THÈSE

pour obtenir le grade de

Docteur

Discipline : **Informatique**

par

Frédéric Dang Ngoc

sur le sujet

Moteur de recherche sémantique par réseau adaptatif de pairs

PAGE NON FINALISÉE

- résumé à retravailler
- traduction en anglais

Résumé

La recherche d'information est devenue plus que jamais un problème d'actualité. Le problème ne se situe pas dans la pénurie d'information mais au contraire sur sa surabondance. Premièrement, référencer toutes les informations disponibles et les analyser entièrement s'avèrent impossible dans un web sans cesse changeant et dont la croissance exponentielle est devenue aujourd'hui impossible à suivre. Deuxièmement, trouver parmi les milliards d'informations analysées, celles qui sont les plus pertinentes à chaque utilisateur et à chacune de leurs questions est un problème difficile qui n'a été que partiellement résolu.

Nous décrirons dans cette thèse un moteur de recherche que nous avons conçu où la publication, l'indexation et la recherche de données sont décentralisées sur les machines des utilisateurs. Ensuite, nous proposerons un modèle des utilisateurs en nous appuyant sur des études effectuées sur les utilisateurs de systèmes de recherche, sur des études bibliométriques et sémantiques. Puis nous définirons un ensemble de mesures pour évaluer l'efficacité d'un système de recherche à retourner des réponses personnalisées à ses utilisateurs. Les résultats obtenus par simulation de notre système de recherche montrent que le système s'adapte aux utilisateurs pour leur fournir des réponses qui leur sont personnalisées. Enfin, nous avons implémenté un prototype de notre système afin de tester sa faisabilité et les différentes fonctionnalités qui seront implémentées dans la version finale du système.

Abstract

Nowadays, information retrieval is still a hard problem. The issue does not lie on information scarcity but rather on its overabundance. First, referencing all available datas and analyse them completely seem impossible in a web still in mutation and growing at an exponential pace. Second, find in the billions of analysed datas, the most relevant ones for each user and for each of their queries is definetely a hard problem which has been partially solved.

We will describe in this thesis the search engine that we have conceived. In this system, publications, indexations and searches are decentralized on user's computers.

Remerciements

Table des matières

Chapitre 1

Introduction

La recherche et l'acquisition du savoir ont toujours été un moteur dans l'évolution de l'humanité. Les méthodes de transmission ont évolué au cours des siècles, passant de l'oral à l'écriture, puis au numérique. Le numérique a permis d'accélérer considérablement le transfert d'information en permettant la réplication d'une donnée en un instant. Mais c'est surtout grâce au développement du réseau Internet qu'une information peut maintenant être propagée à des milliers voire des milliards de personnes en quelques secondes. Les méthodes de recherche, qui traditionnellement se faisaient en explorant les informations classées par catégorie, se font maintenant en tapant quelques mots clés dans un moteur de recherche.

Cependant, avec le développement de nouveaux médias de communication, l'information disponible ne cesse de croître et augmente de manière exponentielle. Tout d'abord avec le web et les groupes de discussion qui ont permis à tout anonyme d'être lu par des millions de lecteurs. Puis maintenant avec les *blogs* qui permettent à tout un chacun de s'exprimer *via* un journal qu'il met régulièrement à jour.

Avec des milliards de documents disponibles, il est très difficile de concevoir un système de recherche qui puisse répondre à des centaines de millions de requêtes quotidiennes. Les moteurs de recherche actuels comme Google, Yahoo et MSN Search sont confrontés à plusieurs problèmes :

- la masse d'informations à analyser est dynamique et gigantesque. L'analyser entièrement et de manière régulière avec une périodicité courte est difficilement réalisable. Par conséquent, les résultats sont incomplets et ne sont pas à jour. Les moteurs actuels fournissent des résultats qui datent d'un mois en moyenne.
- chaque requête peut avoir potentiellement des milliers voire des millions de réponses possibles ; sélectionner parmi ces réponses celles qui satisferont l'utilisateur demande des techniques de classement ou de classification très élaborées ; les réponses fournies actuellement par les moteurs de recherche ont une pertinence moyenne qui s'explique par leurs techniques de classement impersonnel qui ne cherche pas à s'adapter à la personnalité de chaque utilisateur.
- le fonctionnement de ces moteurs de recherche est tenu secret. Les utilisateurs sont donc impuissants face à la politique de confidentialité de ces moteurs de recherche et des biais qu'ils introduisent pour le classement des résultats.
- le système demande une infrastructure lourde et coûteuse.

Avec l'augmentation exponentielle des documents disponibles sur le web, ces problèmes vont devenir de plus en plus aigus et de nouvelles techniques seront nécessaires pour y faire face.

Dans cette thèse, nous tenterons de résoudre ces problèmes en proposant un système de recherche adaptatif par réseau de pairs.

1.1 Contexte

Un *espace d'information* est un ensemble de données, ces données peuvent être textuelles, sonores ou visuelles et sont souvent décrites par des mots. L'espace d'information peut être dynamique, c'est-à-dire que des données apparaissent, disparaissent ou se modifient continuellement.

On appelle *système de recherche* ou moteur de recherche, une application permettant aux utilisateurs de rechercher des données dans un espace d'information. Typiquement, les utilisateurs tapent une requête de recherche composée d'un ensemble de mots clés et reçoivent en réponse une liste des descriptions de donnée (titre, extrait, URL).

Les moteurs de recherche traditionnels se basent sur une architecture centralisée. Ces serveurs parcourent, indexent le web et répondent aux requêtes des utilisateurs.

Une alternative possible aux systèmes centralisés est le système pair à pair (peer-to-peer en anglais, ou P2P). Un *système de pair à pair* est un système où chaque machine est à la fois client - elle peut effectuer des requêtes - et serveur - elle répond aux requêtes. Le système n'est composé que des machines des utilisateurs et n'implique pas de coûteux serveurs dédiés. Une caractéristique commune des systèmes pair à pair est leur autonomie et leur résilience. Les machines s'interconnectent automatiquement entre elles et l'arrivée de participants ou leur départ ne dégrade pas significativement le fonctionnement du système.

1.2 Axe de recherche

Un *système d'échange de fichiers pair à pair* est un système où les utilisateurs peuvent rechercher et s'échanger des fichiers. Chaque utilisateur gère un moteur de recherche local sur son espace d'information correspondant à ses fichiers partagés. La recherche s'effectue en interrogeant le moteur de recherche de plusieurs utilisateurs. Ces systèmes ont plusieurs avantages. Premièrement, les réponses sont à jour puisque la recherche s'effectue localement sur le contenu des machines interrogées. Deuxièmement, le système ne demande pas d'infrastructures coûteuses. Cependant cette architecture n'est pas exempte de défauts. Premièrement, la recherche n'est pas exhaustive car elle est limitée aux contenus des quelques machines interrogées. Seules les fichiers populaires et largement répliqués peuvent donc être trouvés. Deuxièmement, la recherche est coûteuse en bande passante et le temps pour trouver une réponse peut être élevé. Troisièmement ce type de système n'est pas bien adapté aux recherches textuelles avec classement des résultats.

L'idée d'utiliser une architecture pair à pair pour implémenter un système de recherche a commencé en 1999 avec notamment Napster et Gnutella, mais elle n'a été que peu explorée pour la recherche textuelle avec classement par pertinence. Actuellement, seuls les systèmes pair à pair d'échange de fichiers ont été déployés et utilisés.

Un système de recherche doit pouvoir assurer les fonctionnalités suivantes :

- Parcours exhaustif de l'espace d'information.
- Indexation à jour.
- Réponses exhaustives aux requêtes des utilisateurs.
- Classement par pertinence des résultats.

Ce sont ces fonctions que nous allons essayer de décentraliser dans notre système de recherche.

1.3 Plan de la thèse

Nous commençons par étudier les solutions existantes aussi bien centralisées que décentralisées (chapitre 2).

Puis nous définissons le fonctionnement et les algorithmes de notre système de recherche collaboratif par réseau de pairs permettant de fournir à ses utilisateurs des réponses qui leur sont *personnalisées* (chapitre 3).

Pour tester notre système, nous choisissons d'en implémenter une version et d'y simuler des utilisateurs. Pour cela, nous étudions les modèles des utilisateurs (chapitre 4), notamment des systèmes d'échange de fichiers. Puis, nous poussons plus loin ces modèles afin de pouvoir simuler de manière plus complète des utilisateurs de moteur de recherche textuel et nous proposons des mesures pour évaluer l'efficacité d'un système de recherche à fournir des réponses pertinentes et personnalisées à ses utilisateurs.

Ensuite nous implémentons un programme permettant de simuler des milliers d'utilisateurs effectuant des milliers de requêtes de recherche et nous mesurons l'efficacité du système (chapitre 5).

En parallèle, nous implémentons un prototype plus complet avec plus de fonctionnalités qui nous servira de base pour la conception du système final (chapitre 6).

Enfin, nous réfléchissons aux améliorations futures possibles du système aussi bien sur le modèle des utilisateurs que sur la simulation (chapitre 7).

Chapitre 2

Les systèmes de recherche

Grâce à l'évolution des moyens de communication et à la démocratisation de l'informatique, publier des informations et y accéder sont devenus pratiques courantes. Avec l'apparition du web, toute personne peut créer un site web et être potentiellement lu par des millions de personnes. Publier est devenu encore plus facile aujourd'hui notamment grâce aux outils de *blogs* qui permettent de poster une information en quelques clics de souris. Cependant, avec l'explosion des informations publiées et disponibles sur le web, trouver une information disséminée dans les milliards de pages web est devenu une tâche de plus en plus ardue.

Heureusement, cette tâche est grandement simplifiée par l'apparition dès 1993 des moteurs de recherche qui permettent à un utilisateur de trouver en quelques secondes une information présente dans des pages du web à partir des mots clés. Mais les pages indexées par les moteurs de recherche ne sont pas à jour puisque le délai moyen d'indexation est de 30 jours, ce qui n'est pas acceptable pour rechercher des *blogs* ou des pages de *news*.

A partir de l'an 2000, on voit apparaître des systèmes d'échange de fichiers qui n'ont pas ce défaut : ce sont les systèmes pair à pair. Ils permettent à des millions d'utilisateurs de rechercher les fichiers présents chez les autres utilisateurs et de se les échanger entre eux. Grâce à la décentralisation de la recherche et de l'indexation, ces systèmes offrent des résultats de recherche à jour à leurs utilisateurs.

Dans ce chapitre, nous étudierons différents systèmes de recherche. Nous commencerons par décrire les moteurs de recherche centralisés en exposant leurs qualités et leurs défauts. Ensuite, nous aborderons les systèmes de recherche décentralisés ; nous décomposerons cette étude en distinguant les systèmes non structurés et les systèmes structurés. Nous détaillerons pour ces deux types de systèmes leurs architectures, leurs fonctionnements ainsi que leurs points forts et leurs points faibles.

2.1 Moteur de recherche centralisé

2.1.1 Terminologies

Le web Le *world wide web* (web ou *www* en abrégé) désigne le réseau des documents hypertextes interconnectés par des liens hypertextes et accessibles *via* une URL grâce au protocole HTTP (Hypertext Transfer Protocol) de transfert de fichiers.

Une *URL* (Uniform Resource Locator) est une adresse web normalisée d'une ressource (documents hypertextes, images, vidéos, ...) se trouvant sur l'Internet (ou ailleurs). Un *lien hypertexte* (ou *hyperlien*) dans un document est une référence décrite par une URL permettant de passer automatiquement de la page consultée vers la page liée. Ces liens hypertextes

sont généralement rattachés à des mots, phrases ou images présents dans les documents hypertextes.

Un *document hypertexte* ou *page web* est écrit en langage HTML (HyperText Markup Language). Ce langage décrit le contenu et la représentation visuelle du document (police, couleur, fond, ...), les images et vidéos incluses dans le document et ses liens hypertextes.

Un *utilisateur* (ou *surfeur*) accède aux ressources du web en utilisant un navigateur web et « surfe » de pages en pages en suivant les liens hypertextes.

Un *site web* désigne un ensemble de pages web généralement accessibles par une URL racine commune et reliées entre elles par des liens hypertextes. Les pages du site web se trouvent souvent sur le même serveur physique.

Moteur de recherche Un *moteur de recherche* est un programme permettant de rechercher une donnée dans un espace d'information distribué sur une ou plusieurs machines. Ces données peuvent être des textes, des images, des sons ou encore des vidéos. Elles se trouvent généralement sur un serveur web connecté à l'Internet ou à l'intranet d'une entreprise. Le moteur de recherche permet à un utilisateur de rechercher des données suivant certains critères - généralement sur la présence de mots dans le contenu textuel de la donnée - et de récupérer la liste des données satisfaisant ces critères. Le terme moteur de recherche est souvent employé dans le contexte du web et qualifie un programme permettant de rechercher des pages du web.

2.1.2 Introduction

Depuis 1990, l'année où Tim Berner Lee a écrit la première page web, le nombre de pages web[1] n'a cessé de croître, allant de 20 millions de pages indexées en 1995 à 140 millions en 1999 pour atteindre aujourd'hui plus de 8 milliards.

Si au début de l'ère du web, chercher une page consistait principalement à suivre des liens, aujourd'hui elle se fait de plus en plus en tapant sa requête dans un moteur de recherche. Beaucoup de sites, particulièrement les sites de vente en ligne, dépendent de ces moteurs de recherche pour pouvoir attirer de nouveaux visiteurs. Ainsi, d'après les mesures effectuées par eStat[2] en janvier 2005, 40% des visites sur les sites web sont dues à des moteurs de recherche, 32% proviennent des liens venant d'autres sites et 27% par un accès direct en tapant l'URL du site dans un navigateur, en utilisant un marque-page ou en cliquant sur un lien dans un e-mail.

Les moteurs de recherche sont devenus un outil incontournable du web et sont aussi bien utilisés pour le travail que pour les loisirs. Leur efficacité croissante attire de plus en plus d'utilisateurs qui effectuent quotidiennement plus de 550 millions de requêtes de recherche[3]. Plusieurs acteurs, dont les principaux sont Google, Yahoo! et MSN Search, se partagent ce marché qui pourrait représenter 7 milliards de dollars en revenus publicitaires en 2007.

Nous allons décrire dans cette section, l'architecture et le fonctionnement d'un moteur de recherche ainsi que les algorithmes utilisés pour le classement des pages.

---

[1] Search Engine Watch. *Search Engine Size*. `http://searchenginewatch.com/reports/article.php/2156481`. 28 janvier 2005.

[2] Médiamétrie - eStat : mesure et analyse de la fréquentation des sites web. `http://www.estat.com/parution/parution_pano.html`.

[3] Technology Review.com. *Search Beyond Google*. `http://www.technologyreview.com/articles/04/03/roush0304.asp`. Mars 2004.

2.1.3 Architecture et fonctionnement

Un moteur de recherche est composé de trois parties qui sont : le crawler, l'indexeur et le moteur de requêtes.

Crawler Le *crawler* est un programme qui récupère les pages du web de manière automatisée en vue de leur indexation. Le crawler commence à partir d'une liste d'URLs à visiter. En visitant ces URLs, le crawler analyse la page et en extrait les liens hypertextes qu'il rajoute à la liste des URLs à visiter. Le processus se termine quand un certain nombre de liens ont été suivis.

Indexeur Les pages récupérées par le crawler sont ensuite traitées par l'*indexeur* qui analyse le texte visible de la page (texte lu dans un navigateur) et attribue un poids plus ou moins important aux mots suivant leurs positions dans le document, leurs fréquences et leurs représentations visuelles (taille de la police, en gras ou non). Ces informations ainsi que les méta données des documents (mots clés, date de modification, URL, ...) sont stockées dans des index. Afin de pouvoir répondre rapidement aux requêtes de recherche, le système gère un index inversé qui associe à chaque mot l'ensemble des documents qui le contiennent.

Moteur de requêtes Le *moteur de requêtes* répond aux requêtes des utilisateurs en leur renvoyant une liste ordonnée de descriptions de documents (titre, extrait de texte, lien vers la page, ...) qui est affichée dans le navigateur de l'utilisateur. L'utilisateur n'a plus qu'à cliquer sur un lien pour accéder à la page.

2.1.4 Classement des pages

Une requête de recherche peut avoir des milliers voire des millions de résultats si les mots employés dans la requête sont des termes très usités. Le classement des résultats se révèle donc indispensable pour afficher en première position les résultats jugés les plus pertinents. Les premières techniques de classement de pages se basaient principalement sur des critères locaux à la page, comme la fréquence des mots dans la page, et donnaient des résultats peu adéquats. Maintenant, elles se font sur d'autres critères qui utilisent les relations entre pages et permettent d'obtenir des résultats plus pertinents. Le classement des pages ne se base pas sur un seul critère, Google par exemple en utilise 150 dont son célèbre critère PageRank, et des critères se basant sur les fréquences de mots.

Les algorithmes de classement sont souvent tenus secrets et mis à jour régulièrement. En effet, leur connaissance permettrait à des webmestres peu scrupuleux d'exploiter les failles de ces algorithmes pour améliorer artificiellement le classement de leur site et par conséquent rendre le classement inefficace.

Dans cette sous-section, nous étudierons trois critères couramment employés pour le classement des résultats. Nous présenterons un critère permettant de calculer la similarité entre un document et une requête qui se base sur les fréquences de mots, puis nous étudierons deux critères se basant sur les liens hypertextes : PageRank et HITS.

2.1.4.1 TFIDF et VSM

TFIDF *TFIDF* (Term Frequency x Inverse Document Frequency) est une technique statistique pour évaluer le poids d'un mot dans un document. Cette technique fait l'hypothèse qu'un mot est important dans un document s'il y apparaît fréquemment et que peu de documents le contiennent. Il existe plusieurs formules de TFIDF.

Une formule classique pour calculer le poids $w_{m,d}$ d'un mot m dans un document d est :

$$w_{m,d} = tf_{m,d}.log(\frac{N}{df_m})$$

avec $tf_{m,d}$ le nombre d'occurrences du mot m dans le document d, df_m le nombre de documents contenant le mot m et N le nombre de documents.

Cette technique est souvent utilisée pour extraire les mots pertinents d'un document qui sont ceux ayant le poids le plus important.

Modèle d'espace vectoriel (VSM : Vector Space Model) Le *modèle d'espace vectoriel* représente les documents sous la forme d'un vecteur à n dimensions où n est la taille de l'ensemble E des mots utilisés. La valeur de la k^{ieme} coordonnée du vecteur correspond au poids du k^{ieme} mot de E dans le document. Une requête est représentée sous forme de vecteur ayant comme valeur à la k^{ieme} coordonnée 1 si le k^{ieme} mot de E est dans la requête et 0 sinon. Pour calculer si un document d répond bien à une requête q, on utilise une mesure de similarité $sim_{q,d}$ correspondant au cosinus de l'angle formé par le vecteur D du document et le vecteur Q de la requête :

$$sim_{q,d} = \frac{\sum_{m \in Q} w_{m,q}.w_{m,d}}{\sqrt{|Q|.|D|}}$$

Cette mesure permet de classer les documents suivant leur similarité avec la requête. Elle privilégie d'une part les documents dont les mots de la requête apparaissent plusieurs fois et d'autre part ceux ayant les mots apparaissant dans peu de documents et donc qui les discriminent.

2.1.4.2 PageRank

L'algorithme du PageRank [4, 11, 12, 9] a été créé par les fondateurs de Google : Sergey Brin et Lawrence Page et publié dans un article en 1998.

Le calcul du PageRank repose sur les liens hypertextes entre pages. Il se base sur l'intuition qu'un auteur de page web met des liens sur sa page s'il juge les pages pointées pertinentes. Par conséquent plus une page est pointée, plus le nombre de personnes ayant jugé la page pertinente est grand. Et plus une page est pertinente, plus les pages qu'elle va pointer vont être pertinentes.

De manière plus formelle, la valeur du PageRank d'une page A représente la probabilité qu'un surfeur aléatoire arrive sur la page A en suivant aléatoirement les liens hypertextes. Un surfeur aléatoire a deux possibilités pour arriver sur une page, soit il y accède directement en tapant l'URL de la page dans son navigateur ou en utilisant son marque-page, soit il y accède en suivant un lien. Le PageRank $PR(A)$ d'une page A se calcule de la manière suivante :

$$PR(A) = (1-d) + d.\sum_{p_i \in L(A)} \frac{PR(p_i)}{C(p_i)} \tag{2.1}$$

avec d (≈ 0.85) la probabilité que le surfeur aléatoire suive un lien et $(1-d)$, la probabilité qu'il aille directement sur la page, $L(A)$ l'ensemble des pages qui pointent vers la page A et $C(p_i)$ le nombre de liens sortants de la page p_i. Le surfeur aléatoire arrivant sur une page avec plusieurs liens sortants choisit de manière équiprobable le lien qu'il va emprunter.

La valeur du PageRank converge au bout d'un certain nombre d'itérations. La vitesse de convergence dépend de la valeur de d appelée aussi coefficient d'amortissement. La valeur du PageRank converge lentement si d est proche de 0 et oscille autour de sa valeur de convergence si d est proche de 1.

La valeur de 0.85 permet au PageRank de se stabiliser au bout d'une cinquantaine d'itérations dans le cas d'un graphe avec 322 millions de liens.

Les deux exemples (voir figures 2.1 et 2.2) montrent les valeurs du PageRank pour deux topologies de réseau de pages différentes. Dans l'exemple 2.1 où chaque page est reliée de la même manière aux autres, la probabilité qu'un surfeur aléatoire arrive sur l'une des pages est identique. Dans l'exemple 2.2, la page C qui est pointée par trois pages a le plus grand PageRank, la page A, étant la seule page pointée par C, hérite de tout le PageRank que peut fournir C. La page B hérite de la moitié du PageRank de A qui a deux liens sortants. Finalement D qui n'a aucun lien entrant a la valeur du PageRank le plus faible $1 - d$.

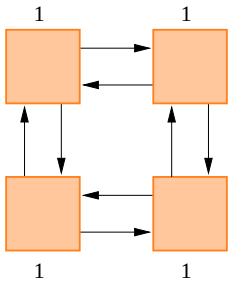


FIG. 2.1: PageRank calculé sur un réseau de pages liées de manière uniforme

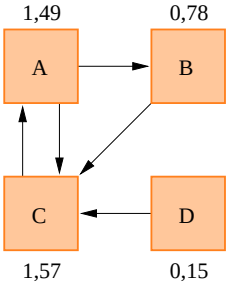


FIG. 2.2: PageRank calculé sur un réseau de pages quelconque

2.1.4.3 HITS

HITS [8] (HyperText Induced Topic Search) a été introduit par Kleinberg en 1998 et est utilisé par plusieurs moteurs de recherche dont Teoma (Ask Jeeves), WiseNut et WebFountain d'IBM. Il permet de pouvoir faire ressortir, pour un sujet donné, les pages centrales (authorities) qui sont pointées par beaucoup d'autres pages (donc doivent avoir un contenu pertinent) et les pages connecteurs (hubs) qui pointent vers beaucoup d'autres pages. Contrairement au PageRank, HITS est relatif à une requête et doit donc être exécuté à chaque recherche.

L'algorithme est le suivant : partant d'une requête de recherche, HITS commence par récupérer un ensemble de pages qui conviennent à la requête en utilisant un moteur de recherche existant comme AltaVista ou Hotbot. Les k (*e.g.* 200) premières pages S retournées, les pages ayant un lien hypertexte vers une des pages de S et les pages vers lesquelles pointent les pages de S vont former un sous-graphe G du web relatif à la requête. Ce sous-graphe G est composée de l'ensemble des pages V et de l'ensemble des liens hypertextes E entre les pages de V.

A chacune des pages p du sous-graphe G, on associe deux valeurs qui sont calculées de manière itérative : la centralité $A(p)$ et la connexité $H(p)$ qui sont initialisées à 1. A chaque itération, on calcule la nouvelle valeur de centralité et de connexité de chacune des pages de la manière suivante :

$$A(p) = \sum_{q/(q,p)\in V, e_{qp}\in E} H(q) \tag{2.2}$$

$$H(p) = \sum_{q/(p,q)\in V, e_{pq}\in E} A(q) \tag{2.3}$$

avec e_{ij} le lien entre la page i et la page j.

La valeur de centralité d'une page p est égale à la somme des valeurs de connexité des pages pointant vers p. De même, la valeur de connexité d'une page p est égale à la somme des

valeurs de centralité des pages qui sont pointées par p. Les valeurs de centralité et de connexité sont ensuite normalisées. On réitère le calcul jusqu'à ce que ces deux valeurs convergent.

Grâce à ces deux valeurs on peut classer les pages et connaître les pages centrales (celles qui ont la plus grande valeur de centralité) et les pages connecteurs (celles qui ont la plus grande valeur de connexité).

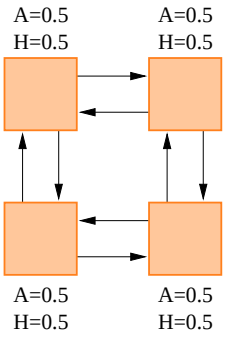


FIG. 2.3: HITS calculé sur un réseau de pages liés de manière uniforme

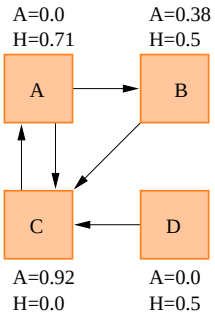


FIG. 2.4: HITS calculé sur un réseau de pages quelconque

Sur l'exemple 2.3 où toutes les pages sont reliées de manière uniforme, toutes les pages ont la même valeur de centralité et de connexité. Sur l'exemple 2.4, la page C qui reçoit le plus de liens entrants a la plus grande valeur de centralité. Par contre, comme elle pointe uniquement vers une page ayant une valeur de centralité nulle, elle a une valeur de connexité nulle. La page A qui pointe vers deux pages ayant une valeur de centralité importante a une valeur de connexité élevée. La page B qui est pointée par un connecteur important a une grande valeur de centralité et comme elle pointe vers une page ayant une grande valeur de centralité, elle a une valeur de connexité élevée. Comme la page D n'est pointée par aucune page, elle a une valeur de centralité nulle. En pointant sur la page C qui a une grande valeur de centralité, D a une valeur de connexité élevée.

