de recherche du noeud qui précède la clé et s'en approche le plus. Notons que cette opération correspond à la fonction de résolution de nexthop σ du modèle général.

Un exemple de lookup est donné à la figure 4. Le noeud $N62$ effectue un lookup pour la clé 31. Conformément à la description de l'organisation de la table de routage d'un noeud, celle-ci contient, pour le noeud $N62$, l'ensemble de noeuds $\{N1, N8, N14, N34\}$. Le noeud dont la valeur d'identifiant précède la clé 31 en s'en approchant le plus est $N14$. $N62$ transmet donc sa requête à $N14$. La table de routage de $N14$ contient les noeuds $\{N20, N25, N32, N50\}$. Le noeud le plus proche qui précède 31 est $N25$. C'est donc à $N25$ que $N14$ retransmet la requête. $N25$ remarque qu'aucun noeud de sa table de routage n'est plus proche de 31 que lui même. Autrement dit, la clé 31 est comprise entre la valeur de son identifiant (25) et l'identifiant de son successeur (32). Par définition, $N32$ est donc le responsable de la clé 31. $N25$ peut alors répondre à l'émetteur initial de la requête ($N62$) en lui fournissant les références vers le noeud $N32$.

Gestion des départs et arrivées de noeuds

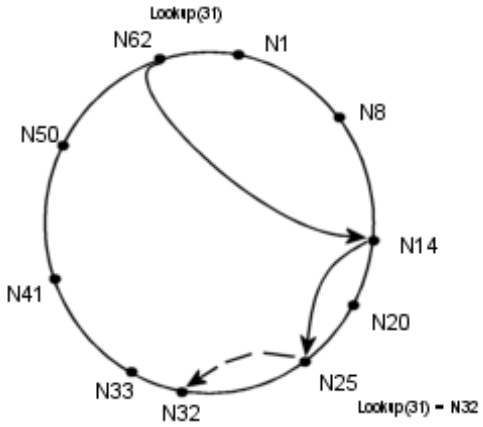

Fig. 4. Exécution théorique d'un "lookup" sur la clé 31 à partir du noeud $N62$.

Lorsqu'un noeud N rentre dans le système, le contact d'entrée du noeud effectue un *lookup* avec la valeur de l'identifiant du noeud entrant comme valeur de clé. Le résultat correspond au futur successeur du noeud entrant. N contacte alors son successeur pour l'avertir de son arrivée. De cette façon, le successeur peut mettre à jour son prédécesseur, considérant le noeud entrant comme son nouveau prédécesseur.

Pour conserver la cohérence du système, chaque noeud doit maintenir une vue correcte de son successeur. Pour cela, chaque hote demande périodiquement le prédecesseur de son successeur présumé. S'il s'avère que ce prédécesseur se situe avant le successeur présumé, alors le noeud vient de découvrir un meilleur successeur. Il met à jour sa vue vers celui-ci. Chaque noeud émet périodiquement un message vers son successeur afin que celui-ci puisse mettre à jour sa vue vers son prédécesseur.

Dans l'article [4], les auteurs proposent le pseudo-code suivant pour la stabilisation :

```
n.stabilize()
   x=successor.get_predecessor()
   if x between(n,successor)
   then
      successor=x
   end
   successor.notify(n)

n.notify(n2)
   if (predecessor is nil or n2 between(predecessor,n))
   then
```

```
    predecessor = n2
end
```

Lorsqu'un noeud quitte le système, il laisse un de ses pairs sans successeur valide. Pour que le système ne s'effondre pas au rythme des départs de noeud, chaque noeud maintient une liste contenant ses successeurs consécutifs. La taille de cette liste doit être établie en fonction du nombre maximal de noeuds dans le système et de la probabilité de départ simultané de noeuds adjacents. Lorsqu'un noeud se rend compte du départ de son successeur, il parcourt la liste, en commençant par le noeud le plus proche, jusqu'à ce qu'il trouve un noeud valide qu'il considère comme son nouveau successeur.

Un noeud crée sa liste de successeurs de la façon décrite ci-dessous.

Initialement, il ne connait que son successeur, la liste de ses successeurs ne contient donc qu'une seule entrée. Lorsqu'il reçoit un message de notification de son prédécesseur, il répond en lui envoyant sa liste de successeurs. Lorsqu'un noeud reçoit une liste de successeurs, il lui ajoute son propre successeur, en enlevant le successeur le plus lointain si la liste a déjà atteint sa taille maximale.

Remarquons que cette liste est incohérente si sa taille maximale est supérieure au nombre de noeud effectivement présents dans le système. Chaque noeud se voit donc contraint d'élaguer cette liste afin qu'elle ne contienne pas de cycle.

5 Conclusion

Après avoir énoncé les faiblesses potentielles des modèles P2P de "première génération", nous avons décrit le principe de table de hachage distribuée. Sur base du modèle général de DHT, nous avons identifié certains aspects critiques pour l'implémentation d'applications basées sur ce principe. La manière à laquelle les descriptions des modèles existants ont été réalisées (sur base d'un modèle général) met en évidence la similarité de ce genre de solutions. Il peut donc s'avérer intéressant de traduire les améliorations possibles des protocoles existants dans les termes d'un modèle général de DHT. Ainsi, l'ensemble des auteurs de DHT pourront transposer ces améliorations à leur modèle particulier.

Nous avons étudié les responsabilités de chaque pair d'un système P2P basé sur une DHT. Posons nous alors la question de savoir comment ces modèles réagissent quand, pour une raison quelconque, certains pairs du système ne respectent pas ces responsabilités. L'idéal consisterait en l'intégration, dans le modèle général lui-même, de solutions permettant de gérer les comportements potentiellement frauduleux des participants. La complexité de la problématique de gestion des comportements frauduleux au sein d'un système distribué est telle que, lorsque une solution devient robuste (grâce à l'ajout de fonctionnalités) contre un type de comportement frauduleux, l'adaptation de ces solutions dans les termes du modèle général peut, dans les cas où cette adaptation est réalisable, profiter à tous les types de DHT existantes.

Remarquons, pour terminer, que l'apport indéniable des DHT dans l'amélioration des performances des systèmes P2P a été réalisé au prix d'un certain

nombre de contraintes que les "développeurs" devront évaluer afin de déterminer si, dans leur cas d'utilisation précis (file sharing, file mirroring, publish/subscribe, etc.), elles n'en rendent pas l'utilisation trop complexe voire impossible.

Références

1. Jordan Ritter. Why gnutella can't scale. no, really.
2. M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network : Properties of large-scale peer-to-peer systems and implications for system design, 2002.
3. http ://opennap.sourceforge.net/napster.txt.
4. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
5. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.
6. Antony Rowstron and Peter Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218 :329– ??, 2001.
7. B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry : An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.
8. Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing.
9. P. Maymounkov and D. Mazieres. Kademlia : A peer-to-peer information system based on the xor metric, 2002.
10. U.S. Department of Commerce/N.I.S.T. National Technical Information Service. Secure hash standard. In *FIPS 180-1*, April 1995.
11. P. Druschel and A. Rowstron. PAST : A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, pages 75–80, Schloss Elmau, Germany, May 2001.
12. Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE : The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.
13. Antony I. T. Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.
14. P. Krishna Gummadi, Stefan Saroiu, and Steven Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems.
15. The gnutella protocol specification v0.4.