2.1.4.4 Conclusion

Nous avons vu trois critères de classement. Le premier critère TFIDF permet de classer les documents par rapport à une requête en accordant plus de poids aux mots de la requête qui discriminent le document. Le deuxième critère, PageRank, indépendant de la requête, utilise les liens comme des recommandations positives sur les pages pointées pour calculer la popularité des pages. Le troisième critère HITS, à l'instar du PageRank, utilise les liens entre pages mais dépend de la requête. Il définit deux critères : la connexité et la centralité et permet d'exhiber les pages centrales (celles qui ont un contenu pertinent) des pages connecteurs (celle qui ont des liens qui pointent sur des pages intéressantes). Son défaut est qu'il doit être calculé à chaque requête.

2.1.5 Défauts des moteurs de recherche

Indexation non à jour Le crawler (ou robot d'indexation) ne peut savoir qu'une page a été modifiée ou supprimée qu'en revisitant la page. Or avec 8 milliards de pages, revisiter toutes ces pages prend du temps, une étude[4] effectuée en mai 2003 montre que la période

[4]Search Engine Showdown. *Search Engine Statistics : Freshness Showdown.* `http://www.searchengineshowdown.com/stats/freshness.shtml`. 17 mai 2003.

entre deux crawls peut varier de 2 à 165 jours avec une période moyenne de 1 mois. Avec une indexation non à jour, des pages peuvent ne pas apparaître dans les résultats ou être présentes de manière erronée en se référant à des versions passées des pages dont les contenus ont été modifiés. Pour remédier à ce problème, Google revisite plus fréquemment les pages populaires (ayant un PageRank élevé) car elles ont d'une part plus de chance d'être modifiées et d'autre part d'apparaître dans les réponses de recherche.

Pages non indexées Le crawler ne peut pas indexer toutes les pages, typiquement les pages accessibles *via* un formulaire web (*e.g.* moteur de recherche de documents comme Google ou de livres comme Amazon). En effet, le nombre de requêtes différentes que l'on peut effectuer est potentiellement infini. Ces pages constituent en grande partie le *deep web*.

DEEP WEB

Le *deep web ou web invisible* est l'ensemble des documents (textes, vidéos, images, ...) accessibles publiquement sur le web mais qui ne sont pas indexés par les moteurs de recherche. En 2000, une étude effectuée par BrightPlanet [3] a montré que le *deep web* pouvait contenir jusqu'à 500 fois plus de documents que le web indexé. Ces pages peuvent pourtant avoir des contenus intéressants, c'est le cas par exemple des dictionnaires en ligne, des pages blanches/jaunes, `etc...`

Le crawler peut tout simplement ne pas indexer des pages car elles ne sont pas référencées ou non atteignables en suivant les liens. En effet, 24% des pages n'auraient pas de liens entrants, c'est ce que décrivent les chercheurs d'IBM dans leur modèle du web en nœud de papillon [5].

La figure 2.5 représente les 5 parties du web et la répartition des pages dans chacune des 5 parties telle qu'elles sont estimées en l'année 2000.

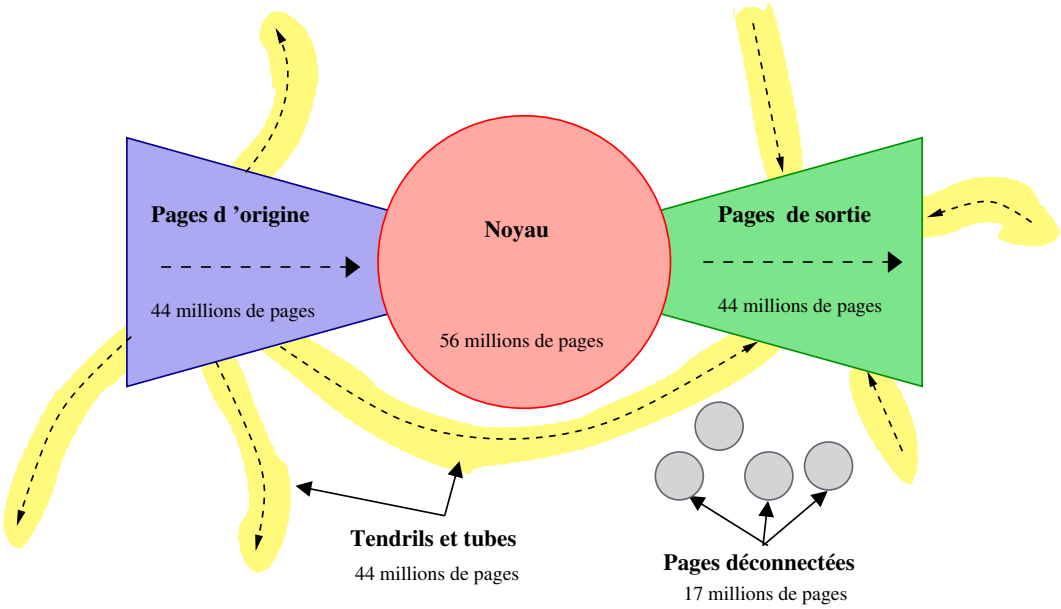


FIG. 2.5: Modèle du nœud de papillon

Dans ce modèle, le web est décomposé en 5 parties distinctes accessibles de manière très inégale :

- le **noyau** forme un graphe connexe de pages où il existe toujours au moins un chemin entre deux pages. Cette partie est facilement atteignable donc facilement référencée par les moteurs de recherche.
- les **pages d'origine** qui pointent vers les pages du noyau mais elles ne peuvent pas être atteintes en partant des pages du noyau. Ces pages sont difficilement atteignables par les crawlers et doivent donc être référencées manuellement.
- les **pages de destination** sont pointées par les pages du noyau mais n'ont pas de lien vers les pages du noyau. Ce sont par exemple les sites commerciaux qui sont pointés par de nombreuses pages mais qui n'offrent que peu de liens sortants en retour. Elles sont donc facilement trouvées par le crawler.
- les **tendrils et tubes** sont les pages accessibles depuis les pages d'origine et/ou qui pointent vers les pages de destination. Ces pages sont difficilement trouvées par les crawlers.
- les **pages déconnectées** ne pointent pas et ne sont pas pointées par les pages des quatre parties précédentes. Ces pages sont difficilement indexées par le moteur de recherche. Elles ne sont accessibles qu'à ceux qui connaissent leur URL exacte.

D'après la répartition des pages que l'on a sur le web en 2000, seules la moitié des pages seraient facilement accessibles en suivant les liens hypertextes.

Personnalisation des résultats Le moteur de requêtes ne gère pas la personnalisation des résultats : les réponses retournées par le moteur de recherche sont indépendantes de l'utilisateur. Dans Google, par exemple, comme le classement des pages est déterminé par les liens hypertextes entre les pages web et que les pages web sont surtout écrites par des informaticiens, les pages traitant d'informatique ont plus de chance d'apparaître dans les premières positions. En tapant les mots *apple*, *windows* ou *java* dans un moteur de recherche, les trois premiers résultats pointent respectivement sur le site du constructeur d'ordinateurs, sur le système d'exploitation et sur le langage de programmation. Or le mot *apple* par exemple peut être interprété d'au moins 3 façons : au sens du fruit, du nom du constructeur et du nom de la chanteuse pop/rock *Fiona Apple*. Et le résultat attendu dépend de l'utilisateur qui peut être soit un informaticien, une cuisinière ou un adolescent écoutant les musiques à la mode. Bien sûr, cela n'empêche pas l'utilisateur d'ajouter un mot à sa requête pour la préciser (et en quelque sorte inclure son profil dans la requête). Mais trouver le mot à rajouter (qui cooccure avec le mot cherché dans la page) peut être difficile dans certains cas.

Contrôle centralisé Les moteurs de recherche sont contrôlés par une seule entité qui garde secret leur fonctionnement. L'utilisateur doit donc faire confiance aux résultats fournis par le moteur de recherche. Or les résultats peuvent être biaisés et privilégier les sites web des partenaires. Ainsi, par exemple, une requête sur le mot *fleur*, va privilégier les sites de vente de fleurs. Ou pire, censurés, en 2002, suite à une plainte de l'Eglise de la scientologie pour violation de droits d'auteurs (DMCA)[5], Google a dû retirer de ses index un site qui relatait ses abus. On peut aussi citer la censure pour la même raison de sites relatifs à KaZaA Lite[6], une version modifiée de KaZaA. Ces moteurs de recherche posent aussi des problèmes de confidentialité étant donné que les requêtes de recherche ainsi que l'identifiant des utilisateurs qui les ont tapés sont conservés.

[5] ZDNet France. *L'Église de scientologie bute contre le « remue-ménage »* , 2002, `http://www.zdnet.fr/actualites/internet/0,39020774,2107320,00.htm`. 25 mars 2002.

[6] ZDNet France. *Sharman Networks gronde et Kazaa Lite K++ disparaît de la circulation.*. `http://www.zdnet.fr/actualites/internet/0,39020774,39133151,00.htm`. 9 décembre 2003.

Aspect financier Google gère 250 000 serveurs sous Linux en décembre 2004 [6] et leur nombre ne cesse de croître. Sa principale source de revenu est essentiellement basée sur les publicités ciblées affichées sur les pages de résultats de leur moteur de recherche. Or un utilisateur est volatil : il n'est pas lié à un moteur de recherche et peut en utiliser un autre si un moteur ne lui satisfait pas. Les principaux acteurs : Google, Yahoo !, MSN doivent sans cesse innover afin d'attirer de nouveaux utilisateurs. Or cette recherche ainsi que l'achat et la maintenance des serveurs demandent des investissements financiers importants qui augmentent sans cesse avec la croissance exponentielle du web.

2.2 Systèmes de recherche décentralisés non structurés

2.2.1 Principes

Le pair à pair (peer-to-peer en anglais ou P2P en abrégé) désigne un protocole de communication dans lequel les communications ne se font pas exclusivement entre un nœud et des serveurs dédiés (modèle client-serveur) mais entre n'importe quel nœud. Chaque nœud joue à la fois le rôle de client et de serveur et possède les mêmes fonctionnalités. Les réseaux pair à pairs permettent de partager des informations, des données, ou encore de la capacité de calcul.

Bien que le terme pair à pair soit devenu populaire à partir de 1999 grâce à l'apparition d'applications pair à pair d'échange de fichiers comme Napster et Gnutella, le concept de pair à pair est plus ancien. Par exemple, le protocole de communication de news NNTP dans le réseau Usenet créé en 1987 est décentralisé et peut être qualifié de pair à pair, de même que le protocole de mail SMTP.

Les systèmes qualifiés de pair à pair n'ont pas tous le même degré de décentralisation : si certains sont totalement décentralisés comme Gnutella v0.4, d'autres systèmes comme Napster ou eDonkey s'appuient sur des serveurs dédiés pour l'indexation. Dans ces systèmes, seul le transfert des fichiers est décentralisé.

Dans la suite, nous ne nous intéresserons principalement qu'aux systèmes purement décentralisés.

Nous traiterons dans cette section des systèmes non structurés, c'est-à-dire des systèmes où chaque nœud est autonome, ses connexions avec les autres nœuds ne lui sont pas contraintes pour satisfaire une topologie de réseau particulière. Dans ces systèmes, l'indexation est partitionnée par document, c'est-à-dire que les nœuds n'indexent localement que les documents qu'ils partagent.

Nous commencerons dans cette section par présenter le système Napster. Puis nous décrirons le protocole Gnutella ainsi que les différentes études qui ont été menées pour l'améliorer. Nous terminerons par les systèmes Freenet et FASD, qui sont à mi-chemin entre les systèmes non structurés et structurés.

2.2.2 Napster : le précurseur

Napster a été créé en 1998 par Shawn Fanning, un étudiant de 18 ans à l'université de Boston (Massachussets). La principale innovation de Napster a été de proposer une manière simple d'échanger des fichiers entre les utilisateurs sans avoir besoin de serveurs dédiés de distribution des fichiers. Dans Napster, les utilisateurs téléchargeaient les fichiers directement chez les autres utilisateurs. Un groupe de serveurs d'index dédiés maintenait la liste des fichiers partagés par les utilisateurs du système et répondaient aux requêtes de recherche en leur renvoyant une liste de fichiers associés aux nœuds qui pouvaient les fournir. Un client

effectuait une recherche en interrogeant un serveur d'index, puis le nœud n'avait plus qu'à télécharger directement le fichier sur le client possédant le fichier (voir figure 2.6).

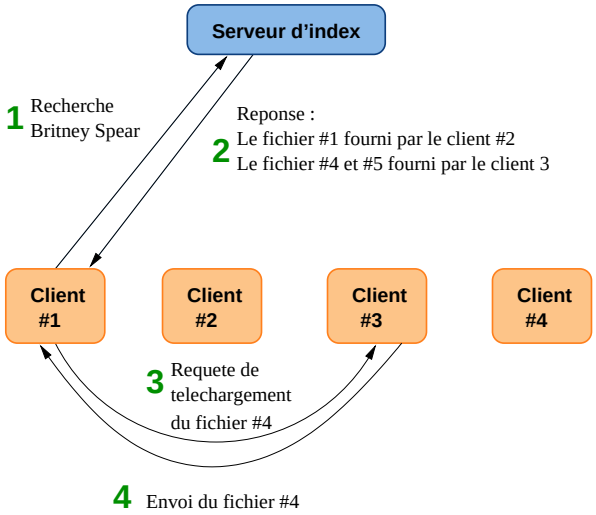


FIG. 2.6: Recherche et téléchargement dans Napster

Napster était rapidement devenu populaire et atteignait 13.6 millions d'utilisateurs en février 2001. Mais ce système n'est pas légal car il permettait l'échange de contenus musicaux en s'affranchissant des droits d'auteur. Dès décembre 1999, Napster fut attaqué par les majors du disque et finalement, en juillet 2001, fut contraint d'arrêter ses serveurs.

La centralisation du système est en grande partie responsable de la fermeture de Napster. En effet, dans ce cas, une entité pouvait être tenue responsable des échanges de fichiers et les serveurs d'index pouvaient être facilement arrêtés.

2.2.3 Gnutella 0.4

Gnutella [28, 26] est le premier système d'échange de fichiers où la recherche et le transfert de fichiers sont décentralisés. Il a été créé en 2000 par Justin Frankel et Tom Pepper. Gnutella qui comptabilise aujourd'hui 1,5 millions d'utilisateur[7] doit son succès notamment grâce à son protocole ouvert et libre qui lui a permis de connaître de nombreuses améliorations et d'avoir plusieurs implémentations libres ou commerciales qui ont facilité son déploiement.

Chaque nœud possède un ensemble de fichiers qu'il met à disposition des autres et qu'il indexe localement. Il est connecté à un ensemble de voisins (typiquement une dizaine).

La recherche se fait par inondation (voir figure 2.7) : le nœud Q envoie une requête de recherche à tous ses voisins qui la retransmettent à leur tour à tous leurs voisins. Pour limiter le nombre de retransmissions, un champ `TTL` (Time-To-Live, initialisé à 7) est associé à chaque message et est décrémenté de 1 à chaque retransmission. Quand celui-ci vaut 0, le message n'est plus retransmis.

Si un nœud trouve dans ses index un fichier qui convient à la requête, il envoie la description du fichier et son adresse `IP` au nœud qui lui a retransmis la requête. La réponse remonte en suivant le chemin inverse de la requête vers le nœud Q.

Le nœud Q choisit ensuite un fichier parmi les réponses reçues puis le télécharge directement sur le nœud qui a le fichier. Le fichier téléchargé est ensuite indexé et mis à la disposition des autres nœuds, ce qui augmente la disponibilité du fichier.

Les avantages de Gnutella sont évidents :

[7] Slyck. `http://www.slyck.com`

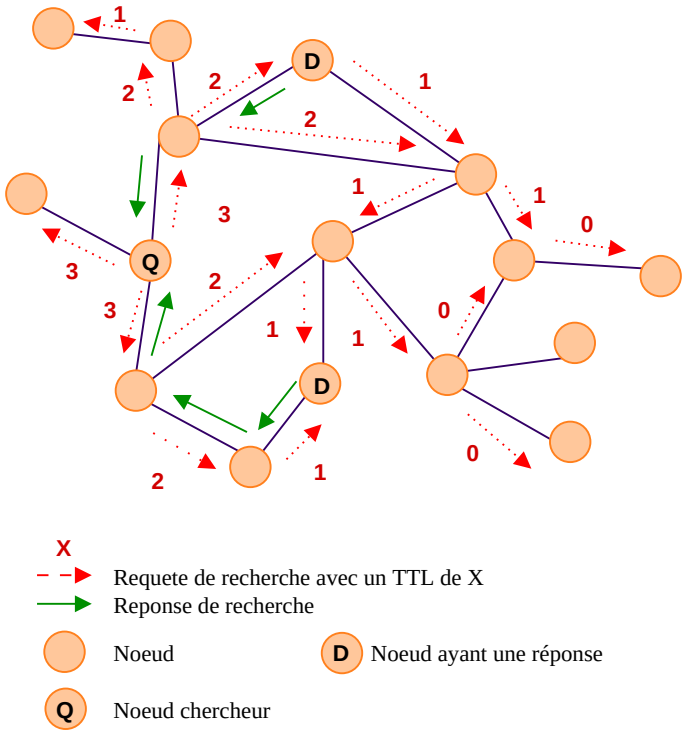


FIG. 2.7: Propagation d'une requête dans le système Gnutella

- **facilité de déploiement** : le déploiement du système ne demande pas de serveurs dédiés. Le coût financier du système correspond principalement au coût de développement du logiciel ;
- **prise en compte de la transience des nœuds** : les nœuds fugaces peuvent participer au système immédiatement après leurs connexions. Ce qui est souhaitable, vu que le temps médian de connexion d'un client Gnutella au système n'est que d'une heure [90] ;
- **réplication** : ce système permet la réplication des données très demandées et donc améliore leur disponibilité ;
- **résultats à jour** : les résultats de recherche correspondent aux données présentes sur le disque dur des utilisateurs à l'instant de la recherche.

Les défauts de Gnutella sont :

- **bande passante utilisée** : la recherche par inondation utilise beaucoup de bande passante. L'étude [38] montre que la bande passante utilisée pour fouiller le disque d'un million d'utilisateurs (typiquement ce que pouvait gérer Napster ou un seul serveur RazorBack pour eDonkey) est de 90 méga-octets par recherche ;
- **recherche non exhaustive** : comme une petite fraction des nœuds du réseau est interrogée, seule une petite partie des documents disponibles sur le réseau peut être recherchée ;
- **fichiers rares non trouvés** : seuls les fichiers populaires largement répliqués ont une chance d'être trouvés.

Ce système a permis de montrer que les réseaux pair à pair pouvaient être un moyen simple de recherche et d'échange de données sans avoir à déployer d'infrastructures onéreuses.

Des améliorations ont été proposées pour diminuer la bande passante utilisée et améliorer l'exhaustivité des recherches. C'est ce que nous allons étudier dans la suite.

2.2.4 Hiérarchie à deux niveaux

L'idée de ce modèle est d'exploiter les avantages des systèmes centralisés comme Napster et des systèmes décentralisés à la Gnutella. C'est ce qu'ont proposé Niklas Zennström et Janus Friis dans leur système KaZaA[8] (lancé en mars 2001) puis Limewire dans son client Gnutella (fin 2001).

La gestion des ultrapeers [41, 45] est maintenant incluse dans le protocole Gnutella v0.6 [33] et est aujourd'hui implémentée dans les principaux clients Gnutella.

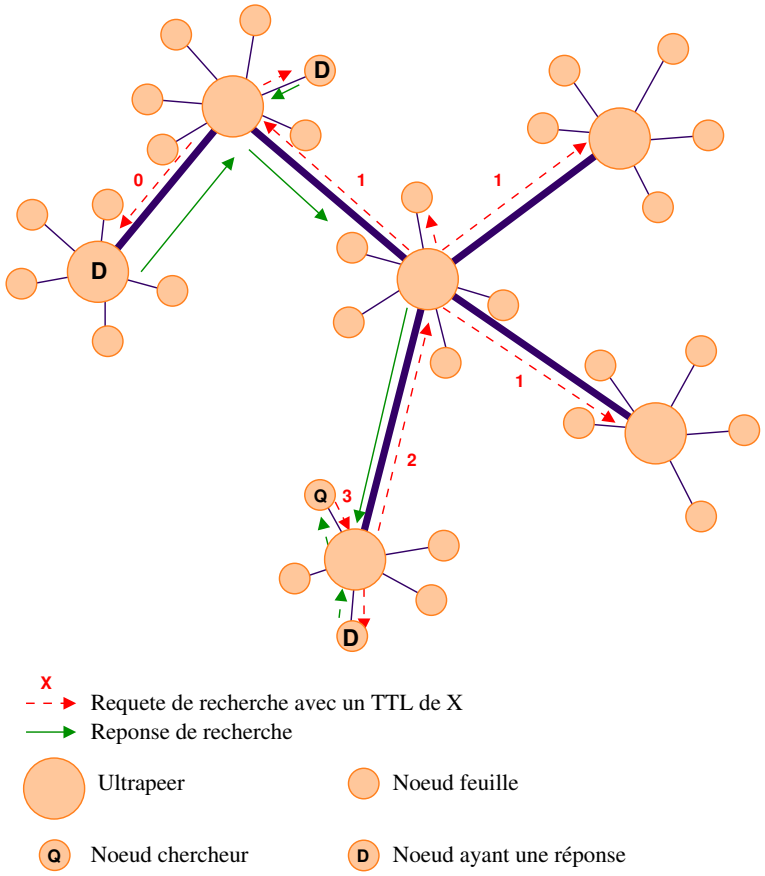


FIG. 2.8: Propagation d'une requête dans le système Gnutella 0.6

Dans ce modèle, il y a deux types de nœuds : les ultrapeers et les feuilles qui sont les clients des ultrapeers. Les ultrapeers jouent le rôle de serveurs d'index en indexant les fichiers de leurs clients. Plus exactement, les ultrapeers gèrent une table de routage associant à chacun de leurs clients, tous les mots contenus dans les noms des fichiers que possède le client. Ainsi quand un ultrapeer reçoit une requête de recherche, il ne la transmet qu'à ses clients qui peuvent lui fournir des réponses.

L'étude [91] montre que chaque ultrapeer est rattaché en moyenne à 30-45 clients et à 30 ultrapeers. Chaque client est connecté en moyenne à 3 ultrapeers et n'est connecté à aucun autre client. Il ne retransmet pas les requêtes de recherche reçues.

La recherche est effectuée dans le réseaux des ultrapeers de la même manière que l'inondation dans Gnutella 0.4. Pour effectuer une recherche (voir figure 2.8), le client Q envoie sa requête à ses ultrapeers. Cette requête est retransmise dans le réseaux des ultrapeers dans

[8] Sharman Networks. *KaZaA*. `http://www.kazaa.com`

un voisinage limité. Les réponses remontent en suivant le chemin inverse de la requête en traversant la chaîne d'ultrapeers jusqu'au nœud initiateur de la recherche.

Le protocole Gnutella 0.6 propose un mécanisme décentralisé de gestion des ultrapeers. Si des nœuds sont suffisamment stables (ils ont une bonne bande passante, restent connectés au réseau suffisamment longtemps, ont une bonne puissance de calcul), ils deviennent ultrapeers. Au contraire, un ultrapeer n'ayant plus de clients peut redevenir nœud feuille d'un autre ultrapeer.

Les avantages de cette hiérarchisation sont multiples, les nœuds feuilles ne reçoivent plus que les requêtes de recherche auxquelles ils sont capables de répondre. La recherche est plus exhaustive en utilisant beaucoup moins de messages : le contenu de plusieurs centaines de nœuds peut être fouillé en interrogeant un seul ultrapeer. L'étude [36] montre des gains de 2 à 13,9 pour le nombre de nœuds fouillés comparés à une approche à la Gnutella 0.4.

GUESS [30, 46] (Gnutella UDP Extension for Scalable Search) propose un mécanisme de recherche qui n'utilise pas l'inondation. Un des problèmes de l'inondation est qu'elle ne permet

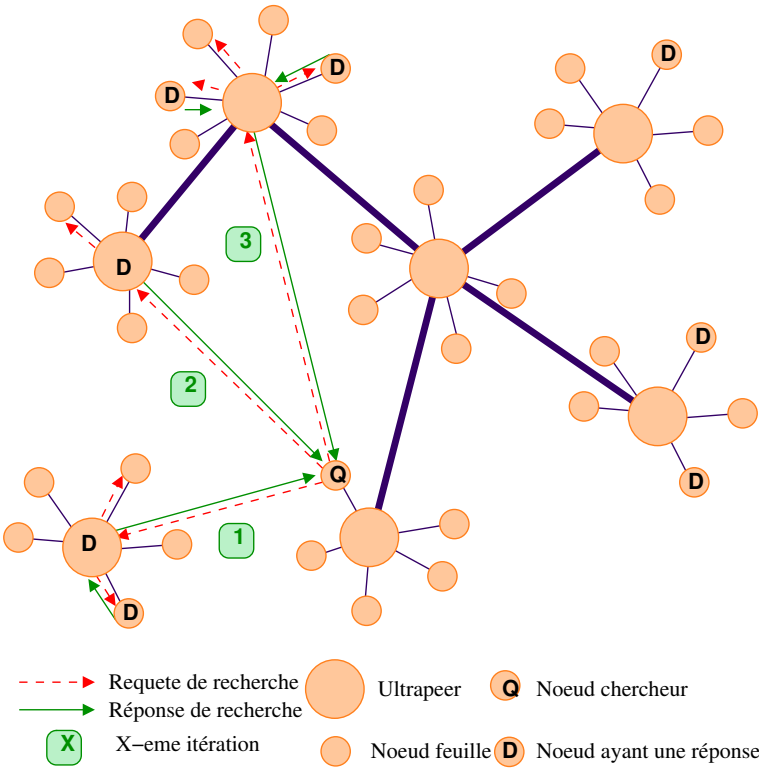


FIG. 2.9: Recherche en utilisant le protocole GUESS

pas de contrôler de manière efficace le nombre d'ultrapeers interrogés. Suivant la popularité des mots utilisés, le nombre d'ultrapeers à interroger pour obtenir un nombre donné de réponses peut être très variable. Pour résoudre ce problème, un nœud utilisant le protocole GUESS effectue sa recherche en interrogeant successivement un ensemble d'ultrapeers, qui ne retransmettent pas la requête (avec un TTL=1), jusqu'à ce qu'il reçoive un nombre suffisant de réponses (voir Figure 2.9). [36] montre des gains de 7 à 20 de la bande passante utilisée.

2.2.5 Retransmission itérative

L'idée est de limiter l'inondation en arrêtant de propager la requête quand un nombre suffisant de réponses est reçu. Pour implémenter la retransmission itérative [44], Yang et

Garcia-Molina définissent une politique $P = \{p_1, \ldots, p_i\}$ avec $p_1, \ldots, p_i$ des entiers dans l'ordre croissant représentant les nombres de retransmission (hops) à partir du nœud initiateur de la requête.

Un nœud a commence une recherche en émettant une requête de recherche avec un TTL de p_1 qui inonde les nœuds à une profondeur p_1. Les nœuds se trouvant à p_1 hops de a enregistrent la requête temporairement. Les nœuds qui ont reçus la requête renvoient leur réponses au nœud a. Le nœud a attend les réponses durant une période W puis si a est satisfait de sa recherche, a ne fait rien. Dans le cas contraire, a effectue une recherche plus en profondeur en envoyant un message *resend* avec un TTL de p_1. Les nœuds à p_1 hops de a recevant le message, retransmettent la requête qu'ils avaient enregistrée avec un TTL de $p_2 - p_1$. Les nouveaux nœuds interrogés renvoient leur réponses. Si a est satisfait des résultats, l'algorithme s'arrête sinon a lance une nouvelle retransmission. Les expérimentations montrent que cette technique permet de réduire la bande passante totale utilisée de 72% et de 53% la charge de calcul par rapport à une inondation classique. Par contre le temps qu'une recherche soit satisfaite est presque doublé.

2.2.6 Routage sémantique

Contrairement à l'inondation où la recherche est effectuée sur un ensemble de voisins choisis aléatoirement, le routage sémantique utilise les informations acquises associées aux autres nœuds pour déterminer les voisins à interroger qui ont de grande chance d'avoir des réponses. Les requêtes sont retransmises dans un voisinage limité. Cela permet entre autres d'interroger moins de voisins et donc d'utiliser moins de bande passante et de diminuer la latence tout en ayant autant voire plus de réponses.

Routage par similarité avec les requêtes passées Dans le système [55], les nœuds sont connectés à un ensemble de voisins et utilisent les messages de réponse reçus pour établir le profil de leurs voisins. Le nœud n gère pour chacun de ses voisins v, un profil $P(v)$ composé de la liste des k dernières requêtes de recherche que lui a envoyé n et pour lesquelles v a pu fournir au moins une réponse. Les requêtes de recherche sont typiquement un ensemble de mots clés. Le routage des requêtes de recherche s'effectue en transmettant la requête aux voisins ayant les profils les plus similaires à cette requête. La similarité $sim(v, q)$ entre le profil d'un voisin v et une requête q est calculée de la manière suivante :

$$sim(v, q) = \sum_{q_j \in P(v)} QSim(q_j, q)$$

La similarité $QSim$ entre deux requêtes est calculée par similarité cosinus entre les deux requêtes en utilisant le modèle VSM décrit précédemment dans la section 2.1.4.1. Les simulations montrent que ce système trouve autant de fichiers qu'une approche à la Gnutella tout en utilisant 3 fois moins de messages de recherche.

Routage vers les nœuds ayant bien répondu dans le passé Le système [49, 58, 60] utilise deux types de liens entre nœuds. D'une part, les nœuds sont connectés à un ensemble de voisins choisis aléatoirement comme dans le cas de Gnutella, cet ensemble est supposé immuable et permet de garantir la connexité du réseau. D'autre part, ils sont liés à un ensemble de connaissances. Chaque nœud gère une liste ordonnée de connaissances et la met à jour à chaque réception des résultats d'une recherche. Les requêtes sont retransmises aux connaissances les mieux classés dans la liste. Si la recherche échoue, la requête est renvoyée aux voisins.

Plusieurs critères ont été proposés pour classer les connaissances :

- **LRU** [49, 60] (Least Recently Used). En recevant une réponse, le nœud demandeur met en première position de la liste le nœud qui a émis la réponse. Le dernier nœud de la liste qui n'a pas fourni de réponses depuis longtemps est enlevée de la liste. Les simulations montrent que le nombre de retransmissions nécessaires (TTL) pour trouver une réponse avec 90% de réussite est de 3 comparé à une approche à la Gnutella qui en nécessite 5. Une autre simulation montre que 40% des recherches sont satisfaites contre 10% dans le cas où les connaissances sont choisies aléatoirement.
- **MOU** [49] (Most Often Used), chaque connaissance de la liste a un score initialisé à 0. Pour chaque réponse reçue, le nœud n connaît les nœuds C par lesquels la réponse a transité. Ces nœuds C sont rajoutés à la liste de connaissance de n s'ils n'y sont pas déjà présents. Le score de chacune des connaissances C est incrémenté d'une valeur exponentiellement proportionnelle à sa distance au nœud qui a émis la réponse. Afin de discréditer les nœuds qui n'ont pas répondu depuis longtemps, un mécanisme de vieillissement des scores est utilisé. A chaque réception de réponse, le score de chacun des nœuds de la liste des connaissances est multiplié par un coefficient de vieillissement (compris entre 0 et 1). Les simulations montrent des résultats presque identiques à l'approche précédente.
- **Ratio de réponses renvoyées**, dans le système [58], chaque nœud n associe à chaque connaissance v un score correspondant au ratio du nombre de requêtes que lui a envoyées n et pour lesquelles v a pu fournir des réponses sur le nombre total de requêtes que lui a soumis n. Pour effectuer une recherche, les connaissances ayant les ratios les plus élevés sont interrogées. Les simulations montrent que cette technique, comparée à une approche à la Gnutella, permet de diminuer par 6 le nombre de requêtes de recherche reçues par nœud et par 3 le nombre de hops par requête nécessaires pour trouver une réponse.
- **Historique**, [60] associe à chaque connaissance un compteur qui est incrémenté de un à chaque fois que celle-ci répond positivement à une requête. Les connaissances ayant le compteur le plus élevé sont interrogées. Les simulations montrent qu'en utilisant cette stratégie, 65% des recherches sont satisfaites contre 10% avec une approche où les connaissances sont choisies aléatoirement.
- **Popularité**[60]. L'idée est de privilégier les nœuds ayant répondu récemment et possédant des fichiers rares qui intéressent le nœud demandeur. Deux nœuds ayant ou étant intéressés par le même fichier vont avoir une probabilité élevée d'avoir des fichiers qui pourraient intéresser l'autre. Cette probabilité est d'autant plus forte que le fichier est rare.

  Chaque connaissance est associée à deux valeurs : *numrep*, le nombre de réponses et *lastreply*, la date à laquelle celle-ci a fourni la dernière réponse. Les recherches se font uniquement sur l'identifiant du fichier (nom complet). En recevant les réponses de recherche qui sont composées de quelques nœuds v possédant le fichier, le nœud initiateur n met à jour sa liste. Si la connaissance v est déjà présente dans la liste, le champ *lastreply* de v est mis à la date courante. Sinon, la connaissance v est rajoutée à la liste avec le champ *numrep* correspondant au nombre de nœuds présents dans la réponse de recherche. Le champ *lastreply* est mis à la date courante.

  Le nombre de connaissances étant limité, celles n'ayant pas fourni de réponses depuis une période donnée sont éliminées. Et les connaissances ayant les valeurs *numrep* les plus élevées, donc qui ont fourni des documents populaires, sont évincées au profit des nouvelles connaissances. Avec cette stratégie, 50% des recherches sont satisfaites contre 10% dans le cas où les connaissances sont choisies aléatoirement.

APS : Routage probabiliste adaptatif Dans le système APS [59](Adaptive Probabilistic Search), les recherches ne se font pas par mots clés mais sur les identifiants des fichiers. Chaque nœud possède un ensemble fixé de voisins et maintient une table associant à chaque couple voisin-fichier un score initialisé à 30. La recherche ne se fait pas par inondation mais est limitée à un seul voisin par retransmission, le nombre de retransmissions est contrôlé par un champ TTL associé aux messages de requête de recherche. La retransmission s'arrête lorsque le TTL est nul ou que le fichier est trouvé. Deux politiques sont utilisées pour mettre à jour les scores. La première est l'approche dite optimiste : lors de la réception d'une requête de recherche, le nœud sélectionne dans sa table un voisin à qui retransmettre la requête avec une probabilité proportionnelle au score du couple voisin-fichier recherché et l'augmente de manière optimiste de 10. Si la recherche réussie, rien n'est fait. Au contraire, si la recherche échoue (le TTL a expiré), le dernier nœud à recevoir la requête va renvoyer une réponse négative qui va suivre le chemin inverse de la requête jusqu'au nœud demandeur. Les nœuds recevant ce message vont mettre à jour leur table en décrémentant de 20 le score voisin-fichier du voisin qui lui a retransmis la réponse négative. La deuxième approche est l'approche dite pessimiste. Contrairement à l'approche précédente, le nœud va décrémenter de 10 le score voisin-objet à qui il va retransmettre la requête. Et ce n'est que lorsque la recherche réussit que le nœud qui possède la donnée va envoyer une réponse positive jusqu'au nœud demandeur. A la réception de cette réponse positive, les nœuds incrémentent de 20 le score voisin-objet du voisin qui leur a retransmis la réponse. Les expériences comparent ce système avec un routage aléatoire et montrent un nombre de recherches satisfaites de plus de 30% sans utiliser beaucoup plus de messages.

Neurogrid : Routage sémantique adaptatif Dans le système Neurogrid [54], les nœuds routent les requêtes en fonction du profil sémantique de leurs voisins. Ces profils sont gérés localement au niveau de chaque nœud et sont mis à jour à chaque réception de réponses de recherche. Le profil d'un voisin v maintenu par le nœud n est constitué d'un ensemble de mots m associé à deux valeurs :

- $rep_n(v, m)$, le nombre de fois où le voisin a fourni une réponse pour le mot m à n
- $relrep_n(v, m)$, le nombre de fois où les réponses fournies par le voisin ont conduit l'utilisateur du nœud n à télécharger le document.

En divisant $relrep_n(v, m)$ par $rep_n(v, m)$, on obtient le ratio du nombre de fois où le voisin a fourni une bonne réponse.

Cependant comme l'auteur le stipule, ce ratio n'est pas significatif car il ne tient pas compte du nombre de fois où le voisin v a fourni une réponse. En effet, plus la valeur de $rep_n(v, m)$ est importante, plus la valeur du ratio est précise. Pour tenir compte de ce facteur, on utilise la borne de Hoeffding [113] (voir encadré) permettant de calculer la valeur pessimiste du ratio avec un niveau de confiance donné.

BORNE DE HOEFFDING

La *borne de Hoeffding* majore la probabilité que la moyenne de n valeurs tirées aléatoirement dévie de sa valeur attendue.

Théorème 1 *Soit ξ_i, $i \in [n]$, n instances indépendantes d'une variable aléatoire (v.a.) ξ avec $\xi \in [a, b]$. Et Q_n sa moyenne, $Q_n = \frac{1}{n}\sum_i \xi_i$.*

Alors pour tout $\epsilon > 0$

$$P[Q_n - E(\xi) \geq \epsilon] \leq e^{-\frac{2.n.\epsilon^2}{(b-a)^2}}$$

$$P[Q_n - E(\xi) \leq -\epsilon] \leq e^{-\frac{2.n.\epsilon^2}{(b-a)^2}}$$

Donc :

$$P[|Q_n - E(\xi)| \geq \epsilon] \leq 2.e^{-\frac{2.n.\epsilon^2}{(b-a)^2}}$$

Dans le cas qui nous intéresse, on cherche la déviation maximale ϵ_{max} connaissant le nombre de tirages n et la confiance c correspondant à la probabilité que la moyenne ne dévie pas de plus de ϵ_{max}.

$$c = 1 - P[|Q_n - E(\xi)| \geq \epsilon]$$

Par le calcul, on obtient comme valeur de ϵ_{max} :

$$\epsilon_{max} \leq \sqrt{\frac{(b-a)^2.ln(\frac{1-c}{2})}{2.n}}$$

La valeur pessimiste de la moyenne est donc :

$$Q_n - \sqrt{\frac{(b-a)^2.ln(\frac{1-c}{2})}{2.n}}$$

Ce ratio pessimiste va être utilisé pour router les requêtes de recherche vers les nœuds ayant le ratio le plus élevé.

Le système implémente aussi le classement des documents par pertinence grâce à une gestion locale du profil sémantique des documents. Le profil d'un document d géré par le nœud n est composé d'un ensemble de mots où chaque mot m est associé :

- $rep_n(v, d)$ au nombre de fois où le document d apparaît dans les réponses d'une requête sur m
- $relrep_n(v, d)$ le nombre de fois où les réponses d'une requête sur m conduisent l'utilisateur du nœud n a télécharger le document d.

De la même manière que précédemment, la valeur pessimiste du ratio de $relrep_n(v, d)$ sur $rep_n(v, d)$ est utilisée. Les documents sont classés par ordre décroissant de la valeur pessimiste du ratio.

Les simulations montrent que le nombre de retransmissions nécessaires pour trouver une réponse diminue au fur et à mesure que le système apprend. La distance entre les nœuds demandeurs et les nœuds fournisseurs est divisée par 2 ou par 3 suivant les hypothèses de la simulation.

Freenet : spécialisation par clé des nœuds Freenet [51, 50] est un système pair à pair permettant la publication, la réplication et la récupération de données tout en assurant l'anonymat des auteurs et des lecteurs, et en étant résistant aux censures. Il a été décrit par Ian Clarke, un étudiant à l'université d'Edinburgh en 1999.

Dans ce système, chaque fichier est associé à une clé générée par une fonction de hachage sur son contenu. La recherche se fait uniquement sur la clé du document. Chaque nœud maintient une table associant, pour tous les fichiers qu'il a publiés ou mis en cache, le nœud source qui lui a fourni le document.

Pour rechercher un document de clé k, le nœud cherche dans sa table le fichier f ayant la clé la plus proche de k, puis envoie sa requête au nœud source s de f. Le nœud s retransmet

la requête à un nœud choisi de la même manière que précédemment. Lorsqu'un nœud possède le fichier, il retourne son contenu au nœud qui lui a retransmis la requête tout en lui précisant le nœud source. Le contenu du document et sa source sont retransmis en suivant le chemin inverse de la requête vers le nœud demandeur. Les nœuds intermédiaires mettent dans leur cache le fichier associé à son nœud source.

La publication d'un fichier f s'effectue de la même manière. Le nœud envoie le fichier f au nœud source s du fichier ayant la clé la plus proche de f dans sa table tout en se proclamant source du fichier. Le nœud s met dans son cache le fichier et le retransmet à un autre nœud suivant le même critère de sélection.

Un champ est associé aux messages de recherche et de publication pour limiter leur retransmission.

L'anonymat dans Freenet est géré de plusieurs façons :

- **anonymat du chercheur** : d'une part, l'identifiant du nœud demandeur n'est pas dans la requête et d'autre part les retransmissions de requêtes ne permettent pas de connaître l'initiateur de la recherche
- **anonymat de l'auteur** : lors de la publication d'un document, les nœuds qui retransmettent la requête de publication peuvent modifier avec une certaine probabilité le nœud source du fichier et se proclamer source du fichier. De même pour les réponses de recherche qui reviennent vers l'initiateur de la requête. Cela permet de cacher l'identifiant de l'auteur du document tout en permettant à une requête de recherche d'être routée vers le fournisseur du document.
- **aucune responsabilité sur le contenu du cache** : un utilisateur ne peut pas connaître le contenu des fichiers qu'il a dans son cache grâce au chiffrement des données. Il ne peut par conséquence en être responsable.

La mise en cache agressive des documents lors de leurs publications et de leurs recherches couplé aux mécanismes de routage par proximité de clé entraîne une spécialisation des nœuds par clés. Les nœuds tendent à avoir d'une part beaucoup de documents dans leur cache ayant des clés proches et d'autre part connaître ou être connus par beaucoup de nœuds spécialisés dans des clés proches des leurs.

Les simulations du systèmes montrent que les nœuds se spécialisent par clé et que la distance entre le nœud demandeur et le nœud fournisseur diminue avec le temps. Cette distance ne croit que de manière logarithmique avec le nombre de nœuds. Autre propriété, le réseau exhibe des propriétés *small world* et *scale free*. Un réseau *scale free* est caractérisé par une distribution exponentielle des degrés de ses nœuds (ici, le nombre de nœuds sources dans la table de routage du nœud) qui se traduit par le fait que la majorité des nœuds ont relativement peu de connexions avec les autres nœuds mais qu'un petit ensemble de nœuds a beaucoup de connexions. Un réseau *small world* exhibe deux propriétés : d'une part, la distance moyenne entre deux nœuds est courte et d'autre part, la probabilité que deux nœuds voisins d'un troisième (ici, qui apparaissent dans la table de routage du troisième nœud) soient voisins entre eux (ici, que le premier ait le deuxième nœud dans sa table et *vice versa*) est élévée.

FASD FASD [57](A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine) est un moteur de recherche par mots clés de documents dans un réseau de pairs fonctionnant à la Freenet. Ce système ne décrit que la recherche, les autres fonctions comme le téléchargement et la publication de contenus sont assurés par un autre système (*e.g.* Freenet). Chaque document est identifié par une clé et possède une description composée de la clé du document et une liste de mots pondérés par TFIDF.

Chaque nœud maintient une table de descriptions de documents associés aux nœuds sources qui lui ont fourni les descriptions.

L'algorithme de recherche est assez similaire à celui de Freenet sauf que la recherche ne se termine pas au premier résultat trouvé et que la requête est routée vers le nœud source qui possède la description la plus similaire (similarité cosinus) aux mots de la requête.

Les résultats sont retournés au nœud demandeur en empruntant le chemin inverse de la requête. Un nœud recevant les réponses met en cache les descriptions reçues dans sa table de descriptions. Puis il les classe avec les siennes en fonction de leur similarité cosinus avec la requête. Seules les meilleures descriptions de documents sont retournées au nœud amont.

Comme dans Freenet, les nœuds se spécialisent par documents ayant une description proche. Les simulations montrent que le nombre de hops nécessaires pour trouver les k meilleurs résultats diminue fortement avec le temps passant de 500 à 100. De même que dans Freenet, ce nombre de hops n'augmente que de manière logarithmique avec la taille du réseau.

2.2.7 Bilan

Différentes approches ont été proposées pour augmenter l'exhaustivité des résultats tout en diminuant la bande passante utilisée dans les systèmes décentralisés comme Gnutella 0.4. Une approche est de centraliser les informations en proposant une architecture à deux niveaux avec d'une part les ultrapeers qui jouent le rôle de serveurs d'index et d'autre part leurs clients. Cette architecture permet de profiter des avantages des systèmes centralisés comme Napster tout en gardant les avantages de Gnutella 0.4. Une autre approche décrit par GUESS et le mécanisme de retransmission itérative est de laisser l'initiateur de la requête contrôler le nombre de nœuds interrogés en fonction du nombre souhaité de résultats. Cette approche permet d'économiser la bande passante utilisée mais au prix d'un temps de recherche plus élevé. Enfin, nous avons décrit plusieurs techniques de routage sémantique. Le routage sémantique contrairement à Gnutella où les voisins sont choisis au hasard, propose d'utiliser divers critères pour choisir le voisinage d'un nœud de manière à augmenter la probabilité qu'une requête soit satisfaite tout en utilisant moins de bande passante.

2.3 Systèmes de recherche décentralisés structurés

Les systèmes structurés dans les réseaux pair à pair sont principalement basés sur les tables de hachage distribuées (*Distributed Hash* Table ou DHT).

Nous allons décrire dans cette section ce qu'est une DHT et trois topologies couramment employées pour l'implémenter. Puis nous étudierons plusieurs systèmes de recherche de documents par mots basés sur les DHT.

2.3.1 Les tables de hachage distribuées

Une *table de hachage* est une structure qui associe un élément à une clé. On accède à chaque élément de la table *via* sa clé. Les deux opérations élémentaires d'une table de hachage sont la création d'une association et la récupération d'un élément par sa clé.

Une *table de hachage distribuée* répartit les associations clé-élément sur les nœuds du réseau. Chaque nœud est responsable de quelques unes de ces associations, typiquement d'un intervalle de clés. A chaque nœud est associé un identifiant qui lui indique l'intervalle de clés dont il a la responsabilité. Suivant la nature de l'application, les clés et les éléments représentent différentes choses. Dans le cas d'une application d'échange de fichiers, les clés sont les identifiants des fichiers et l'élément est la liste des adresses des nœuds qui possèdent le fichier. Si on voulais implémenter une application de messagerie instantanée par exemple

en utilisant une DHT, la clé correspondrait à l'identifiant du nœud (login) et l'élément serait l'adresse IP du nœud.

Dans ces systèmes, rechercher un élément revient dans un premier temps à retrouver le nœud responsable de sa clé, puis dans un deuxième temps à l'interroger sur la valeur de l'élément. De même, la publication d'un élément revient à trouver le nœud responsable de sa clé avant de lui envoyer l'association. Pour que la recherche de nœuds par clé soit efficace, les nœuds forment un réseau structuré où les voisins sont choisis suivant leur identifiant. Les topologies adoptées peuvent être l'anneau, le tore, l'hypercube, `etc...` Ces différentes topologies vont définir entre autre les algorithmes de routage pour chercher le nœud responsable d'une clé.

Nous allons décrire dans cette sous-section trois systèmes qui implémentent les DHT en utilisant des topologies différentes : l'anneau avec le système CHORD, le tore avec le système CAN et l'hypercube avec le système Pastry.

2.3.1.1 CHORD

La topologie utilisée dans CHORD [76] est l'anneau. Chaque nœud n a un identifiant id unique appartenant à l'espace $K = [0, 2^p - 1]$ qui définit sa position dans l'anneau avec p suffisamment élevé pour que l'espace K puisse accueillir assez de nœuds.

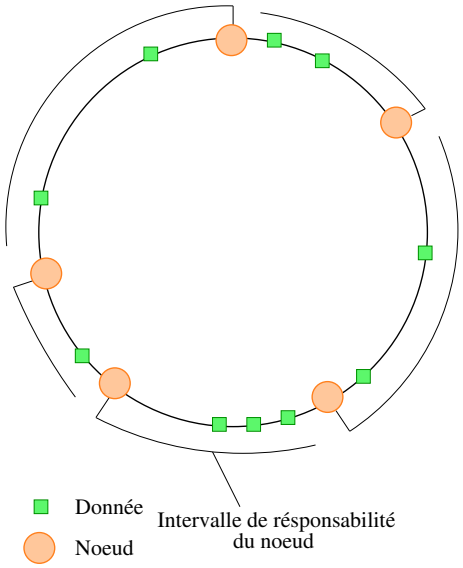


FIG. 2.10: Représentation de l'espace des responsabilités des nœuds

L'identifiant id définit d'une part l'espace des clés dont le nœud n est responsable (voir figure 2.10). Son espace des clés est délimité par son identifiant et celui du nœud qui a l'identifiant immédiatement inférieur. Et d'autre part, les nœuds voisins qu'il doit connaître (voir figure 2.11). Ces nœuds voisins sont choisis de telle sorte que le nœud n ait plus de connaissances de nœuds ayant un identifiant proche du sien que de nœuds ayant un identifiant éloigné du sien. Le nœud connaît pour chaque valeur $i = id + 2^i \bmod 2^n$, le nœud ayant l'identifiant immédiatement supérieur à i. Ces nœuds vont constituer la table d'index du nœud n.

La figure 2.11 décrit les tables d'index gérées par les nœuds et leur espace de clés dont ils sont responsable. L'espace des clés est $K = [0, 7]$ et il y a 4 nœuds présents dans le système : #0, #1, #3 et #6. Le nœud d'identifiant #6 est responsable de l'espace des clés comprises entre 4 à 6 puisque le nœud qui le précède dans l'anneau a comme identifiant #3. Sa table

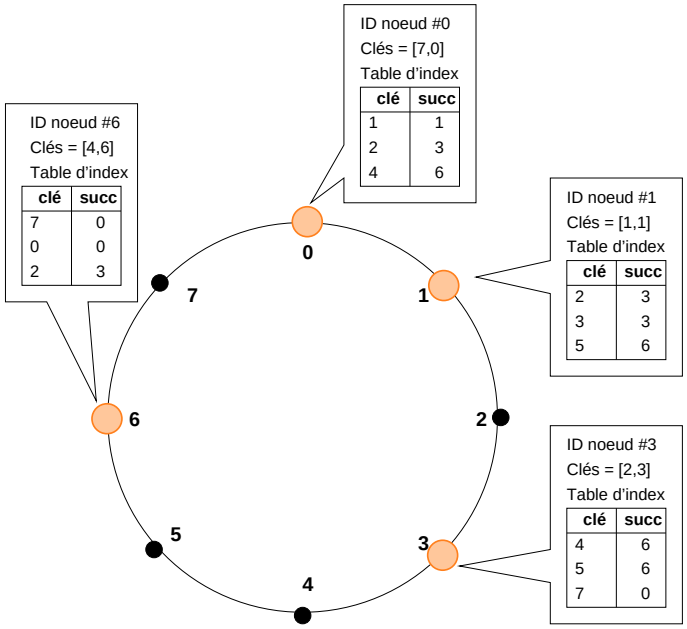


FIG. 2.11: Table d'index des voisins

d'index contient les nœuds dont les identifiants succèdent à : 7 : nœuds #0, 0 : nœuds #0 et 2 : nœuds #3.

Pour trouver le nœud responsable d'une clé k, c'est-à-dire le nœud dont l'identifiant est égal ou immédiatement supérieur à k, le nœud n interroge parmi sa liste de connexions, le nœud ayant l'identifiant immédiatement inférieur à k. Celui-ci répète le processus. La recherche est terminée quand le dernier nœud interrogé l ne connaît pas de nœud ayant un identifiant strictement inférieur à k. Dans ce cas, le nœud recherché est alors le nœud connu par l ayant un identifiant égal ou immédiatement supérieur à k.

La manière dont les intervalles sont choisis font que la recherche est effectuée de manière dichotomique et donc a une complexité en $o(log_2(p))$. On peut aussi noter qu'un nœud i peut avoir plusieurs nœuds qui le pointent dans leur table d'index ce qui améliore la tolérance aux fautes en cas de déconnexion de nœuds.

Pour se joindre au système, un nœud doit au moins connaître un nœud du système. Il commence par remplir sa table d'index en recherchant les nœuds responsables de l'identifiant du début de chaque intervalle I. Puis il doit se notifier auprès des nœuds qui pourraient l'avoir dans leur table d'index. Enfin, il doit récupérer les associations dont il devient responsable. Avant de partir, un nœud doit signaler son départ aux nœuds qui l'ont dans leur table d'index et envoyer ses associations au nœud qui possède l'identifiant immédiatement supérieur au sien et qui va en devenir responsable.

2.3.1.2 CAN

La topologie choisie dans CAN (Content Addressable Network) [72] est le tore. L'espace des clé K dans ce système est un espace à d dimensions avec d la dimension du tore. Chaque nœud a est responsable d'une zone A de K (voir figure 2.12) et doit connaître les nœuds v qui sont responsables des zones adjacentes à A. Nous appellerons ces nœuds v les voisin de n.

Lorsqu'un nœud a se connecte au système, il tire au hasard les coordonnées d'un point dans l'espace k, puis il cherche le nœud r responsable de P. La zone gérée par r est découpée

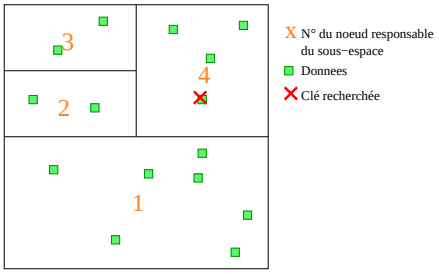


FIG. 2.12: Représentation de l'espace des responsabilités des nœuds

en deux et les nœuds a et r deviennent chacun responsable d'une moitié de la zone. Le nœud r envoie à a les associations dont il est devenu responsable. La découpe est effectuée en supposant un certain ordre sur la dimension pour lequel la zone va être découpée, de tel manière que les zones puissent être fusionnées quand les nœuds se retirent. Les nœuds a et r notifient les nœuds responsables des zones adjacentes à la zone découpée pour qu'ils puissent mettre à jour leur voisinage.

La figure 2.13 montre l'insertion de deux nœuds dans le réseau.

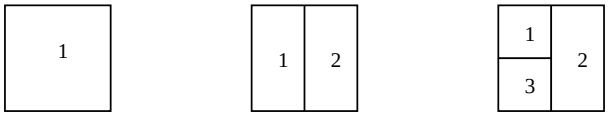


FIG. 2.13: Topologie en tore de CAN (ici de dimension 2)

Quand un nœud recherche une clé, il interroge son voisin qui gère la zone la plus proche de la clé recherchée. Celui-ci va faire de même jusqu'à atteindre le nœud responsable de la clé. Comme il existe plusieurs chemins pour arriver à un nœud, si un voisin tombe en panne ou se déconnecte, on peut encore retrouver le nœud responsable de la clé en suivant un autre chemin. La recherche s'effectue en $o(d(n^{1/d}))$.

2.3.1.3 Pastry

Pastry [74] utilise une topologie qui approxime l'hypercube. Les clés sont composées de n chiffres compris entre 0 et 2^b.

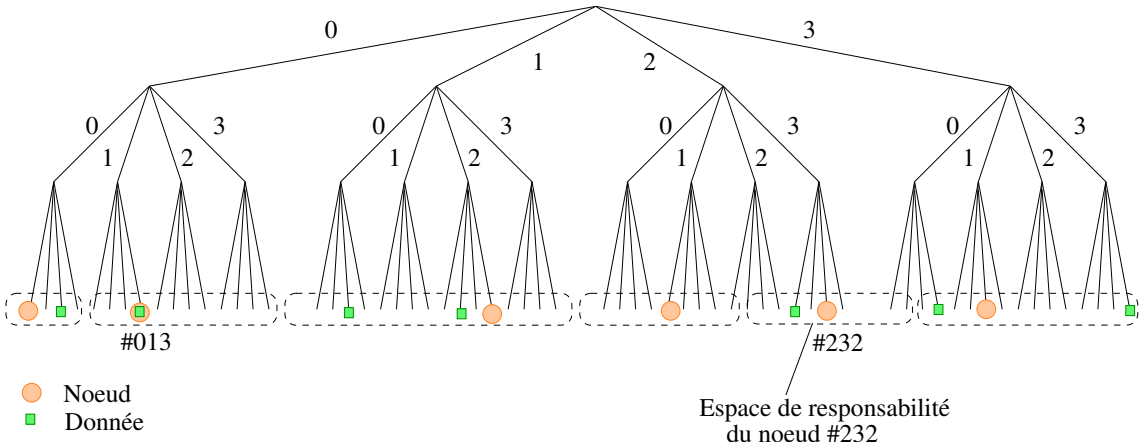


FIG. 2.14: Représentation de l'espace de responsabilité des nœuds

A l'instar des deux systèmes présentés précédemment, un nœud a une connaissance plus importante des nœuds dont l'identifiant est proche du sien, ici qui vont avoir le plus long préfixe commun. Pour chaque longueur l de préfixe, chaque nœud n connaît pour tous les chiffres i compris entre 0 et b, un nœud qui a une clé ayant un préfixe commun de longueur l avec celui de n et comme chiffre après le préfixe égal à i. Il maintient aussi une liste ordonnée L de nœuds ayant les identifiants les plus proches du sien (supérieurement et inférieurement). On note L_i le i^{ieme} élément de L supérieur à l'identifiant de n et L_{-i} le i^{ieme} élément inférieur avec $-\frac{|L|}{2} < i < \frac{|L|}{2}$.

Le tableau 2.1 montre un exemple de table de routage sur le nœud p d'identifiant 10233102 avec des clés de longueur 8 et de base 4 ($b = 2$). La k^{eme} rangée décrit les nœuds connus par p qui ont un préfixe commun de longueur $k-1$ avec celui de p. La i^{eme} case de cette rangée est remplie par l'identifiant d'un nœud dont le chiffre après le préfixe commun est égal à $i-1$. Or il y a une valeur de i pour lequel l'identifiant du nœud p convient, dans ce cas, on remplie cette case par -. Comme tous les identifiant des clés ne sont pas utilisés par tous les nœuds, il est normal que les dernières lignes de la table de routage du tableau ne sont pas toutes remplies. Le nœud p gère aussi une liste de nœuds proches qui est représentée en bas de la table.

NodeID 10233102			
Table de routage			
02212102	-	22301203	31203203
-	11301233	12230203	13021022
10031203	10132102	-	10323302
10233001	-	10233232	-
10233001		10233232	
		-	
Nœuds proches			
10233033	10233021	10233120	10233122

TAB. 2.1: Table de routage d'un nœud Pastry

Pour chercher le nœud responsable d'une clé k, le nœud demandeur d route sa requête de recherche suivant la valeur de k. Si $L_{-|L/2|} < k < L_{|L/2|}$, alors le nœud responsable de la clé se trouve dans l'ensemble des nœuds proches. La requête de recherche est transmise au nœud ayant l'identifiant plus proche de la clé. Sinon la requête de recherche est envoyée au nœud ayant le préfixe commun le plus long avec celui de k. Le nœud recevant la requête retransmet la requête de la même manière et ainsi de suite. La recherche est finie quand le nœud le plus proche de k est trouvé.

Cette technique permet typiquement d'atteindre à chaque retransmission un nœud ayant un identifiant dont le préfixe commun à k se complète d'un chiffre. Au final, le nombre de retransmissions nécessaires est en $o(log_{2^b}(n))$.

2.3.1.4 Conclusion

Les systèmes utilisant les DHT ont plusieurs avantages par rapport aux systèmes décentralisés :

- **recherche exhaustive** : le demandeur est sûr de trouver une donnée dans le réseau si elle existe ;
- **recherche peu coûteuse en temps et en bande passante utilisée**

Les inconvénients des DHT viennent de la rigidité du réseau :

- **connexions et déconnexions de nœuds coûteuses en bande passante** : un nouveau nœud qui se connecte au réseau va être assigné une responsabilité sur un espace de clé. Il doit donc récupérer les associations relatives à son espace de clés auprès des nœuds qui géraient cet espace. A la déconnexion d'un nœud, celui-ci doit transmettre aux nœuds qui géreront son espace la liste de ses associations. Pour que le système fonctionne correctement, il faut donc que les nœuds soient relativement stables pour éviter ces opérations très coûteuses en bande passante. Si un nœud se déconnecte sans prévenir ses voisins, ses associations sont perdues. Pour éviter ces pertes, des mécanismes de réplication ou de publication périodique des associations assez lourdes peuvent être implémentées.
- **pas d'équilibrage de charge** : les clés sont réparties de manière uniforme entre les nœuds, par contre la popularité des données ne l'est pas, plusieurs études montrent qu'elle suit une loi de Zipf. La loi de Zipf stipule que la popularité d'une donnée est inversement proportionnelle à son rang, en d'autres termes, une donnée de rang r va être demandée r fois moins qu'une donnée de rang 1. Par conséquent, les nœuds gérant les données populaires sont surchargés.

2.3.2 Recherche de documents par mots

Plusieurs approches ont été étudiées pour implémenter un système de recherche par mots en utilisant les DHT.

Dans ces approches, les index inversés associant à chaque mot les documents dans lesquels il apparaît sont répartis sur les nœuds. Contrairement aux systèmes non structurés présentés précédemment, les index sont répartis par mot. Si un nœud connaît l'ensemble des documents qui contiennent un mot, il ne connaît en revanche pas l'ensemble des mots contenus dans ces documents.

Chaque nœud n est responsable d'un ensemble de mots. Pour une fonction de hachage h donnée, si $h(mot)$ retourne une clé dont n est responsable, alors celui-ci est responsable du mot. Être responsable du mot, c'est connaître les identifiants des documents publiés sur le réseau qui contiennent ce mot. Cette liste est mise à jour lors de la publication ou la suppression d'un document dans le réseau.

Pour publier ou supprimer un document, le nœud contacte les nœuds responsables de chacun des mots du document pour qu'ils mettent à jour leurs index inversés. Pour rechercher les documents qui contiennent le mot m, il suffit d'interroger le nœud responsable de ce mot.

Le problème devient plus difficile lorsqu'on a affaire à des requêtes multi-mots, une méthode naïve serait de demander aux nœuds responsables de chacun des mots de la requête l'ensemble des documents qui contiennent le mot puis d'effectuer l'intersection de ces ensembles. Il est aisé de voir que cette méthode demande des échanges entre les nœuds assez conséquentes et ne peut pas passer l'échelle quand plusieurs millions de documents sont publiés sur le réseau.

Nous allons présenter différentes techniques permettant de réduire la bande passante utilisée lors des recherches multi-mots.

Filtrage des résultats [73] effectue l'intersection des ensembles de documents par filtrage des résultats par les nœuds responsables des mots de la requête. Le nœud demandeur envoie sa requête composée de k mots au nœud A responsable du premier mot de la requête. A envoie sa liste d'identifiants des documents contenant le premier mot au nœud B gérant le deuxième mot de la requête. B effectue un filtrage de cet ensemble en supprimant de la liste les documents qui ne contiennent pas le deuxième mot puis renvoie sa liste filtrée au nœud C responsable du troisième mots. Ce processus est répété jusqu'au nœud Z gérant le

dernier mot de la requête qui après avoir effectué le filtrage renvoie sa liste filtrée au nœud demandeur.

Pour diminuer davantage la bande passante utilisée par l'envoi des ensembles de documents aux différents intervenants, l'auteur propose d'utiliser des filtres de Bloom.

FILTRES DE BLOOM

Les filtres de Bloom [112] sont utilisés pour effectuer des tests d'appartenance à un ensemble de manière économique et rapide. Un ensemble $A = \{a_1, a_2, ..., a_n\}$ est représenté par un vecteur v de m bits. Pour construire ce vecteur, on initialise chacun de ses bits à 0. On choisit k fonctions de hachage indépendantes $h_1, h_2, ...h_k$ qui retournent un nombre entre 0 et m. Pour chaque élément a de A, les bits $h_1(a), h_2(a), ..., h_k(a)$ sont mis à 1. Un bit peut être mis plusieurs fois à 1. Pour savoir si un élément b appartient à l'ensemble A, on teste si les bits à la position $h_1(b), h_2(b), ..., h_k(b)$ sont tous à 1. Si un des bits est à 0, b n'appartient pas à A. Par contre si tous les bits sont à 1, b est dans a avec quand même une probabilité que ce ne soit pas le cas. Ce cas est appelé *faux positif*, cette probabilité $P(k, m, n)$ dépend du nombre de fonctions de hachage utilisées k, de la longueur m du vecteur et du nombre d'éléments n dans l'ensemble :

$$P(k, m, n) = (1 - (1 - \frac{1}{m})^{kn})^k$$

Le filtrage se fait en deux passes. Lors de la première passe, le nœud A gérant le premier mot génère le filtre de Bloom correspondant à l'ensemble des identifiants des documents contenant le premier mot puis l'envoie à B. Celui-ci teste pour chacun des documents convenant au deuxième mot, ceux qui vérifient le filtre de Bloom. Ce nœud génère un filtre de Bloom correspondant aux identifiants de ces documents puis l'envoie à C et ainsi de suite. Le nombre de bits à 1 dans le filtre de Bloom diminue à chaque retransmission. Le nœud Z envoie l'ensemble de ses identifiants qui vérifient le filtre de Bloom au nœud A. La deuxième passe commence, le nœud A filtre l'ensemble reçue en supprimant les documents ne répondant pas au premier mot puis l'envoie au deuxième nœud. Le filtrage se poursuit jusqu'à l'avant dernier nœud qui renvoie la réponse au nœud demandeur.

La deuxième passe est nécessaire à cause des faux positifs, en effet certains identifiants de documents renvoyés par le dernier nœud lors de la première passe peuvent ne pas contenir tous les mots de la requête.

Keyword-Set Search [65] propose de résoudre les problèmes des requêtes multi-mots en étendant la responsabilité des nœuds à des couples de mots, triplet ou k-uplet. Pour simplifier l'explication, on prend $k = 2$. Pour une fonction de hachage h donnée, si pour les mots m_1 et m_2, $h(m_1, m_2)$ retourne une clé dont le nœud est responsable, alors celui-ci, doit connaître les identifiants des documents qui les contiennent tous les deux. Un nœud publie un document en contactant d'une part les nœuds responsables de chaque mot du document pour qu'ils mettent à jour leurs index, et d'autre part les nœuds responsables des différents couples de mots du document. Pour effectuer une recherche sur deux mots, un nœud n'a plus qu'à contacter le nœud responsable du couple de mots. Et pour les recherches s'effectuant sur k mots, il n'a qu'à contacter $k/2$ nœuds. Cette méthode permet de réduire les communications lors des recherches multi-mots mais elle induit un nombre plus important de messages envoyés lors de la publication d'un document ou de sa suppression.

Décomposition en groupes [75] répartit les nœuds en plusieurs groupes. Les membres d'un groupe gèrent les index inversés des documents qui sont publiés au sein de leur groupe. Ils peuvent rechercher par mot les documents qui ont été publiés dans leur groupe en interrogeant les membres de leur groupe responsables des mots de la requête. Pour effectuer une recherche globale, la recherche est propagée à tous les groupes. Le membre interrogé de chaque groupe effectue la recherche au sein de son groupe puis renvoie les résultats obtenus à l'initiateur de la requête.

Comme les membres d'un groupe sont choisis en fonction de leur localité (temps de latence entre eux), les recherches et publications à l'intérieur d'un groupe sont rapides. De plus, chaque groupe gérant un ensemble de documents plus petit, les ensembles de documents échangés entre les nœuds pour effectuer leur intersection sont donc plus petites. Enfin, cela permet de paralléliser la recherche dans les groupes et répartir la charge.

Sélection des mots pertinents dans les documents Dans le système eSearch [78], on considère, pour chaque document, un ensemble de mots pertinents. Ces mots pertinents sont extraits du document en utilisant des techniques issues de la recherche documentaire (*e.g.* tfidf), les k (*e.g.* $k = 5, 10, 20$) mots ayant les poids les plus élevés sont considérés comme les mots pertinents du document.

Lorsqu'un nœud publie un document, il contacte les nœuds qui gèrent les mots pertinents du document pour qu'ils le rajoutent dans leurs index inversés. L'index inversé associe à un mot l'identifiant du document et la liste de tous les mots qu'il contient associé à un poids (*e.g.* tfidf).

Pour effectuer une recherche multi-mots, un nœud envoie la requête aux nœuds responsables de chaque mot m de la requête. Chaque nœud interrogé utilise l'index inversé associé au mot m, dont il est responsable, pour sélectionner l'ensemble des documents qui conviennent à tous les mots de la requête. Puis il classe ces documents par leur similarité cosinus avec la requête. Les documents ayant les meilleures valeurs de similarité sont retournés à l'initiateur de la requête. Le nœud demandeur n'a plus qu'à fusionner les résultats reçus et les classer.

Il est aisé de voir que la recherche n'est pas exhaustive, une recherche sur un mot ne retournera que les documents dans lesquels le mot est considéré comme pertinent et non tous les documents dans lesquels le mot apparaît. Cela ne pose pas de problème comme le soulève les auteurs puisqu'il y a peu de chance que les documents manquants n'eussent été classés dans les premiers résultats. Cependant, pour réduire les chances de rater des documents intéressants, l'expansion de requête est utilisée. La recherche se fait en deux phases. Lors de la première phase, le nœud effectue sa recherche normalement et récupère un petit ensemble (*e.g.* 10) de descriptions de document. Le nœud calcule le poids moyen des mots apparaissant dans ces documents puis sélectionne les k mots ayant les poids moyens les plus élevés pour les ajouter à la requête. La deuxième phase consiste en l'envoi de la requête de recherche enrichie puis à la récupération des résultats. La pertinence des documents est calculée en attribuant moins de poids aux termes rajoutés à la requête.

Les simulations comparent le système avec un moteur de recherche centralisé et mesurent la répartition de charge entre les nœuds. Les simulations montrent que la qualité des résultats dépendent fortement de k le nombre de termes pertinents pour lesquels le document est publié. Pour $k = 20$, les 10 premiers résultats retournés par eSearch sont pratiquement aussi bons que ceux d'un moteur de recherche centralisé. Par contre pour $k = 5$, les résultats sont deux fois moins bons.

Utiliser la popularité des mots Dans l'architecture Keyword Fusion [69], un service de dictionnaire décentralisé est implémenté et permet de connaître les mots qui sont populaires, c'est-à-dire qui apparaissent dans beaucoup de documents. L'idée est que les mots populaires

sont peu importants car ils ne caractérisent pas précisément des documents, donc ont peu de chance d'être tapés. Il est donc inutile de gérer un énorme index inversé pour ces mots. La recherche est effectuée comme décrite en 2.3.2 en n'interrogeant que les nœuds responsables des mots non populaires de la requête. Pour permettre à ces nœuds de sélectionner les documents qui contiennent aussi les mots populaires de la requête, l'index inversé associe à chaque mot, en plus de l'identifiant du document, l'ensemble de ses mots qui sont populaires. Les simulations montrent une meilleure répartition de la taille des index.

2.3.3 Recherche par coordonnée sémantique

pSearch [80] s'appuie sur un espace sémantique où les documents, les requêtes et les nœuds vont avoir une position dans cet espace. L'espace sémantique est défini par le modèle LSI[13] (Latent Semantic Indexing). LSI tente de résoudre les problèmes du modèle VSM comme la synonymie et la polysémie. Dans le cas de la synonymie, les écrivains peuvent utiliser des mots différents pour décrire la même idée. Une personne tapant sa requête dans un moteur de recherche peut taper un autre mot que celui qui apparaît dans le document et ne pas le trouver. Dans le cas de la polysémie, un même mot peut avoir plusieurs sens, l'utilisateur d'un moteur de recherche peut avoir des résultats sur des documents n'ayant pas le sens qu'il attend.

Dans LSI, les documents ne sont pas caractérisés par leurs mots mais par des concepts. Chaque concept représente un ensemble de mots. Par exemple les mots arbre, branche, chêne sont représentés par le même concept. Pour trouver ces concepts, on part d'un ensemble de documents représentatifs et on déduit les concepts en utilisant une décomposition en vecteur propre (SVD).

L'ensemble des documents représentatifs est représenté en VSM par la matrice D suivante :

$$D = \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & & d_m \end{array} \\ \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{bmatrix} \end{array} \begin{array}{l} m_1 \\ m_2 \\ \\ m_n \end{array}$$

avec m_1, ..., m_n les mots et d_1, ..., d_m les documents de l'échantillon et w_{ij} le poids du mot m_i dans le document d_j.

De la décomposition SVD, on en déduit les vecteurs propres qui représentent les concepts et les valeurs propres associées qui représentent l'importance de ces concepts. Seul les concepts les plus importants sont pris en compte. Après renormalisation, on déduit une matrice de conversion permettant de transformer un document en représentation VSM en un document en représentation LSI de dimension moins élevée. Les documents précédents sont représentés en LSI par la matrice D' :

$$D' = \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & & d_m \end{array} \\ \begin{bmatrix} w'_{11} & w'_{12} & \dots & w'_{1m} \\ w'_{21} & w'_{22} & \dots & w'_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w'_{o1} & w'_{o2} & \dots & w'_{om} \end{bmatrix} \end{array} \begin{array}{l} c_1 \\ c_2 \\ \\ c_o \end{array}$$

avec c_1, ... c_o les concepts avec o le nombre total de concepts et w'_{ij}, le poids du concept c_i dans le document d_j.

pSearch utilise la DHT CAN avec une dimension égale au nombre de concepts considérés.

Pour publier un document, le nœud commence par déterminer son vecteur VSM de mots puis le transforme en un vecteur de concepts. Enfin, il le publie sur le nœud qui est responsable de ce vecteur.

Rechercher un document s'effectue de la même manière : la requête de recherche Q qui est un vecteur de mots est transformée en un vecteur de concepts[9]. Cette requête est routée vers le nœud responsable de ce vecteur et est éventuellement retransmis à ses voisins. Les documents ayant une coordonnée proche de celle du vecteur sont renvoyés vers le nœud qui a initié la requête. Ces documents sont classés suivant leur similarité avec la requête.

Les simulations montrent que seul 0.4% à 1% des nœuds doivent être visités pour trouver 95% des réponses.

Cette technique permet de publier et de rechercher des documents de manière peu coûteuse aussi bien en bande passante qu'en nombre de messages utilisés. Contrairement aux techniques présentées précédemment, les documents n'ont pas besoin d'être publiés sur plusieurs nœuds et la recherche multi-mots ne demande pas d'effectuer d'intersections d'ensembles de documents gourmandes en bande passante.

Cependant ce système demande à chaque nœud de connaître les vecteurs propres et valeurs propres pour transformer un vecteur VSM en vecteur LSI. Et dans le cas où ces vecteurs changent, chaque nœud doit déterminer les nouveaux documents dont il est responsable.

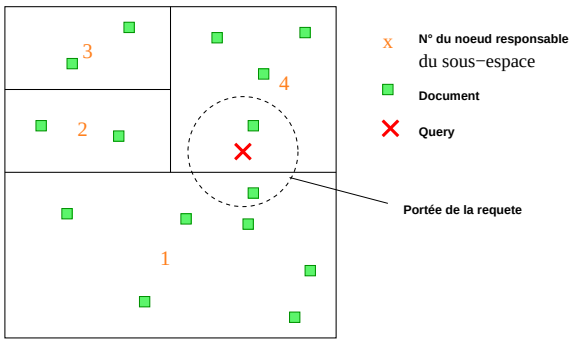


FIG. 2.15: Distribution des documents, nœuds et requêtes dans un tore de dimension 2

2.3.4 Bilan

Différentes solutions ont été proposées pour réduire la taille des index inversés gérés par chaque nœud ainsi que la bande passante utilisée pour les recherches. Cependant, une étude de [67] montre que si on veut indexer un espace d'information aussi grand que le web en répartissant son index inversé sur 60 000 machines, il faudrait au minimum pour chaque machine 1 giga-octet réservé aux index et chaque recherche demanderait en moyenne 530 méga-octets de communication. Des optimisations telles que les deux premières présentées précédemment permettent de réduire sensiblement le coût de communication. Les plus poussées permettent de le réduire d'un facteur 75.

On peut souligner le fait que la publication ou la suppression d'un document demande de mettre à jour les index inversés gérés par les nœuds responsables de chacun des mots du document (typiquement un millier) ce qui entraîne un traffic d'autant plus considérable que l'espace d'information est dynamique.

De plus, un problème inhérent aux DHT demeure, la connexion et la déconnexion des nœuds demandent des transferts de responsabilités entre les nœuds et donc des quantités de

[9] L'algorithme présenté a été simplifié pour plus de clarté

données d'index non négligeables. D'autant plus qu'un nœud se déconnectant peut ne pas avertir ses voisins ni transférer ses index, ce qui implique une réplication des responsabilités et des mécanismes de réparations très coûteuses en bande passante et en messages échangés.

2.4 Conclusion

Nous avons montré que les systèmes de recherche décentralisés permettent d'échanger facilement des informations entre les utilisateurs. Mais ils sont encore loin de pouvoir rivaliser avec les moteurs de recherche pour indexer la masse d'information gigantesque disponible sur le web. Cependant, ils offrent des fonctionnalités différentes, un utilisateur peut publier des fichiers facilement et ces fichiers peuvent être trouvés et téléchargés dès leurs publications. Ces systèmes décentralisés se divisent en deux groupes : les systèmes structurés et les systèmes non structurés. Les systèmes structurés distribuent l'index inversé des mots sur les nœuds et permettent une recherche exhaustive. Par contre, la structuration du réseau, ainsi que la gestion de l'index distribué a un coût en terme de bande passante utilisée et demande une certaine stabilité des nœuds. Dans les systèmes non-structurés ce sont les index des documents qui sont distribués sur les différents nœuds, les recherches ne sont pas exhaustives car une requête ne peut pas visiter l'ensemble des nœuds ayant des réponses. Par contre, l'absence de structuration autorise une transience plus importante des nœuds. Freenet et FASD se situent à la frontière entre les deux types de systèmes. Leur politique de cache et de routage conduit à la spécialisation des nœuds qui deviennent en quelque sorte responsables d'un espace d'identifiants à la manière des systèmes structurés.

Les points forts et défauts de chaque type de systèmes sont résumés dans les tableaux 2.2 et 2.3

Architecture	recherche	fraîcheur des résultats	nœuds transients
centralisée	exhaustive	date d'en moyenne d'un mois	non géré
décentralisée non structurée	données pop. uniquement	à jour	bien géré
décentralisée structurée	exhaustive	à jour	peu adapté

TAB. 2.2: Comparatif qualitatif des résultats suivant l'architecture utilisée

Architecture	Indexation	BP pour les recherches	équi. de charge
centralisée	coûteuse (BP+CPU) pour les serveurs	très faible	
décentralisée non structurée	peu coûteuse car locale	très importante	correct
décentralisée structurée	assez coûteuse (nb. messages)	dépend de la req.	peu adapté

TAB. 2.3: Comparatif des différents coûts suivant l'architecture utilisée

Un aspect souvent négligé dans les systèmes décentralisés est la pertinence des résultats, peu de systèmes gèrent des mécanismes de classement des résultats et aucun ne gère la personnalisation des résultats. C'est ce que nous allons essayer de traiter dans notre système en s'inspirant des mécanismes de spécialisation et des différentes travaux autour du routage sémantique.

Chapitre 3

Maay : un moteur de recherche sémantique par réseau adaptatif de pairs

MAAY [56] est un système de recherche de documents par réseau de pairs. Un réseau de pairs est un réseau où tous ses participants ont les mêmes fonctions et sont simultanément clients et serveurs des autres. Les utilisateurs de MAAY peuvent rechercher, publier et télécharger des documents disponibles sur les machines des autres utilisateurs. MAAY s'inspire des mécanismes de recherche des systèmes pair à pair présentés précédemment pour fournir des résultats à jour, prendre en compte les nœuds transients et améliorer la disponibilité des documents par leurs réplications. A l'instar de Google qui utilise les liens hypertextes dans le réseau des pages web pour définir ses critères de pertinence, MAAY essaye de capturer et d'exploiter les relations complexes entre les utilisateurs, documents et mots pour fournir à ses utilisateurs des réponses qui leur soient personnalisées et pertinentes.

MAAY se base sur deux hypothèses.

La première est que le graphe de cooccurrence des mots, dans lequel deux mots sont liés s'ils apparaissent tous les deux dans une même phrase, est à la fois *small world* et *scale free*. Un graphe est dit *small world* si d'une part la distance moyenne entre deux nœuds du graphe est courte et si d'autre part il y a une probabilité élevée que deux voisins d'un nœud donné soient aussi voisins entre eux. Un graphe est dit *scale free* si la distribution des degrés de ces nœuds suit une loi de puissance, autrement dit, si la probabilité qu'un nœud soit lié à k autres nœuds, est proportionnelle à une puissance de k.

La propriété small world se traduit dans le graphe de cooccurrence d'une part par la distance entre les mots qui est courte, typiquement de quelques unités. Et d'autre part par le fait que si on a deux mots qui sont chacun employés dans des phrases avec un mot donné, alors la probabilité est élevée qu'ils apparaissent ensemble dans une même phrase.

La propriété scale free s'exprime dans le graphe de cooccurrence par d'une part, une majorité de mots qui ne sont employés qu'avec un nombre réduit de mots (*e.g.* couci n'est quasiment employé qu'avec le mot couça) et d'autre part, par un petit nombre de mots qui sont souvent utilisés avec beaucoup d'autres mots (*e.g.* et, le, de)

La deuxième hypothèse est que le graphe des utilisateurs partageant des fichiers en commun est aussi *small world* et *scale free*. Cela veut dire, entre autre, que si deux utilisateurs a et b ont chacun un fichier en commun avec un troisième utilisateur alors il y a une probabilité élevée que l'utilisateur a ait au moins un fichier en commun avec b.

Nous étudierons plus en détail ces graphes dans le chapitre 4.

Les utilisateurs sont caractérisés par le vocabulaire qu'ils utilisent et les documents qu'ils ont écrits ou téléchargés. Ces deux ensembles décrivent le *profil sémantique* de l'utilisateur. Deux utilisateurs ont un profil proche si ils ont plusieurs documents en commun et s'ils ont plusieurs mots de leur vocabulaire en commun.

D'après les deux hypothèses précédemment énoncées, le réseau des utilisateurs liés par profil sémantique proche doit être *small world* et *scale free*. Ainsi un utilisateur pourra trouver les documents qui l'intéresse sur un petit ensemble de nœuds ayant un profil proche du sien. Les recherches peuvent donc se limiter à cet ensemble restreint sans pour autant diminuer la qualité des résultats. De plus, les documents présents sur ces nœuds ont de grande chance d'être très pertinents pour lui.

Dans ce chapitre, nous commencerons par définir le protocole de recherche et de téléchargement utilisé dans le système. Puis, nous verrons la technique d'apprentissage que nous avons conçue pour que les nœuds apprennent localement le profil de leurs voisins. Nous détaillerons les critères utilisés pour choisir les voisins et interroger ceux ayant un profil sémantique proche. Ensuite, nous proposerons des méthodes permettant aux nœuds d'apprendre les profil des documents. Finalement, nous décrirons les critères de classement de documents que nous avons défini pour offrir aux utilisateurs des résultats qui leur soient pertinents et personnalisés.

3.1 Modèle

MAAY est un système de recherche décentralisé en plein texte, c'est-à-dire qu'un document peut être recherché par les mots qui sont présents dans son contenu textuel, et pas uniquement par les mots apparaissant dans ses méta-données.

Un nœud MAAY est un processus ayant des capacités de communication et qui est associé à un utilisateur. Les nœuds peuvent chercher, télécharger et publier des documents. Un nœud b est voisin de a quand le nœud a connaît l'identifiant du nœud b. On note $\mathcal{V}(a)$ l'ensemble des voisins de a. Le nœud a peut envoyer des messages aux nœuds qui sont dans $\mathcal{V}(a)$.

Un document est composé de plusieurs mots et est identifié de manière unique grâce à une clé obtenue par une fonction de hachage (*e.g.* SHA-1) sur son contenu. Un nœud stocke les documents qu'il publie et télécharge, et les rend disponibles aux autres. On note $\mathcal{D}(a)$ l'ensemble des documents partagés par le nœud a et $\mathcal{DM}(d)$ l'ensemble des mots du document d. Ces documents sont indexés localement par le nœud et conviennent à une requête s'ils contiennent tous les mots de la requête.

Chaque nœud maintient dans une base de données des informations sur ses voisins et les documents dont il a connaissance et les met régulièrement à jour. Chaque réception de messages apportent un ensemble d'informations qui sont pris en compte et sauvegardé dans le base de données. Bien que l'espace mémoire ne soit pas un problème crucial, nous limitons les informations sauvegardées par les nœuds de telle sorte que seules les informations pertinentes pour l'utilisateur soient sauvegardées.

3.2 Protocole

Terminologies On appelle *initiateur de la requête* le nœud qui a créé la requête de recherche. Une requête de recherche peut être reçue par un nœud puis retransmise à un autre nœud. Du point de vue du nœud qui reçoit la requête, le nœud qui lui a envoyé ou retransmis la requête s'appelle *l'expéditeur de la requête* ou *émetteur de la requête*. Un *fournisseur d'un document* d est un nœud qui possède le document d et qui le rend disponible aux autres.

Structure des messages Le système MAAY utilise quatre types de message dédié à la recherche et au téléchargement :

- la **requête de recherche** contient l'identifiant du nœud qui a émis la requête, l'identifiant de la requête (*query_id*), la question composée de mots, le nombre de résultats attendus EHC (Expected Hit Count) et des paramètres de propagation ;
- la **réponse de recherche** est composée de l'identifiant du nœud qui a retransmis la réponse, l'identifiant de la requête et un ensemble de descriptions de document convenant à la requête. Chaque description contient un ensemble de méta-données décrivant le document (titre, taille, ...), un ensemble de scores pour décrire son profil (nous décrirons plus tard ce concept) et un ensemble de nœuds pouvant fournir le document ;
- la **requête de téléchargement** contient l'identifiant de la requête, l'identifiant du document à télécharger et la question utilisée pour trouver ce document ;
- la **réponse de téléchargement** est composée de l'identifiant de la requête, d'une description du document contenant d'autres fournisseurs potentiels du document et du contenu du document.

L'identifiant de la requête (*query_id*) permet d'identifier de manière unique une recherche ou un téléchargement. Les messages (requêtes et réponses) issues d'une même requête ont le même identifiant de requête. Cet identifiant est généré par le nœud qui a initié la requête.

La structure des messages est détaillée dans la table 3.1.

MESSAGES :

```
REQUETE DE RECHERCHE =
       EN-TETE,
       fnc,     nombre de nœuds auxquels la requête est retransmise
       ehc,     nombre de résultats attendus
       ttl,     nombre de retransmissions restantes
       question
REPONSE DE RECHERCHE =
       EN-TETE, HIT1, HIT2, ...
REQUETE DE TELECHARGEMENT =
       EN-TETE,
       id du document,
       QUESTION        question utilisée pour trouver le document
REPONSE DE TELECHARGEMENT =
       EN-TETE,
       HIT,                méta-données du document
       contenu du document
```

SOUS-PARTIES :

```
EN-TETE =
       identifiant de l'expéditeur du message, identifiant de la requête
QUESTION =
       mot1, mot2, ...
HIT =                                   description d'un résultat
       id du document,
       méta-données du document,        titre, taille, date, ...
       SCORE_1, SCORE_2, ...            score des mots
       FOURNISSEUR1, FOURNISSEUR2, ...  où récupérer le document
SCORE =
       mot, pertinence, popularité,     du mot dans le document
       extrait                          du document contenant le mot
FOURNISSEUR =              où récupérer le document
       id du nœud
       date de possession  quand il a affirmé avoir le document
```

FIG. 3.1: Description des messages utilisés dans MAAY

Paramètres de propagation de la requête Pour éviter les problèmes dus à la recherche par inondation, nous utilisons deux paramètres pour contrôler la propagation de la requête. Le `TTL` (Time-To-Live) compte le nombre de retransmissions restantes du message et est diminué de 1 à chaque retransmission. Quand celui-ci vaut 0, le message n'est plus retransmis. Le `FNC` (Forwarding Node Count) décrit le nombre de voisins vers lesquels la requête est retransmise. Il est réduit de moitié à chaque retransmission et est arrondi à l'entier supérieur.

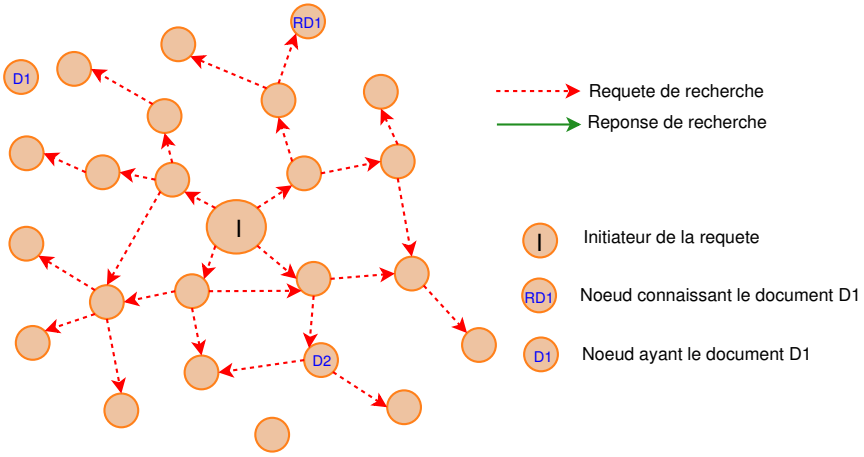


FIG. 3.2: Propagation d'une requête de recherche

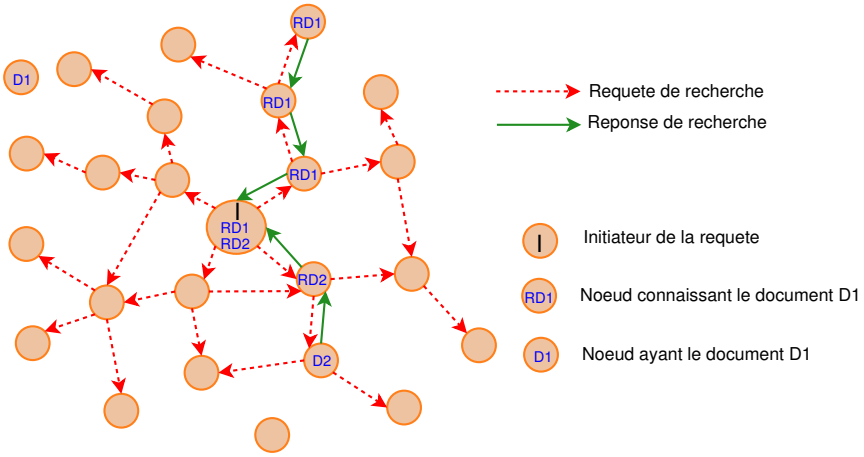


FIG. 3.3: Retour des réponses vers l'initiateur de la requête

Exemple de recherche Sur la figure 3.2 est représentée la propagation d'une requête de recherche. Le nœud I est l'initiateur de la recherche. Deux documents $D1$ et $D2$ qui conviennent à la requête de recherche se trouvent sur deux nœuds. Ces deux documents sont indexés localement par le nœud sur lequel ils se trouvent. Un nœud possède une référence $RD1$ du document $D1$ et ne connaît que des informations partielles sur le document (titre, nœuds qui ont le document, quelques mots présents dans le document, ...). Ce nœud connaît le document grâce à une réponse de recherche reçue antérieurement.

Chaque nœud maintient localement un ensemble de profils sémantiques de ses voisins. Pour effectuer une recherche, le nœud I sélectionne les voisins dont le profil présente des similarités avec l'émetteur de la requête (ici, lui même) et qui ont manifesté leurs intérêts

pour les mots de la requête. Puis il leur envoie une requête de recherche. Les nœuds recevant la requête de recherche la retransmettent à leurs voisins choisis suivant les mêmes critères, et ainsi de suite, ... Des paramètres sont associés aux requêtes de recherche pour contrôler leur propagation et limiter leur nombre de retransmissions.

La figure 3.3 décrit la manière dont les réponses sont retournées à l'initiateur de la recherche. Les nœuds contactés et possédant pour l'un le document $D2$ et l'autre une référence sur le document $D1$ vont renvoyer leur réponse (description du document + identifiants des nœuds possédant le document) au nœud qui leur a retransmis la requête. Les réponses remontent de proche en proche dans le sens inverse de la requête jusqu'à l'initiateur de la recherche. Les nœuds intermédiaires mettent dans leur cache les réponses reçues pour pouvoir répondre à des requêtes ultérieures. En utilisant les identifiants des fournisseurs présents dans les réponses de recherche, le nœud peut directement les interroger pour télécharger le document.

3.2.1 Protocole de recherche

Chaque nœud maintient à jour une table de recherches qui lui permet de garder le contexte des recherches qu'il a émises (ou retransmises) : le buffer BRP des résultats les plus pertinents pour la question Q et son émetteur e, l'identifiant du nœud e, la question et le nombre de résultats attendus.

Réception d'une requête de recherche L'algorithme 1 décrit les opérations exécutées par un nœud a lors de la réception d'une requête de recherche sur la question Q transmise par le nœud e. La requête est aussi décrite par son identifiant qid et par ses paramètres de propagation ttl et fnc.

Lors de la réception d'une requête de recherche, le nœud commence par regarder la valeur du ttl de la requête pour savoir s'il doit retransmettre la requête. Si le ttl n'est pas nul alors le nœud a choisit un ensemble de fnc voisins suivant leur similarité avec le nœud émetteur e et suivant la requête Q (le choix des voisins sera détaillé dans la section 3.3.2). Puis il leur retransmet la requête en modifiant les paramètres de propagation ttl et fnc.

Ensuite, le nœud a cherche localement parmi ses documents ceux qui sont les plus pertinents pour la question Q et le nœud e. Le choix des documents sera décrit plus en détail dans la section 3.4.2. Les `EHC` meilleurs documents sont sélectionnés et stockés dans le buffer BRP, puis leurs descriptions sont renvoyées au nœud e dans un message de réponse contenant les méta-données des documents, des scores décrivant leur profil et des fournisseurs de ces documents.

Le contexte de la requête : le buffer BRP, l'émetteur e de la requête, l'identifiant de l'émetteur e, la question Q et le nombre de résultats attendus ehc est sauvegardé dans la table de recherches.

Réception d'une réponse de recherche L'algorithme 2 décrit les opérations exécutées par un nœud a lors de la réception d'une réponse de recherche transmise par le nœud r.

En utilisant sa table de recherches, le nœud a commence par récupérer le contexte associé à cette recherche : le buffer BRP, l'émetteur e de la requête, la question Q et le nombre de résultats attendus ehc.

Puis il fusionne les résultats présents dans le message de réponse avec le buffer BRP. La fusion consiste à unir les deux ensembles puis à ne garder que les documents les plus pertinents pour la question Q et pour l'émetteur e de la requête. Le résultat est mis dans le buffer BRP. Comme le nœud a a retransmis la requête à plusieurs voisins, il peut recevoir plusieurs réponses. Cependant, il ne doit pas retransmettre toutes les réponses reçues mais

Algorithme 1 : Réception d'une `requête de recherche` par le nœud a

Fonctions :
$choix\_voisins(Q, e, n)$: renvoie n voisins choisis suivant leur similarité avec le nœud e et leurs intérêts pour Q ;
$choix\_docs(Q, e, n)$: renvoie n documents les plus pertinents pour Q et pour le nœud e.
Paramètres :
e : le nœud émetteur de la requête ;
qid : l'identifiant de la requête (Query IDentifier).
ttl : nombre de retransmissions restantes de la requête (Time-To-Live) ;
fnc : nombre de nœuds vers lesquelles retransmettre la requête (Forwarding Node Count) ;
ehc : nombre de résultats attendus (Expected Hit Count) ;
Q : l'ensemble des mots de la requête ;
Données :
$recherche[]$: table associant à un identifiant de requête, son buffer des meilleurs résultats, le nœud émetteur, la question et le nombre de résultats attendus.
Actions :
Si $ttl > 0$ **alors**
$F \leftarrow choix\_voisins(Q, e, fnc)$
$msg \leftarrow$ "`requête de recherche`, $a, qid, ttl - 1, fnc/2, ehc, Q$"
$\forall f \in F$: **envoie** msg **à** f
$recherche[qid] \leftarrow R, e, Q, ehc$
$msg \leftarrow$ "`reponse de recherche`, a, qid, R"
envoie msg **à** e
$R \leftarrow choix\_docs(Q, e, ehc)$ `(enregistrement du contexte de la recherche)`

seulement compléter les réponses qu'il a déjà renvoyées. Pour cela, le nœud a retransmet seulement les nouveaux résultats reçus qui apparaissent dans le buffer des résultats les plus pertinents BRP.

Le nouveau contexte de la requête est sauvegardé dans la table de recherches pour tenir compte des résultats qui ont déjà été envoyé à l'émetteur.

Les réponses d'une recherche apparaissent à l'utilisateur comme un ensemble trié de descriptions de document avec des liens vers les fournisseurs pour télécharger le document.

3.2.2 Protocole de téléchargement

Pour télécharger un document, le nœud a envoie à un fournisseur du document (connu grâce aux réponses de recherche), une requête de téléchargement composée de l'identifiant du document et des mots qui ont été utilisés pour trouver le document. Le nœud interrogé répond au nœud q en lui envoyant d'une part, une liste de fournisseurs potentiels du document et d'autre part, le contenu du fichier si il a.

Après avoir reçu la réponse de téléchargement, le nœud a met a jour sa liste de fournisseurs associés au document. Si le contenu du fichier est présent dans la réponse, le nœud a sauvegarde le fichier dans son cache de documents, l'indexe et le rend disponible aux autres. Sinon, le nœud a essaye de télécharger le document en interrogeant un autre fournisseur. Le processus est réitéré jusqu'à ce que le nœud arrive à télécharger le document.

Algorithme 2 : Réception d'une `réponse de recherche` envoyé par le nœud r et reçu par a

Fonctions :
$fusionne\_resultats(R, R_n, e, Q, k)$: fusionne les résultats R_n avec ceux de R et retourne les k résultats les plus pertinents pour le nœud e et la requête Q

Paramètres :
r : l'émetteur de la réponse.
qid : l'identifiant de la requête (Query IDentifier).
R_r : résultats envoyés par le nœud r

Données :
$recherche[]$: table associant à un identifiant de requête, son buffer des meilleurs résultats, le nœud émetteur, la question et le nombre de résultats attendus.

Actions :
$R, e, Q, ehc \leftarrow recherche[qid]$ `(récupération du contexte de la recherche)`
$fusionne\_resultats(R, R_r, e, Q, ehc)$
$msg \leftarrow$ "`reponse de recherche`, $a, qid, R_r \cap R$"
envoie msg **à** e
$recherche[qid] \leftarrow R, e, Q, ehc$ `(enregistrement du contexte de la recherche)`

3.3 Profil sémantique des voisins

Un *profil sémantique de voisin* est décrit par un ensemble de mots valués qui décrivent ses centres d'intérêts. Chaque nœud apprend localement le profil des voisins en utilisant les informations contenues dans les messages qu'il reçoit. En utilisant ces profils, le nœud peut interroger les voisins ayant un profil similaire à l'émetteur de la requête (qui peut être le nœud lui même) et qui sont les plus à même de lui fournir des documents qui lui soient pertinents.

Nous allons décrire dans cette section comment les nœuds apprennent les profils de leurs voisins, et comment ils les utilisent pour router les requêtes de recherche.

3.3.1 Apprentissage du profil des voisins

Un nœud connaît de nouveaux voisins soit en recevant un message de leur part, soit en recevant un message de réponse où ils apparaissent comme fournisseurs de documents.

En utilisant ces messages, les nœuds vont apprendre peu à peu le profil sémantique de leurs voisins. Un nœud n est au courant qu'un de ses voisins v a manifesté son intérêt pour un mot m :

- en recevant une requête de recherche sur m émis (ou retransmis) par le nœud v ;
- en recevant une réponse de recherche sur m émis (ou retransmis) par le nœud v ;
- en recevant une requête de téléchargement associée au mot m provenant du nœud v ;
- en apparaissant comme fournisseur d'un document dans une réponse à une recherche sur le mot m.

On note $EXPR_n(v, m)$ le total pondéré des manifestations d'intérêt du voisin v pour le mot m reçues par le nœud n.

Remarque : Lorsqu'un utilisateur effectue une recherche, il envoie *via* son interface graphique une requête de recherche à son nœud local. Le nœud connaît ainsi le profil de son utilisateur.

Ces valeurs sont locales à chaque nœud et deux nœuds peuvent avoir des informations différentes pour le même voisin.

On définit le profil d'un voisin par un ensemble de mots associés aux valeurs de *spécialisation* et d'*expertise*.

SPÉCIALISATION : La spécialisation $SP_a(v, m)$ d'un voisin v pour un mot m représente sa part d'intérêt pour le mot m comparée aux autres mots du point de vue du nœud a :

$$SP_a(v, m) = \frac{EXPR_a(v, m)}{\sum_{m' \in \mathcal{M}(a)} EXPR_a(v, m')}$$

avec $\mathcal{M}(a)$ l'ensemble des mots connus par le nœud a et $EXPR_a(v, m)$ le total pondéré des manifestations d'intérêt du voisin v pour le mot m et reçues par a.

Un voisin de a ayant une spécialisation proche de 1 n'aura pratiquement manifesté son intérêt au nœud a que pour un seul mot.

EXPERTISE : L'expertise $XP_a(v, m)$ d'un voisin v décrit sa part d'intérêt pour un mot m comparée aux autre voisins du point de vue du nœud a :

$$XP_a(v, m) = \frac{EXPR_a(v, m)}{\sum_{v' \in \mathcal{V}(a)} EXPR_a(v', m)}$$

avec $\mathcal{V}(a)$ l'ensemble des voisins du nœud a et $EXPR_a(v, m)$ le total pondéré des manifestations d'intérêt du voisin v pour le mot m et reçues par a.

Ainsi si un voisin d'un nœud a a une valeur d'expertise proche de 1 pour le mot m, cela signifie qu'il est l'un des seuls voisins de a a avoir manifesté son intérêt pour le mot m.

Un exemple des informations sauvegardées par le nœud sur un de ses voisins est présenté dans la figure 3.4. Le profil du voisin montre que celui-ci a surtout manifesté son intérêt pour le mot *p2p* (spécialisation de 0.4) et qu'il est son voisin le plus intéressé par le mot *routing* (expertise de 0.7).

ID du nœud : 92344db76d908050f70217efb20b4bbe90d7071c
IP/Port : 192.33.178.30 :6446

Profil du nœud

Mot	Spécialisation	Expertise
p2p	0.4	0.2
search	0.1	0.1
engine	0.1	0.3
routing	0.05	0.7
...	...	...

FIG. 3.4: Exemple de profil d'un voisin

Un nœud a une plus grande probabilité de trouver des documents intéressants chez un nœud ayant un profil sémantique proche. Ainsi, en effectuant une recherche sur le mot `java`, un informaticien va avoir plus de chance de trouver des documents traitant du langage de programmation en interrogeant un autre informaticien qu'une personne quelconque (qui pourrait

être intéressée par la danse française ou par l'île indonésienne). Pour cela nous introduisons la mesure d'affinité.

AFFINITÉ : La fonction $affinite_a(b, c)$ mesure la similarité de profil entre deux voisins b et c du point de vue du nœud a. Elle se calcule de la manière suivante :

$$affinite_a(b,c) = \frac{\sum_{m \in \mathcal{M}(a)} SP_a(b,m).SP_a(c,m)}{\sqrt{\sum_{m \in \mathcal{M}(a)} SP_a^2(b,m). \sum_{m \in \mathcal{M}(a)} SP_a^2(c,m)}}$$

avec $SP_a(b, m)$ la spécialisation du voisin b pour le mot m et $\mathcal{M}(A)$ l'ensemble des mots connus par le nœud A.

Deux voisins ayant exprimé des intérêts similaires et dans les mêmes proportions auront une valeur d'affinité proche de 1.

3.3.2 Routage des requêtes de recherche

L'idée est de choisir parmi les nombreuses connaissances du nœud, un petit nombre de voisins présentant des similarités avec l'expéditeur de la requête et qui ont manifesté leurs intérêts pour les mots de la requête.

Le nœud choisit parmi sa liste de voisins ceux auxquels il va retransmettre la requête en privilégiant :

- la **spécialisation** : un nœud affichant exclusivement ses intérêts pour les mots de la requête ;
- l'**expertise** : un nœud exhibant plus que les autres nœuds son intérêt pour les mots de la requête ;
- l'**affinité** : un nœud ayant un profil sémantique proche de celui qui a émis la requête.

Cependant, lorsqu'un nœud arrive dans le système ou change brutalement d'intérêt, il peut ne pas connaître de voisins intéressés par les mots de la requête. Une solution est d'interroger les voisins intéressés par des mots proches des mots de la requête. Nous utilisons pour cela, la distance de Levenshtein qui correspond au nombre d'insertions, de suppressions et de remplacements nécessaires pour transformer un mot en un autre (d'autres distances peuvent être utilisées, voir encadré). Par conséquent, la distance entre un mot au singulier et sa forme au pluriel est courte, de même qu'entre deux mots ayant la même racine étymologique. A la manière de Freenet, les nœuds vont se spécialiser pour les mots qui sont proches de ceux qui les intéressent le plus. Ils vont ainsi pouvoir retransmettre une requête portant sur ces mots proches à des nœuds intéressés par ces mots (mettre en relations ces nœuds) et/ou pouvoir directement répondre à ces nœuds en utilisant les réponses de recherche qu'ils ont reçus précédemment.

DISTANCE / SIMILARITÉ ENTRE MOTS

Distance de Hamming
La distance de Hamming est définie seulement pour deux chaînes de caractères de même taille. Elle est égale au nombre de positions pour lesquelles les éléments correspondants diffèrent. Dit d'une autre façon, elle mesure le nombre de substitutions nécessaires pour transformer l'une en l'autre.

Coefficient de Dice
Un n-gramme est une sous-séquence de n-lettres d'une chaîne de caractères après avoir supprimé tous les espaces. Les trigrammes (3-grammes) du mot `Gnutella` sont `Gnu`, `nut`,

`ute`, `tel`, `ell`, `lla`. Le coefficient de Dice mesure la proportion de n-grammes que deux mots X et Y ont en commun :

$$\frac{2 * |C|}{x + y}$$

avec C le nombre de n-grammes en commun entre les mots X et Y, x et y le nombre de n-grammes dans le mot X et Y. Plus le coefficient de Dice est proche de 0, moins les mots sont similaires et plus il est proche de 1 plus les mots sont semblables.

Distance de Levenshtein
La distance de Levenshtein a été proposée par le scientifique russe Vladimir Levenshtein en 1965. Elle est aussi appelée la distance d'édition et mesure le nombre minimum d'insertions, suppressions et de remplacements de lettre pour transformer un mot en un autre. Elle peut être considérée comme une généralisation de la distance de Hamming. La distance entre deux mots identiques est de 0. Et les mots à distance 1 du mot `root` sont `roots`, `foot`, `riot`, `roost`, ... Cette distance est utilisée pour la correction orthographique automatique, la reconnaissance vocale, l'analyse d'ADN, la détection de plagiat, ...

Plus grande sous-chaîne commune (LCS : Longest Common Substrings)
LCS décrit la sous-chaîne commune la plus grande entre deux chaînes de caractères. Une sous-chaîne d'une chaîne X est une chaîne qui peut être obtenue par suppression de lettres de X. Ainsi le mot `souris` a par exemple pour sous-chaînes `ris`, `suis`, `oui` ou encore la chaîne vide. La plus grande sous-chaîne commune entre les mots `algorithme` et `élire` est `lie` ou `lre`. Plus la longueur de la sous-chaîne commune est grande, plus les deux chaînes sont similaires.

Nous pouvons maintenir définir le score de routage.

SCORE DE ROUTAGE : Le *score de routage* $SR_a(e, v, m)$ calculé par le nœud a pour router une requête portant sur le mot m provenant d'un expéditeur e vers un voisin v est calculé de la manière suivante :

$$SR_a(e, v, m) = affinite_a(e, v). \sum_{m' \in \mathcal{M}(a)} \frac{SP_a(v, m').XP_a(v, m')}{2^{distance(m,m')}}$$

avec $distance(m, m')$ la distance de Levenshtein entre les mots m et m', $affinite_a(e, v)$ l'affinité entre le profil du voisin e et celui du voisin v, $SP_a(v, m)$ la spécialisation du voisin v pour le mot m et $XP_a(v, m)$ l'expertise du voisin v pour le mot m du point de vue du nœud a.

Avant de retransmettre une requête de recherche, le nœud calcule les scores de routage de chacun de ses voisins puis retransmet la requête aux `EHC` voisins ayant les meilleurs scores de routage.

Cependant, en utilisant cette technique, des voisins risquent de ne jamais être interrogés. Or ils peuvent posséder des contenus et des voisins intéressants. Pour remédier à ce problème, une partie des voisins auxquels la requête est retransmise est choisie aléatoirement parmi la liste des voisins.

3.3.3 Propagation de la requête de recherche

Les voisins qui sont interrogés par l'initiateur de la requête ont plus de probabilité d'être pertinents pour lui que ceux interrogés par ses voisins car les profils des voisins des voisins

ont moins de chance d'être similaires à celui qui a initié la requête. Par conséquent, plus une requête s'éloigne de son initiateur, moins elle a de chance d'atteindre un nœud intéressant. Mais plus elle a plus de chance d'atteindre des nouveaux nœuds qui peuvent se révéler avoir un profil proche de l'initiateur de la requête. Pour tenir compte de ces deux facteurs, un nœud recevant une requête de recherche la retransmet à peu de voisins s'il est éloigné de l'initiateur et à beaucoup de voisins s'il est proche de celui-ci.

Pour cela, on utilise deux paramètres qui vont contrôler la propagation de la requête de recherche :

- *TTL* contrôle le nombre de retransmissions. Il est décrémenté de 1 à chaque retransmission. Quand sa valeur vaut 0, la requête n'est plus retransmise.
- *FNC* contrôle le nombre de voisins auxquels la requête est retransmise. Il est divisé par 2 et arrondi à l'entier supérieur à chaque retransmission.

La figure 3.5 montre comment la requête est propagée dans le réseau. La requête de recherche est initiée avec un `TTL` initial de 4 et un `FNC` initial de 4. L'initiateur de la requête commence par envoyer la requête à 4 de ses voisins ($ttl = 3$ et $fnc = 2$). Ces voisins la retransmettent à 2 de leurs voisins ($ttl = 2$ et $fnc = 1$). Ces voisins l'envoient à leur tour à 1 de leurs voisins ($ttl = 1$ et $fnc = 1$). Enfin, ce voisin la retransmet à 1 seul de ses voisins ($ttl = 0$ et $fnc = 1$). Le `TTL` de la requête étant nul, la requête n'est plus retransmise. Au total, la requête a été envoyée à 28 nœuds.

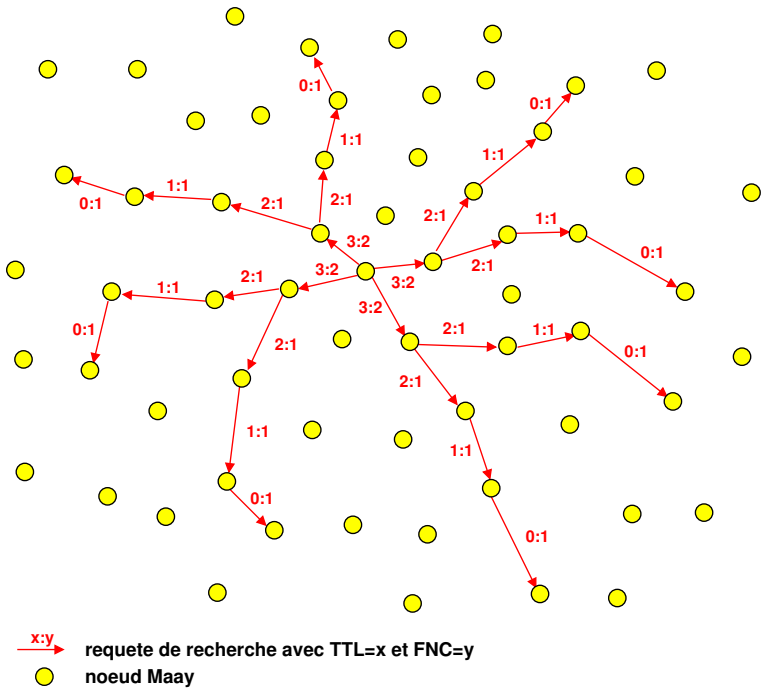


FIG. 3.5: Propagation d'une requête de recherche dans MAAY

3.3.4 Vieillissement

On introduit le mécanisme de vieillissement des manifestations d'intérêt des voisins pour donner une plus grande importance à celles qui ont eu lieu récemment et moins d'importance à celles qui ont eu lieu il y a longtemps. L'idée est d'une part de tenir compte des changements d'intérêt des voisins. Et d'autre part, de continuellement mettre à l'épreuve les nœuds qui doivent sans cesse prouver leurs expertises. Un nœud présent dans le réseau depuis longtemps

et ne manifestant plus son intérêt pour un mot m pourra être dépassé en expertise par un nouveau nœud plus actif.

Le vieillissement est effectué à intervalle régulier en multipliant pour chacun des voisins leurs manifestations d'intérêt par un coefficient de vieillissement (*e.g.* 0.9).

Si le nombre de profils de voisin devient trop important, le nœud peut évincer les profils des voisins avec lesquels il possède peu d'affinité.

3.4 Profil des documents

Un nœud peut apprendre les profil des documents à partir des messages reçus. En utilisant ces profils, le nœud peut sélectionner les documents ayant un profil en adéquation avec l'émetteur de la requête de telle manière à lui présenter les documents qui lui soient le plus pertinents.

Nous allons décrire dans cette section comment les nœuds apprennent les profils de leurs documents, et comment ils les classent en fonction du profil de l'émetteur de la requête.

3.4.1 Apprentissage du profil des documents

Pour apprendre le profil des documents, le nœud peut utiliser les informations contenues dans :

- les requêtes de téléchargement : en utilisant les mots (utilisés pour trouver le document) associés à la requête de téléchargement, le nœud peut connaître les mots qui caractérisent le document ;
- les réponses de recherche : en utilisant les scores des documents dans les résultats, le nœud connaît le profil partiel du document.

Les nœuds ayant localement un document d stocké dans leur cache (document publié ou téléchargé) l'ont indexé et ont donc son profil complet. Ils mettent à jour ce profil en utilisant les requête de téléchargement.

Au contraire, si le document d n'est pas stocké dans son cache, le nœud ne peut apprendre le profil de celui-ci qu'à partir des profils partiels contenus dans les réponses de recherche.

Apprentissage du profil des documents locaux à partir des requêtes de téléchargement
Une requête de téléchargement sur un document d pour le mot m peut être considéré comme deux votes : un pour la pertinence du mot m pour le document d, et l'autre pour la popularité du document d parmi les documents qui conviennent au mot m.

On note $TC_a(d, m)$ le nombre de téléchargements reçus par le nœud a pour le document d associé au mot m. Le profil d'un document est composé d'un ensemble de mots valués par la *pertinence* et la *popularité*.

La valeur de pertinence d'un mot m pour un document d décrit si le mot m caractérise le document d. On peut le définir comme le ratio du nombre de téléchargements du document d associé au mot m sur le nombre total de téléchargement du document d.

Cependant, cette mesure est imprécise si le nœud a reçu peu de téléchargements pour ce document. Pour tenir compte de cette incertitude de mesure, on utilise la borne de Hoeffding qui va nous permettre d'en définir une valeur pessimiste.

On va donc utiliser les mesures pessimistes de la pertinence et de la popularité.

PERTINENCE : La *pertinence* $PTN_a(d,m)$ d'un mot m pour un document d est calculée par le nœud a de la manière suivante :

$$PTN_a(d,m) = \frac{TC_a(d,m)}{\sum_{m' \in \mathcal{M}(a)} TC_a(d,m')} - \sqrt{\frac{ln(\frac{1-c}{2})}{2. \sum_{m' \in \mathcal{M}(a)} TC_a(d,m')}}$$

avec c la valeur de confiance dans la précision de la mesure (*e.g.* $c = 0.95$), $\mathcal{M}(a)$ l'ensemble des mots connus par le nœud a, $TC_a(d,m)$ le nombre de téléchargements reçus par le nœud a pour le document d et associés au mot m.

Ainsi, plus un mot m aura une pertinence élevée dans le document plus il est représentatif du document.

POPULARITÉ : La *popularité* $POP_a(d,m)$ d'un mot m pour un document d est mesuré par le nœud a de la manière suivante :

$$POP_a(d,m) = \frac{TC_a(d,m)}{\sum_{d' \in \mathcal{D}(a)} TC_a(d',m)} - \sqrt{\frac{ln(\frac{1-c}{2})}{2. \sum_{d' \in \mathcal{D}(a)} TC_a(d',m)}}$$

avec c la valeur de confiance dans la précision de la mesure (*e.g.* $c = 0.95$), $\mathcal{D}(a)$ l'ensemble des documents locaux (publiés ou téléchargés) du nœud A et $TC_a(d,m)$ le nombre de téléchargements reçus par le nœud a pour le document d et associés au mot m.

Plus un document aura une popularité élevée pour le mot m, plus il est représentatif du mot m.

Un exemple des informations associées à chaque document est représenté dans la figure 3.6. Ce profil montre que les mots *gnutella* et *protocol* décrivent bien le document (valeur de pertinence de 0.4 et 0.3). Le document est une bonne réponse à donner pour une requête sur le mot *specification* (valeur de popularité de 0.4).

ID du document : a80bbc6526c0efaafbe24f25128e8704b937b9f2
Titre : The Gnutella Protocol Specification v0.4
Taille : 44425 bytes

Profil du document

Mot	Pertinence	Popularité
gnutella	0.4	0.2
protocol	0.3	0.1
query	0.1	0.1
specification	0.1	0.4
routing	0.05	0.1
...	...	...

FIG. 3.6: Exemple de profil d'un document

Apprentissage du profil partiel des documents non locaux à partir des réponses de recherche En recevant des réponses de recherche, le nœud va connaître des nouveaux documents. Il met dans son cache les profils partiels des documents décrits dans les réponses de recherche pour pouvoir répondre à des requêtes ultérieures.

Le profil partiel d'un document est composé d'un ensemble de mots associés aux scores de pertinence et de popularité. Cet ensemble de mots comprend les mots utilisés dans la recherche et les mots les plus pertinents du document (sélectionnés par celui qui a retransmis la réponse).

Pour personnaliser les résultats à l'émetteur de la requête, nous introduisons la mesure d'*adéquation*.

ADÉQUATION : L'*adéquation* $ADQ_a(d,n)$ entre un document d et un voisin v est calculée par le nœud a de la manière suivante :

$$ADQ_a(d,n) = \frac{\sum_{m \in \mathcal{M}(a)} PTN_a(d,m).SP_a(v,m)}{\sqrt{\sum_{m \in \mathcal{M}(a)} PTN_a^2(d,m). \sum_{m \in \mathcal{M}(a)} SP_a^2(v,m)}}$$

avec $\mathcal{M}(a)$ l'ensemble des mots connus par le nœud a, $PTN_a(d,m)$ la pertinence du document d pour le mot m et $SP_a(v,m)$ la spécialisation du voisin v pour le mot m.

Plus cette valeur est proche de 1, plus le profil du document et le profil du voisin sont similaires.

3.4.2 Classement des documents

Après avoir sélectionné parmi les profils des documents locaux et non locaux ceux qui répondent à la requête, le nœud les classent en calculant pour chacun d'eux un score de classement (défini ci-dessous) pour déterminer ceux qui conviennent le mieux à l'émetteur de la requête.

Le classement des documents se base sur 3 critères :
- la pertinence des mots de la requête dans le document ;
- la popularité du document pour les mots de la requête ;
- l'adéquation entre le document et le profil de l'expéditeur de la requête (ce critère permet de personnaliser les résultats à l'émetteur de la requête).

SCORE DE CLASSEMENT : Le score de classement $SC_a(e,d,m)$ du document d pour une requête de recherche portant sur le mot m et pour son voisin e est calculée par le nœud a de la manière suivante :

$$SC_a(e,d,m) = ADQ_a(d,e).PTN_a(d,m).POP_a(d,m)$$

avec $ADQ_a(d,e)$ l'adéquation entre le document d et le voisin e, $PTN_a(d,m)$ la pertinence du mot m pour le document d et $POP_a(d,m)$ la popularité du document d pour le mot m du point de vue du nœud a.

3.4.3 Vieillissement

Les nouveaux documents publiés vont avoir une popularité initiale faible et vont difficilement apparaître dans la liste des meilleurs résultats. Ce qui va les empêcher d'être vu et donc téléchargé par les utilisateurs. Nous avons pour cela implémenté un mécanisme de vieillissement. Périodiquement, la valeur TC associée à chaque mot et à chaque document est multipliée par un facteur de vieillissement (*e.g.* 0.9). Ainsi la popularité des documents est continuellement mise à l'épreuve, ce qui permet aux anciens documents devenus moins intéressants de laisser leur place aux nouveaux documents.

Si le nombre de profils de document devient trop important, ou les documents stockés dans le cache prennent trop de place, le nœud peut choisir de supprimer les profils ou les documents en privilégiant ceux avec lesquels il a le moins d'adéquation.

3.5 Conclusion

Nous avons proposé un système de recherche par réseau de pairs où l'indexation des données et la recherche s'effectuaient de manière collaborative. Nous avons défini les mécanismes d'apprentissage permettant aux nœuds de connaître les profils de leurs voisins et des documents. Puis nous avons proposé des critères de routage pour que les nœuds s'agrègent par profil similaire proche. Les nœuds peuvent ainsi trouver dans leur voisinage les documents qui les intéressent et avoir de leur point de vue des recherches qui leur sont exhaustives. Enfin, nous avons défini des critères de classement permettant de rendre les résultats de recherche pertinents et personnalisés aux utilisateurs.

Nous allons montrer dans les chapitres suivants que les critères de routage et de classement que nous avons définis permettent bien aux utilisateurs de trouver les documents qui leur soient pertinents et personnalisés.

Chapitre 4

Modèles des utilisateurs

Pour tester ou valider les algorithmes utilisés dans des systèmes de recherche ou dans des systèmes pair à pair, trois approches sont couramment employées : la preuve formelle, la simulation et le déploiement.

Si la première approche peut être utilisée dans des systèmes relativement simples, elle est difficilement applicable à des systèmes aussi dynamiques que les réseaux de pairs où le contenu des nœuds ainsi que les nœuds présents sur le réseau changent à tout moment.

Dans la deuxième approche, la simulation permet d'observer des propriétés du système dans le cas où ses acteurs, ici les utilisateurs ou leurs nœuds, se comportent d'une manière prédéfinie. La simulation est par définition une imitation ou représentation d'une situation ou d'un système par un autre. En tant que telle, plusieurs aspects de la simulation sont des simplifications du système réel ou proviennent de déductions faites à partir d'études sur des systèmes similaires. Bien entendu ces abstractions peuvent biaiser les résultats observés par simulation.

Enfin, la dernière approche, le déploiement consiste en l'implémentation du système proprement dit et à sa diffusion. Cependant cette approche bien qu'elle permette de tester en situation réelle le logiciel demande un investissement important pour la conception et l'implémentation du logiciel sans assurance qu'il soit réellement utilisé. En effet, inciter les utilisateurs à adopter le logiciel est loin d'être évident. Peu de systèmes P2P peuvent se vanter de dépasser les milliers d'utilisateurs[1]. Enfin, mettre à jour le logiciel pour améliorer les algorithmes ou corriger des problèmes est une tâche difficile dans des systèmes décentralisés.

Nous avons étudié en parallèle les deux dernières approches.

D'une part la simulation d'utilisateurs de MAAY afin de pouvoir observer dans un contexte précis l'efficacité du système puis de modifier les algorithmes et les paramètres pour l'améliorer.

D'autre part le déploiement en réfléchissant sur les fonctionnalités de notre système qui pourront inciter les utilisateurs à l'utiliser et la manière de lier ces fonctionnalités avec les algorithmes décrits dans le chapitre 3.

Dans ce chapitre, nous allons décrire dans une première partie, les différentes études de traces qui ont été effectuées et les différents modèles d'utilisateurs qui ont été proposés. Nous verrons que ces traces et ces modèles sont très spécifiques à leurs systèmes et si certaines caractéristiques des utilisateurs peuvent en être extraites, elles demeurent encore incomplètes pour modéliser des utilisateurs d'un moteur de recherche textuel. C'est pourquoi nous allons définir dans la deuxième partie, un modèle des utilisateurs de moteur de recherche textuel qui se base sur les études des caractéristiques des utilisateurs de moteur de recherche et de système d'échange de fichiers, ainsi que sur des études bibliométriques. Nous décrirons tout

[1] Slyck. `http://www.slyck.com`.

d'abord la partie statique avec les relations et les répartitions entre les utilisateurs, documents et mots. Puis nous définirons le comportement dynamique des utilisateurs pour simuler leurs recherches, téléchargements et arrivée dans le système. Enfin, nous proposerons des mesures pour évaluer l'efficacité du système a retourner des résultats pertinents et personnalisés.

4.1 Terminologie

4.1.1 Graphes et réseaux

Réseau Un *réseau* désigne un ensemble d'objets appelés nœuds reliés entre eux par des relations. Il existe des multitudes de réseaux dont le réseau routier, le réseau social, le réseau Internet, le réseau électrique, `etc...`

Graphe Le *graphe* est une représentation symbolique d'un réseau. Il s'agit d'une abstraction de la réalité qui permet sa modélisation. Un *graphe* est composé de *sommets* (ou nœuds) reliés par des liens appelés *arcs* (ou arêtes). Ces *arcs* peuvent être orientés, on parle alors de *graphe orienté* dans l'autre cas, on parle de *graphe non orienté*. Ces arcs peuvent aussi être pondérés.

On note le graphe $G = (V, E)$ avec $V = \{v1, v_2, \ldots, v_n\}$ l'ensemble de ses n sommets et E l'ensemble des arcs entre les sommets. On note e_{jk} l'arc reliant le sommet v_j au sommet v_k.

Degré d'un sommet Dans le cas d'un graphe non-orienté, le *degré* d'un sommet s est égal au nombre d'arcs incidents à s. Dans le cas d'un graphe orienté, on peut distinguer le *degré sortant* d'un sommet s égal au nombre d'arcs ayant s comme extrémité initiale et le *degré entrant* d'un sommet s égal au nombre d'arcs ayant s comme extrémité finale.

Distance entre deux sommets La *distance* entre deux sommets A et B est égale au nombre minimal d'arcs traversés pour aller de A vers B. Les arcs peuvent être traversés dans les deux sens dans le cas d'un graphe non-orienté mais dans un seul sens dans un graphe orienté.

Distance moyenne du graphe La distance moyenne se calcule en effectuant la moyenne des distances entre toutes les paires de sommets du graphe.

Diamètre d'un graphe Le diamètre d'un graphe représente la distance maximale entre deux sommets du graphe.

Coefficient de clustering d'un sommet Il mesure la probabilité moyenne que deux sommets voisins d'un troisième sommet soient voisins entre eux.

Soit N_i le voisinage du sommet v_i composé des sommets connectés à v_i :

$$N_i = \{v_j / e_{ij} \in E\}$$

Le degré k_i d'un sommet v_i est égal au nombre de ses voisins : $k_i = |N_i|$.

Le coefficient de clustering C_i d'un sommet v_i du graphe est égale au ratio de liens entre les voisins de i sur le nombre total de liens possibles entre ses voisins, c'est-à-dire :

$$C_i = \frac{|\{e_{jk} / v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)}$$

Coefficient de clustering du graphe Le coefficient de clustering $\overline{C}$ du graphe correspond à la moyenne du coefficient de clustering de chacun de ses sommets :

$$\overline{C} = \sum_{i=1}^{n} \frac{C_i}{n}$$

4.1.1.1 Réseau aléatoire

Un réseau aléatoire est un réseau où les liens entre les nœuds sont disposés de manière aléatoires. Erdös et Rényi ont proposé un modèle éponyme pour construire ces graphes.

Soit n le nombre de sommets et $\bar{k}$ le degré moyen des sommets. Partant de n sommets, on sélectionne des couples de nœuds au hasard et on les relie jusqu'à obtenir le nombre d'arêtes souhaitées. Une autre construction possible est de considérer tous les couples de sommets et pour chacun d'entre eux, mettre une arête entre les deux avec une probabilité p .

Ces réseaux exhibent une distance moyenne faible $d = \frac{log(n)}{log(\bar{k})}$ mais un coefficient de clustering faible $\overline{C} = \frac{2\bar{k}}{n}$. La distribution des degrés suit une loi de Poisson, c'est-à-dire que la probabilité qu'un nœud ait pour degré k est égale à :

$$p_k = \frac{e^{-\bar{k}}\bar{k}^k}{k!}$$

4.1.1.2 Réseau small world

Un réseau *small world* [109, 97] est une classe de réseau dans lequel le voisinage local de chaque nœud est préservé et où la distance moyenne entre deux nœuds du réseau est courte comparé au nombre de nœuds.

De manière plus formelle, un réseau est dit *small world* si :

- son coefficient $\overline{C}$ de clustering moyen est supérieur à celui d'un réseau aléatoire :

$$\overline{C} > \frac{2\bar{k}}{n}$$

  avec $\bar{k}$ le degré moyen des nœuds et n le nombre de nœuds dans le réseau.
- la distance $\overline{D}$ moyenne est comparable à celle d'un réseau aléatoire :

$$\overline{D} \approx \frac{log(n)}{log(\bar{k})}$$

  avec $\bar{k}$ le degré moyen des nœuds et n le nombre de nœuds dans le réseau.

4.1.1.3 Réseau scale free

Les coefficients de clustering ainsi que la distance moyenne ne suffisent pas à caractériser les réseaux *small world* observés dans la réalité. En effet, ceux-ci sont aussi *scale free*, c'est-à-dire que la distribution des degrés de ces nœuds suit une loi de puissance.

Un réseau est dit *scale free* si la probabilité p_k qu'un nœud dans le réseau soit de degré k est proportionnelle à une puissance de k :

$$p_k \approx k^r$$

avec r l'exposant de la loi de puissance (souvent proche de 2).

Cette propriété contraste avec les réseaux aléatoires où la distribution des degrés suit une loi de Poisson.

4.1.2 Loi de distribution

Les lois de distribution décrivent la probabilité d'apparition de toutes les valeurs d'une variable aléatoire.

Fonction de densité Soit une variable aléatoire (v.a.) continue X qui peut prendre toute les valeurs d'un intervalle I dans $\mathbb{R}$. On note la probabilité que X prenne des valeurs comprises dans l'intervalle J : $P[X \in J]$. La fonction de densité f décrit les probabilités des valeurs que peut prendre la v.a. X et on a :

$$P[X \in J] = \int_J f(x).dx$$

Fonction de répartition La fonction de répartition F décrit la probabilité que la v.a. X soit inférieure à une valeur x :

$$F(x) = P[X < x]$$

Et on a :

$$F(x) = \int_{-\infty}^{x} f(x).dx$$

Espérance L'espérance E de la v.a. X correspond à la moyenne des valeurs possibles de la v.a. Xsur l'intervalle I :

$$E(x) = \int_I x.f(x).dx$$

4.1.2.1 Loi de distribution uniforme sur un intervalle

La loi de distribution uniforme sur un intervalle $[a, b]$ décrit une densité dont la distribution est constante sur l'intervalle [a,b]. La fonction de densité f d'une loi de distribution uniforme s'écrit :

$$f(x) = \frac{1}{b-a}$$

4.1.2.2 Loi de puissance

Deux variables x et y sont liées par une loi de puissance si on a :

$$y = a.x^k$$

avec a la constante de proportionnalité et k l'exposant de la loi de puissance.

Plusieurs lois de puissance ont été établies et touchent des domaines divers dont la physique, la biologie, les réseaux sociaux, ...

Un exemple de représentation d'une fonction qui suit une loi de puissance est présenté sur la figure 4.1 et sur la figure 4.2 en échelle logarithmique.

Loi de Zipf (rang-fréquence) La loi de Zipf est née des observations de George Kingsley Zipf, professeur de linguistique à Harvard. Il a découvert que la fréquence d'utilisation du k^{ieme} mot dans n'importe quel langage est inversement proportionnelle à k.

Les distributions qui vérifient la loi de Zipf sont observées dans plusieurs domaines : la distribution des tailles des villes, la distribution des accès aux pages du web ou encore la magnitude des tremblement de terre.

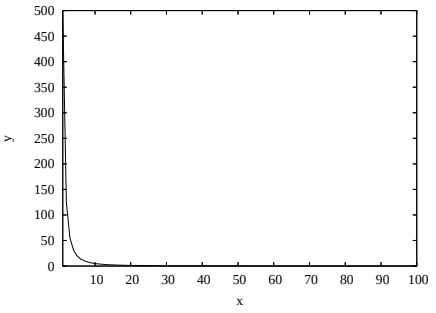


FIG. 4.1: Représentation de la fonction $f(x) = 500.x^{-2}$

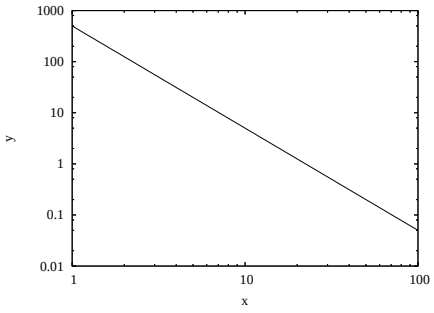


FIG. 4.2: Représentation de la fonction $f(x) = 500.x^{-2}$ en échelle logarithmique

La loi de Zipf stipule que la probabilité P_i d'un évènement i est proportionnelle à son rang r_i à la puissance $-b$ avec b souvent proche de 1 :

$$P_i \approx r_i^{-b}$$

Pour montrer qu'une distribution suit une loi de Zipf, il suffit de représenter la distribution dans un graphe en échelle logarithmique avec en abscisse le rang et en ordonnée la fréquence. Si l'ensemble des points peut être approximé par une droite, alors la distribution suit une loi de Zipf.

Loi de Lotka Alfred Lotka un scientifique autrichien a été rendu célèbre pour avoir montré que le nombre d'auteurs ayant écrit n publications est décrit par une loi de puissance de la forme $\frac{C}{n^a}$ avec C une constante égale à $\frac{6}{\pi}$ et a un exposant proche de 2. Et donc que 61% des auteurs n'ont qu'une seule contribution.

Equivalence de la loi de Lotka et la loi de Zipf On peut montrer mathématiquement que la formulation de la loi de Lotka et celle de la loi de Zipf sont équivalentes. Autrement dit, si le nombre de publications écrites par un auteur est proportionnel à son rang exposant une constante, alors le nombre d'auteurs ayant écrit k publications est proportionnel à k exposant une constante.

Dans la suite, nous utiliserons la loi de puissance telle qu'exposée par celle de Lotka, c'est-à-dire la relation entre une valeur caractérisant un objet (nombre de voisins, nombre de mots d'un document, ...) et le nombre d'objets caractérisés par cette valeur.

Loi de Poisson La distribution de Poisson découverte par Siméon-Denis Poisson a été publiée en 1838 pour ses travaux sur la probabilité des jugements en matières criminelles et matière civile. Cette distribution est souvent utilisée pour décrire aussi bien le nombre d'appels reçus par minute par un call center, le nombre d'accès à un serveur web par minute, le nombre d'avions qui arrivent sur un aéroport.

Cette loi stipule que la probabilité $P(N_t = k)$ qu'un évènement se produise k fois durant un intervalle de temps t est égale à :

$$P(N_t = k) = \frac{e^{-\lambda t}(\lambda t)^k}{k!}$$

avec λ le nombre attendu d'occurrences durant l'intervalle t. Par exemple, si un évènement se produit en moyenne toutes les 2 minutes et que l'intervalle t est de 10 minutes, alors $\lambda = 5$

4.2 Etudes des utilisateurs à partir de données réelles

Nous allons décrire dans cette section des études effectuées sur des systèmes de recherche, sur la bibliométrie et sur les relations entre les mots.

Les traces des moteurs de recherche [22, 14, 19, 92, 21, 20] contiennent les requêtes de recherche soumises au moteur et les pages décrites dans les résultats qui ont été visitées. Elles concernent des moteurs de recherche variés : AltaVista, AllTheWeb.com, Excite et Vivisimo, et s'étalent sur des périodes pouvant aller d'une journée à 6 semaines. Les traces décrivent pour chacune des requêtes soumises au moteur de recherche, l'identifiant de l'utilisateur ainsi que la date à laquelle elle a été effectuée.

Les traces de système pair à pair [85, 86, 90, 19, 88] sont plus hétérogènes et varient en fonction du système étudié : Gnutella, KaZaA, Napster, eDonkey et du moyen utilisé pour récupérer les traces.

Des études bibliométriques [107, 96, 99] permettent indirectement de caractériser les documents qui intéressent les utilisateurs.

Enfin, les travaux [102, 108] sur les relations entre les mots proviennent d'études sur des corpus de documents, des thésaurus ou des classification de mots. Elles permettent de donner une idée sur la distribution des mots contenus dans les documents et les mots utilisés dans les requêtes de recherche des utilisateurs.

Nous allons résumer dans cette sous-section, les caractéristiques des requêtes de recherche, des utilisateurs et des relations entre les mots et les documents qui ont été observées.

4.2.1 Les requêtes de recherche

Composition des requêtes Les études [92, 22, 21, 14, 20] sur les moteurs de recherche montrent que le nombre moyen de mots utilisés par requête est compris entre 2 et 3. L'étude des traces d'Altavista[22] montrent que 25.8% des requêtes n'ont qu'un mot, 26.0% 2 mots, 15% 3 mots et 12.6% ont plus de 3 mots. L'étude des requêtes de Gnutella [88] montrent que la moyenne du nombre de mots utilisés par requête est de 4 avec 19% des requêtes qui ne contiennent que deux mots et 63% des requêtes qui en contiennent plus de 2.

Popularité des requêtes de recherche Une étude [92] sur les traces d'Excite et de Vivisimo en début 2001 montrent que la distribution des requêtes de recherche suivent une loi de Zipf. Et que 16 à 22% des requêtes sont répétées par le même utilisateur. Des requêtes sont plus demandées que les autres, les 25 requêtes les plus fréquentes représentent 1.5% des requêtes de recherche. Une autre étude [86] sur le système Gnutella 0.4 en janvier 2001 montre que les requêtes ne suivent pas strictement une loi de Zipf. Les requêtes très populaires ont une fréquence d'apparition presque égale.

Popularité des mots dans les requêtes de recherche L'étude [21] montre que la distribution des mots utilisés dans les requêtes des moteurs de recherche ne suit pas strictement une loi de Zipf : les mots très populaires ont une fréquence d'apparition presque égale ainsi que les mots peu populaires. On observe la même distribution dans les requêtes de recherches utilisées dans le système Gnutella [88]. Les auteurs justifient le fait que les mots populaires ont un nombre d'occurrences quasiment égal à cause de l'intérêt élevé des utilisateurs pour les contenus classés X qui d'une part, contrairement aux fichiers musicaux ne désignent pas une chanson en particulier et d'autre part, utilisent des mots qui leurs sont propres.

Popularité des fichiers Des études sur les traces de plusieurs serveur proxy[2] web [17] ont montré que la distribution des requêtes de téléchargement de pages web suivent une loi de Zipf.

Les études [83, 85] sur les traces des systèmes eDonkey2000 et KaZaA montrent que la distribution de nombre de réplicats des fichiers suit presque une loi de Zipf hormis pour les fichiers de rang élevé dont le nombre de réplicats est presque constant. Les raisons qui expliquent cette distribution sont que contrairement aux pages du web, les fichiers sont immuables et ne sont téléchargés qu'au plus une seule fois par utilisateur.

4.2.2 Utilisateur

Taille du lexique L'étude [92] montre que la distribution de la taille du lexique (vocabulaire) des utilisateurs ne suit pas strictement une loi de Zipf. Les utilisateurs ayant un lexique riche ont une taille de lexique plus faible que ce qu'ils devraient avoir si la distribution suivait une loi de Zipf de même que les utilisateurs avec un lexique pauvre.

Relations entre les mots du lexique Solé et Ferrer i Cancho [102] ont étudié le graphe d'interactions des mots où deux mots sont liés s'ils coocurrent dans la même phrase et ont montré que ce graphe présente des propriétés *small world* et *scale free*. La distance moyenne dans ce graphe est de 2.6 pour un graphe de 470000 mots et le coefficient de clustering de 0.4 à 0.6 (celui d'un réseau aléatoire est de $1.55.10^{-4}$). Une étude [108] sur 3 autres graphes de mots aboutit aux mêmes conclusions. Dans le premier graphe, un mot est lié à un autre si parmi les 6000 participants, au moins un participant à qui on présentait le premier mot, citait parmi les mots qui lui venaient à l'esprit l'autre mot. Le deuxième graphe se base sur le thésaurus de Roget où chaque mot est associé à plusieurs catégories. Pour construire le graphe des mots, on relie deux mots s'ils appartiennent à la même catégorie. Le troisième graphe utilise le réseau WordNet[3] qui contient plus de 120000 mots et plus de 100000 significations. Le graphe des mots étudiés est celui qui relie deux mots s'ils sont ont une relation de synonymie (*e.g.* FROID et GLACIAL), antonymie (*e.g.* LENT et RAPIDE), hyperonymie (*e.g.* le CHÊNE est un ARBRE) et méronymie (*e.g.* un ARBRE a des FEUILLES).

Popularité des mots parmi les utilisateurs L'étude [88] sur le système Gnutella a montré que la distribution du nombre d'utilisateurs (ayant le mot dans le nom des fichiers qu'ils partagent) par mot ne suit pas une loi de Zipf mais une fonction de la forme $a.x - (b.x + c)^{\alpha}$ avec a, b, c et α des constantes et x le rang du mot. Sa déviation par rapport à la loi de Zipf est de 13%.

Nombre de résultats lus / récupérés L'étude [22] sur des traces du moteur de recherche AltaVista montre que le nombre de pages de (10) résultats observés par utilisateurs est en moyenne de 1.39. Pour 85% de leurs requêtes de recherche, les utilisateurs ne regardent qu'une page de résultats.

Temps de connexion de l'utilisateur L'étude [90] sur les réseau Gnutella 0.4 et Napster montrent que la durée médiane de connexion au système Gnutella et Napster est de une heure ce qui correspond au temps moyen pour télécharger quelques fichiers de musique.

[2] Un serveur proxy web est un serveur qui a pour fonction de relayer les requêtes de téléchargement de page web des utilisateurs et de leur retourner le contenu des pages demandées.

[3] WordNet. A Lexical Database for the English Language. `http://wordnet.princeton.edu/`.

Graphe de coopérations Le graphe des coauteurs (ou de collaboration) est composé d'auteurs qui sont liés s'ils ont écrit un article ensemble. Newman [106] a étudié ces graphes extraits de différentes bases de données dont une sur MEDLINE (recherche biomédicale), Los Alamos e-Print Archive (physique) et NCSTRL (informatique). Il a montré que ces graphes présentaient des caractéristiques *small world* et *scale free*. Et que la distribution du nombre de collaborateurs suivait une loi de puissance mais avec une coupure exponentielle. L'étude [99] qui étudie le graphe bipartie entre les auteurs et leurs publications à partir de la base de donnée de DBLP[4] montre que le graphe des coauteurs est *small world* et *scale free*, la distance moyenne dans ce graphe est de 6. Iamnitchi, Ripeanu et Foster [103] ont étudié 3 réseaux d'utilisateurs : (i) le réseau des physiciens qui relient deux physiciens s'ils ont partagé plusieurs ressources de calcul et de stockage ; (ii) le réseau des internautes ayant accédé au site web de Boeing où deux utilisateurs sont liés s'ils ont visité un ensemble de pages en commun ; (iii) le réseau des utilisateurs de KaZaA où deux utilisateurs sont liés s'ils partagent plusieurs fichiers en commun. Et ils ont montré que ces réseaux exhibaient des propriétés *small world* et que les deux derniers réseaux affichaient des propriétés *scale free*.

Nombre de documents publiés Alfred Lotka un scientifique autrichien a été rendu célèbre pour avoir montré que le nombre d'auteurs de n publications est décrit par une loi de puissance de la forme $\frac{C}{n^a}$ avec C une constante égale à $\frac{6}{\pi}$ et a un exposant souvent proche de 2. Et donc que 61% des auteurs n'ont qu'une seule contribution. L'étude [99] sur le graphe bipartie entre les auteurs et leurs articles confirme cette loi.

Nombre d'auteurs par publication L'étude [99] sur les articles de DBLP montre que la distribution du nombre d'auteurs parmi les articles suit une loi de puissance.

Nombre de fichiers partagés par utilisateur Les études [82] effectuée en 2000 et [90] en 2001 sur le système Gnutella montre que le nombre de fichiers échangés par les utilisateurs est très hétérogène. Une part non négligeable d'utilisateurs, les freeriders, ne partagent aucun fichier. L'étude [82] effectuée en 2000 montre une proportion importante de 66% de freeriders et que les 20% des nœuds qui partagent le plus de fichiers partagent 98% des fichiers. Une autre étude [88] sur le réseau Gnutella effectuée en septembre 2003 montre que 27% des utilisateurs sont des freeriders et que 68% des utilisateurs partagent moins de 100 fichiers et 6.4% plus de 1000 fichiers. L'étude [90] effectuée en 2001 montre une proportion beaucoup moins élevée de freeriders qui ne représentent que 25% des utilisateurs. Cette étude montre que 75% des utilisateurs partagent moins de 100 fichiers et seulement 7% partagent plus de 1000 fichiers. Une autre étude [83] sur les traces eDonkey2000 en novembre 2003 montre que 68% des utilisateurs sont des freeriders.

4.2.3 Fichiers

Popularité des mots dans les documents George Kingsley Zipf un linguiste américain a étudié la fréquence statistique des mots dans différents langages. Il est à l'origine d'une loi éponyme qui stipule que la fréquence d'un mot suit une la loi de puissance $\frac{1}{n^a}$ avec n le rang du mot et a une constante proche de 1. Ce qui veut dire qu'un mot de rang r occurre r fois moins que le mot de rang 1. L'étude [88] sur les fichiers partagés dans le système Gnutella montre que la distribution des mots dans les noms de fichiers suit approximativement une loi de Zipf.

[4] DBLP : Computer Science Bibliography. `http://dblp.uni-trier.de/`.

Distribution des références Des études sur la distribution du nombre de références par publication scientifique [107, 96] sur différents journaux ont montré que la probabilité qu'un papier soit cité k fois était proportionnelle à une puissance de k (loi de puissance).

Relations entre les fichiers L'étude [83] sur des traces du système eDonkey2000 montre que plus un utilisateur partage des fichiers en commun avec un autre utilisateur, plus la probabilité qu'ils partagent ensemble un autre fichier augmente. Ainsi en partageant un fichier en commun, la probabilité de partager un autre fichier en commun est de 35% et augmente rapidement à 80% quand le nombre de fichiers en commun est de 10.

4.3 Modèle des utilisateurs d'échange de fichiers

Peu de modèles existent sur les utilisateurs de systèmes pair à pair d'échange de fichiers. Ces modèles s'inspirent des études que nous avons montrées précédemment pour définir les fichiers partagés par les utilisateur, leurs requêtes, leur arrivée dans le système, ...

Nous allons décrire deux modèles généralement utilisés pour simuler un système de recherche pair à pair.

4.3.1 Modèle basé sur les propriétés observées

Les articles [95, 49] propose un modèle des utilisateurs d'un système d'échange de fichiers.

Distribution des fichiers Les fichiers sont répartis dans plusieurs catégories et le nombre de fichiers dans chaque catégorie dépend de la popularité de la catégorie qui suit une loi de Zipf. Au sein de ces catégories, des fichiers vont être plus populaires que d'autres et suivre une loi de Zipf. Pour la répartition du nombre de fichiers partagés par utilisateur, le modèle utilise l'étude de Saroiu [90] sur les freeriders. La probabilité qu'un fichier appartenant à une catégorie soit partagé par un utilisateur est proportionnel à son niveau d'intérêt pour la catégorie et inversement proportionnel au rang du fichier dans la catégorie.

Distribution des intérêts Chaque utilisateur est intéressé par un ensemble de catégories : il ne va partager et rechercher que des fichiers appartenant à ces catégories. Des catégories vont être plus populaires que d'autres et intéresser davantage d'utilisateurs. La distribution des catégories parmi les utilisateurs suit une loi de Zipf. La probabilité qu'un utilisateur soit intéressé par une catégorie est inversement proportionnelle au rang de la catégorie. Le niveau d'intérêt pour chacune de ces catégories est choisie de manière uniforme dans le modèle [95] ou suivant une loi de Zipf dans le modèle [49].

Requête de recherche La distribution des périodes entre deux émissions de requêtes de recherche d'un utilisateur suit une loi de Poisson avec une moyenne choisie de manière uniforme sur un intervalle borné. Les utilisateurs recherchent un fichier d'une catégorie proportionnellement à leur niveau d'intérêt pour la catégorie et inversement proportionnelle au rang du fichier dans la catégorie (Zipf). Cette distribution en loi de Zipf des requêtes est confirmée par l'étude de Sripandikulchai[86].

Connexion des utilisateurs Les utilisateurs restent connectés au systèmes suivant une probabilité reprise de l'étude de Saroiu [90] avec un temps médian de une heure. Les utilisateurs se joignent au système en se connectant à un nœud choisi proportionnellement à

son nombre de connexions. Cette attachement préférentiel permet d'obtenir des réseaux qui exhibent des propriétés *scale free*.

Croissance du nombre d'utilisateurs Deux types de stratégies sont utilisés pour la construction du réseau. Certains modèles considèrent qu'initialement tous les utilisateurs sont déjà connectés au réseau. Le réseau des nœuds peut être construit de manière aléatoire [55, 59, 54] ou avec des propriétés *scale free* [59]. Au contraire, d'autres modèles [57, 51] considèrent que le réseau croit de manière linéaire jusqu'à atteindre un nombre maximal de nœuds (*e.g.* 1000).

4.3.2 Modèle utilisant des données réelles

Un autre système d'échange de fichiers[58] à la Gnutella va utiliser des traces de requêtes de téléchargement de pages du web et des traces sur les requêtes de téléchargement dans les systèmes Gnutella et KaZaA. Ces traces contiennent l'ensemble des téléchargements qui sont chacun décrit par l'utilisateur qui l'a initié, la référence du fichier téléchargé et sa date. Lors de la simulation, les utilisateurs générés vont rejouer les requêtes qu'ils ont effectuées : ils vont lancer des requêtes de recherche sur les fichiers puis les télécharger. Ces fichiers téléchargés sont ensuite partagés. Au début de la simulation, les utilisateurs vont partager les fichiers pour lesquels ils sont les premiers à les demander (dans les traces).

C'est aussi l'approche utilisée pour simuler le système QRE [31]. Ce système a été simulé en utilisant les traces du système eDonkey. Les traces contiennent pour chaque requête de recherche d'un fichier, l'identifiant du nœud qui a émis la recherche et les identifiants des nœuds qui possèdent le fichier. A partir de ces traces, des utilisateurs et leurs fichiers partagés ont été générés. Puis les requêtes de recherche ont été rejouées dans l'ordre chronologique pour simuler les requêtes des utilisateurs.

4.3.3 Mesures d'efficacité

Pour mesurer l'efficacité d'un système par rapport à un autre, on peut utiliser différentes mesures et les comparer pour déduire les gains du système.

4.3.3.1 Mesures quantitatives

Plusieurs mesures peuvent être utilisées pour évaluer quantitativement l'efficacité du système. Une liste non exhaustive est présentée ci-dessous :

- nombre de messages pour effectuer une requête de recherche
- nombre de nœuds interrogés par recherche
- nombre de retransmissions minimum pour trouver un résultat
- latence entre l'émission d'une requête de recherche et la réception de la première/dernière réponse
- pourcentage de réussites des recherches
- bande passante utilisée par recherche
- nombre de résultats reçus (exhaustivité des réponses)
- charge des nœuds (bande passante et nombre de messages reçus) et répartition des charges entre les nœuds
- évolution de la distance moyenne entre nœuds
- évolution de la topologie du graphe des nœuds (propriété *small world* et *scale free*).

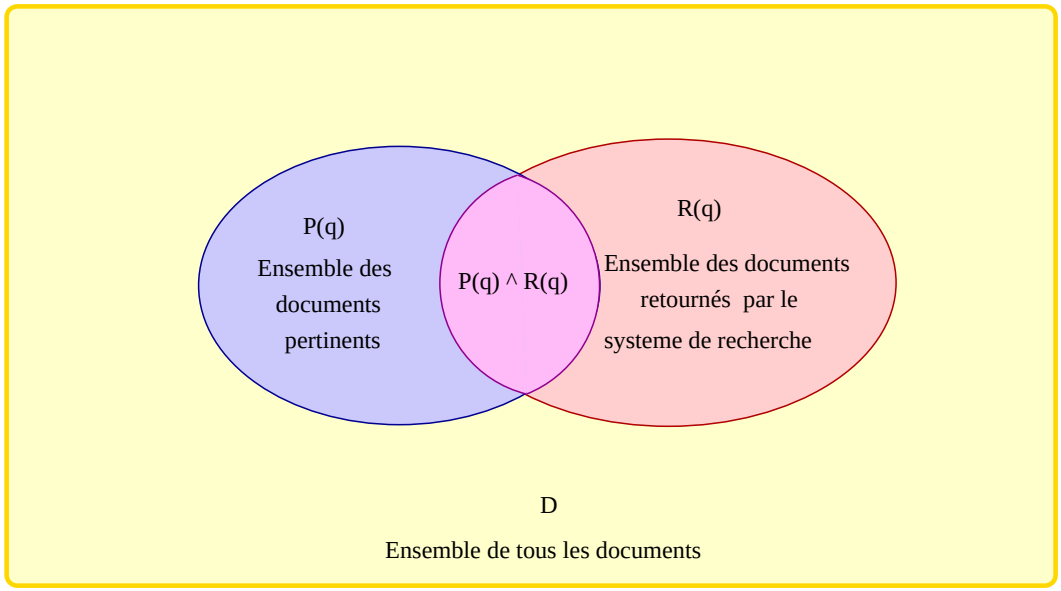


FIG. 4.3: Précision et rappel

4.3.3.2 Mesures qualitatives

En recevant une requête de recherche q, le système de recherche va sélectionner parmi l'ensemble D des documents indexés, un sous-ensemble $R(q)$ choisi suivant plusieurs critères (exemples de critères à la section 2.1.4). On définit $P(q)$ l'ensemble des documents pertinents à la requête.

Pour mesurer la qualité des résultats, c'est-à-dire si une part importante des documents pertinents de $P(q)$ sont dans $R(q)$, on peut utiliser deux mesures couramment employées dans les systèmes de recherche d'information : la *précision* et le *rappel*.

PRÉCISION : La précision $precision(q)$ mesure le ratio de documents pertinents pour la requête q retournés par le système de recherche sur le nombre de résultats reçus :

$$precision(q) = \frac{P(q) \cap R(q)}{R(q)}$$

avec $P(q)$ l'ensemble des documents pertinents pour la requête q et $R(q)$ l'ensemble des documents retournés par le système de recherche.

Cette mesure permet d'apprécier si les résultats retournés par le système de recherche sont pertinents ou non. Une valeur de précision de 0.33 signifie qu'un tiers des résultats retournés par le moteur de recherche sont pertinents.

RAPPEL : Le *rappel* mesure le ratio de documents pertinents retournées par le moteur de recherche sur le nombre total de documents pertinents :

$$rappel(q) = \frac{P(q) \cap R(q)}{P(q)}$$

avec $P(q)$ l'ensemble des documents pertinents pour la requête q et $R(q)$ l'ensemble des documents retournés par le système de recherche.

Cette mesure décrit la complétude des réponses pertinentes retournées par le système de

recherche. Une valeur de rappel de 0.33 signifie que le système n'a retourné qu'un tiers de tous les résultats pertinents.

4.3.4 Conclusion

Nous avons étudié au travers de traces, les caractéristiques des utilisateurs, de leurs documents et de leurs mots. Nous avons vu notamment que plusieurs distributions dont la popularité des fichiers, popularité des mots, taille du lexique suivent une loi de Zipf ou s'en rapprochent. Nous avons pu observer que plusieurs propriétés dont la popularité des requêtes et des fichiers sont communes aux systèmes centralisés et pair à pair.

Nous avons ensuite décrit deux techniques couramment employées pour simuler des utilisateurs de systèmes pair à pair.

Finalement, nous avons énuméré un ensemble de mesures couramment employées pour mesurer l'efficacité d'un système de recherche.

Dans le cas particulier des systèmes d'échange de fichiers, quelques modèles parmi ceux présentés précédemment permettent de définir les requêtes de recherche de l'utilisateur en fonction de son profil. Cependant les requêtes de recherche dans ces systèmes correspondent plus à une recherche de fichiers précis par leurs identifiants (nom du fichier). Ce qui est différent d'un système de recherche textuel où l'utilisateur recherche des documents qui contiendrait une information mais pour lesquels il n'a que pour seuls indices que quelques mots clés. Dans le cas général, aucune étude à notre connaissance n'ont été effectués sur les liens entre les fichiers téléchargés et les mots utilisés dans la requête de recherche. De même qu'entre les réponses de recherche et les fichiers téléchargés.

C'est précisément ces propriétés que nous allons essayer de déduire dans la section suivante à partir des études présentées dans ce chapitre.

4.4 Modèle proposé

Nous allons décrire dans ce chapitre le modèle d'utilisateurs que nous avons conçu qui respecte les caractéristiques des utilisateurs, documents et mots présentées dans le chapitre précédent. Nous exposerons tout d'abord le modèle statique des utilisateurs en détaillant les distributions des documents et des mots parmi les utilisateurs. Puis nous aborderons le modèle dynamique en décrivant les actions effectuées par les utilisateurs, typiquement les mots tapés dans leurs recherches, leurs téléchargements et leur arrivée dans le système. Nous introduirons une mesure de qualité pour mesurer la pertinence d'un document pour un utilisateur et une requête de recherche donnés.

4.4.1 Modèle statique

Un utilisateur connaît un ensemble de mots qui compose son vocabulaire. Le nombre de mots, généralement de plusieurs milliers pour une personne érudite ne représente qu'une petite fraction des mots existants (Google a un lexique de 14 millions de mots [4]). Un utilisateur est aussi caractérisé par ses connaissances acquises lors de lectures de document et certains utilisateurs sont beaucoup plus cultivés que d'autres. Les documents sont composés de mots et le nombre de mots peut varier de plusieurs centaines pour une petite dépêche de presse à plusieurs millions pour une encyclopédie. Un résultat bien connu est que quelques mots vont être plus employés que d'autre, c'est ce que décrit la loi de Zipf.

Ces relations peuvent être modélisées par 3 graphes biparties (voir figure 4.4) :

- le graphe bipartie entre les utilisateurs et les documents qu'ils connaissent
- le graphe bipartie entre les documents et les mots qu'ils contiennent

– le graphe bipartie entre les utilisateurs et les mots qu'ils emploient

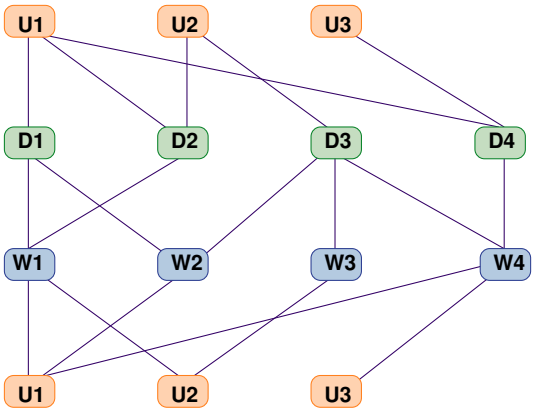


FIG. 4.4: Modèle des utilisateurs

Nous allons décrire dans cette section les caractéristiques et les propriétés de ces graphes.

4.4.1.1 Propriétés du modèle

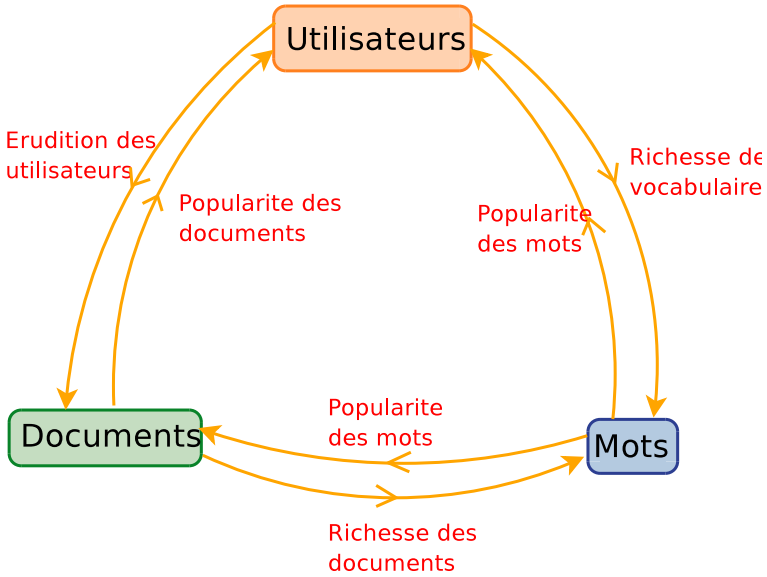


FIG. 4.5: Distribution

Propriétés des distributions Nous allons essayer de caractériser les relations entre les documents, mots et utilisateurs en étudiant leurs distributions.

Les 3 relations sont caractérisées par les six distributions suivantes représentées dans la figure 4.5 et le tableau 4.1.

D'après les études que nous avons présentées dans le chapitre précédent, il apparaît que la distribution du nombre de documents par utilisateur, du nombre de documents par mot, du nombre d'utilisateurs par document et du nombre de mots par utilisateur suivent une loi de puissance. Intuitivement, on peut aussi penser que la distribution du nombre d'utilisateurs par mot suit aussi une loi de puissance.

Distribution	propriété
nombre de documents lus/connus par utilisateur	loi de puissance
nombre d'utilisateurs par document	loi de puissance
nombre d'utilisateurs par mot	loi de puissance
nombre de documents par mot	loi de puissance
nombre de mots employés par utilisateur	loi de puissance
nombre de mots utilisés par document	?

TAB. 4.1: Distributions

Pour confirmer ces études, nous avons étudié la base de données de DBLP contenant la liste des publications d'articles de conférences scientifiques dans le domaine informatique. Cette base contient 613995 articles et 399785 auteurs. Chaque entrée de la base est composée d'une description d'un article contenant son titre, et ses auteurs.

Nous avons étudié les 6 distributions possibles entre les utilisateurs, documents et mots et avons représenté en échelle logarithmique ces différentes distributions afin de pouvoir mettre en évidence le fait qu'elles suivent une loi de puissance ou non. (figure 4.12)

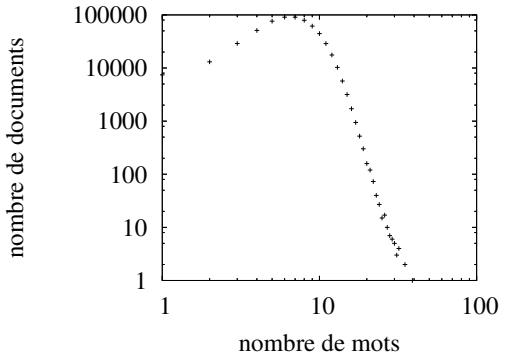


FIG. 4.6: Distribution du nombre de mots par document.

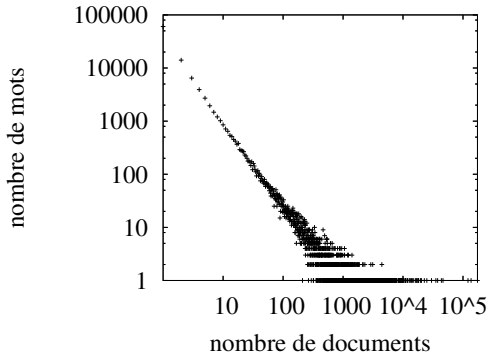


FIG. 4.7: Distribution du nombre de documents par mot. Par exemple il y a 851 mots qui apparaissent dans 10 documents.

Il apparaît clairement que presque toutes les distributions suivent une loi de puissance. La distribution du nombre de mots par utilisateur si elle est tronquée aux utilisateurs employant plus de 7 mots suit aussi une loi de puissance. Par contre la distribution du nombre de mots par document ne suit pas une loi de puissance.

Propriétés des réseaux Ces 6 relations définissent 6 réseaux étroitement liés (voir table 4.4.1.1).

D'après les études qui ont été effectuées et présentées aux sections 4.2.3 et 4.2.2 sur le graphe des utilisateurs liés par fichier en commun, à la section 4.2.2 sur le graphe des mots liés par document en commun, les deux premiers réseaux ont été formellement identifiés comme *small world* et *scale free*. Par contre, pour les quatre derniers réseaux, les propriétés de ces réseaux n'ont pas encore été étudiées à notre connaissance. Mais d'après les données que l'on possède sur d'autres réseaux, on peut fortement supposer qu'ils le sont aussi.

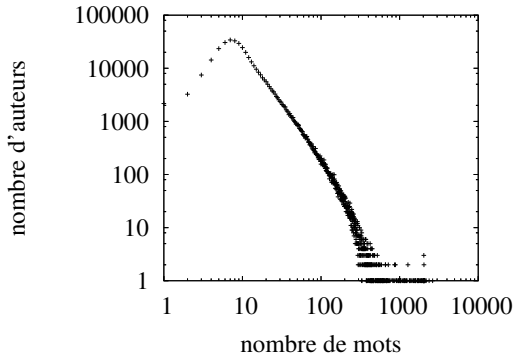


FIG. 4.8: Distribution du nombre de mots par auteur.

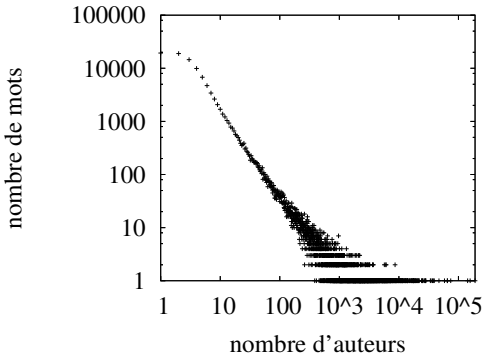


FIG. 4.9: Distribution du nombre d'auteurs par mot

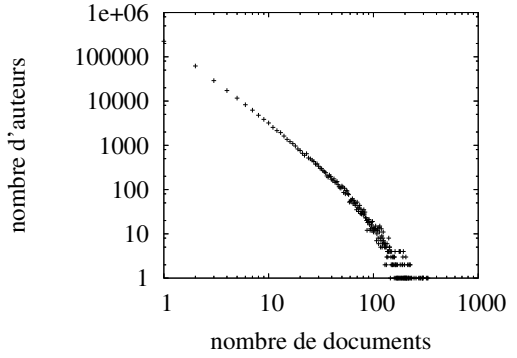


FIG. 4.10: Distribution du nombre de documents par auteur.

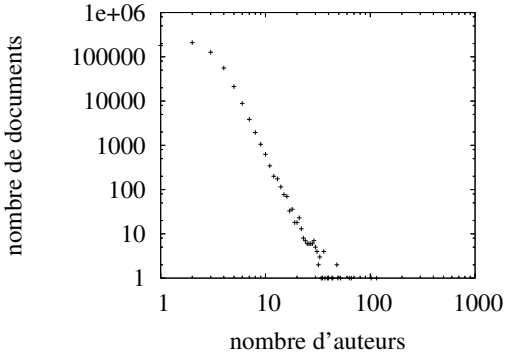


FIG. 4.11: Distribution du nombre d'auteurs par document

réseau	propriété
réseau des utilisateurs liés par document	SW et SF
réseau des mots liés par document (réseau d'interaction des mots)	SW et SF
réseau de documents liés par utilisateur	SW et SF
réseau des mots liés par utilisateur	SW et SF
réseau des utilisateurs liés par mot	SW et SF
réseau de documents liés par mot	?

TAB. 4.2: réseaux construits à partir des relations entre utilisateurs, documents et mots

De manière plus formelle, on peut le montrer en utilisant les travaux de Latapy [101] sur les graphes biparties. Ils ont montré que si l'on a :

- un graphe bipartie composé de deux types de nœuds : les nœuds hauts et bas
- la distribution du nombre de connexions vers les nœuds hauts par nœud bas suit une loi de puissance

alors le graphe unipartie des nœuds bas où deux nœuds bas sont liés s'ils ont au moins une connexions vers un même nœud haut est *small world* et *scale free*.

On peut donc déduire grâce aux 5 distributions en loi de puissance que l'on a montré précédemment que les 5 premiers réseaux sont *small world* et *scale free*.

4.4.1.2 Cohérence entre les documents connus par l'utilisateur et son lexique

Un utilisateur acquiert son vocabulaire à partir de ses connaissances. Ainsi par exemple, un français dont la majorité de ses connaissances est en français emploiera majoritairement des mots français tirés de ses connaissances. Pour que le vocabulaire de l'utilisateur soit cohérents avec ses connaissances, son vocabulaire est construit à partir des mots des documents qu'il connaît (figure 4.12).

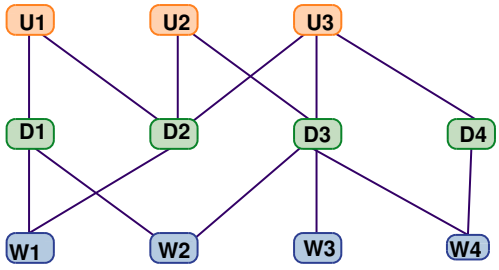


FIG. 4.12: Relations entre les utilisateurs et les documents et entre les documents et et les mots (vocabulaire)

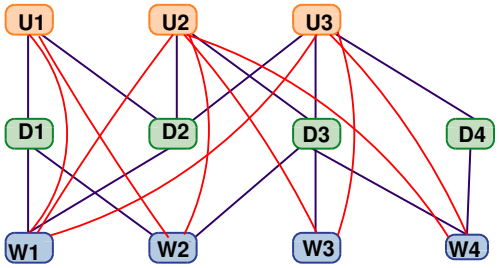


FIG. 4.13: Relations entre les utilisateurs et les mots

4.4.2 Modèle dynamique

4.4.2.1 Les requêtes de recherche des utilisateurs

Nombre de mots Les études sur les moteurs de recherche s'accordent sur une moyenne de 2-3 mots par requêtes de recherche. En utilisant les statistiques de [20, 22] et en les ajustant pour supprimer les requêtes vides ou les requêtes avec plus de 3 mots, on obtient : 34% des requêtes qui ont un mot, 42% qui en ont deux et 25% qui en ont trois.

Choix des mots Un utilisateur utilise dans ses requêtes de recherche les mots de son vocabulaire. Les mots sont choisis en fonction du nombre de documents connus par l'utilisateur qui possèdent ces mots et du nombre total de documents dans lesquels il apparaît. L'idée étant de privilégier les mots caractéristiques de ses connaissances, c'est-à-dire qui apparaissent fréquemment dans les documents de l'utilisateur mais rarement dans les autres documents. On définit la probabilité qu'un utilisateur n choisisse un mot m :

$$proba\_choix\_mot(n, m) = \frac{nombre\ de\ documents\ de\ n\ ayant\ le\ mot\ m}{nombre\ de\ documents\ ayant\ le\ mot\ m}$$

4.4.2.2 Les téléchargements des utilisateurs

Après avoir effectué une recherche, l'utilisateur reçoit des réponses et choisit parmi celles-ci un document à télécharger. L'utilisateur effectue son choix en fonction de son appréciation pour les documents.

Pour quantifier cette appréciation, nous définissons une mesure de qualité décrivant la pertinence d'un document d pour l'utilisateur u et pour une requête donnée. Cette mesure utilise les liens entre le document d et ses connaissances. L'idée étant que celle-ci est d'autant plus grande que d'une part, le document a des relations sémantiques avec les connaissances de u et d'autre part, que des utilisateurs partageant des connaissances avec u connaissent aussi le document d.

A partir du graphe bipartie entre les utilisateurs et les documents, et du graphe bipartie entre les documents et les mots, on peut définir deux graphes :

- le *graphe sémantique des documents* où deux documents sont liés s'ils ont un mot en commun (voir figure 4.14)
- le *graphe social des documents* où deux documents sont liés s'ils sont partagés par un utilisateur commun (voir figure 4.15)

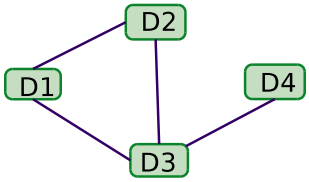


FIG. 4.14: Graphe sémantique des documents déduit des graphes biparties de la figure 4.12

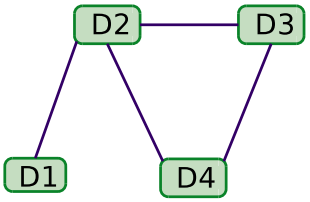


FIG. 4.15: Graphe social des documents déduit des graphes biparties de la figure 4.12

La qualité du document va être d'autant plus importante que :

- le nombre de chemins dans le graphe sémantique et sociale entre les documents de l'utilisateurs et le document cible est important
- ces chemins sont courts

Pour cela, on définit la *curiosité sémantique* et la *curiosité sociale* entre un document a et un document b qui quantifie l'intérêt que porte un utilisateur pour le document b sachant qu'il est intéressé par le document a. La curiosité sémantique utilise les relations sémantiques entre les documents (voir figure 4.15) alors que la curiosité sociale utilise les relations sociales entre les documents (voir figure 4.14).

CURIOSITÉ SÉMANTIQUE ET SOCIALE : La *curiosité sémantique* entre deux documents a et b est égale à :

$$curiosite\_sem(a \to b) = \sum_{l=1}^{k} \frac{nb\_chemins\_sem(a \to b, l)}{\sum_c nb\_chemins\_sem(a \to c, l)}$$

avec $nb\_chemins\_sem(x \to y, l)$ le nombre de chemins entre x et y dans le graphe sémantique des documents de longueurs l.

De la même façon, on définit la *curiosité sociale* :

$$curiosite\_soc(a \to b) = \sum_{l=1}^{k} \frac{nb\_chemins\_soc(a \to b, l)}{\sum_c nb\_chemins\_soc(a \to c, l)}$$

avec $nb\_chemins\_soc(x \to y, l)$ le nombre de chemins entre x et y dans le graphe sociale des documents de longueurs l.

Pour définir la qualité d'un document par rapport à un utilisateur u et un mot m, on suppose que le sens du mot m dépend des documents d de u qui contiennent le mot m. Si un document d' est sémantiquement proche de d, le document d' est sémantiquement pertinent pour u. De la même façon, si un ou plusieurs utilisateurs possèdent les documents d et d', il y a de grande chance que d' soit aussi pertinent pour u.

QUALITÉ D'UN DOCUMENT : Nous définissons la qualité d'un nœud n pour un document

d et pour une requête q :

$$qualite(n \to d, q) = \sum_{d_s \in D_{n,q}} curiosite\_sem(d_s \to d) + curiosite\_soc(d_s \to d)$$

avec $D_{n,q}$ les documents connus par n et convenant à la requête q.

En utilisant la valeur de qualité, on peut déterminer le document que va télécharger l'utilisateur. Pour cela, on calcule la valeur de qualité de tous les documents D présents dans les réponses de recherche, puis on choisit avec une probabilité proportionnelle à la qualité du document, le document que l'utilisateur va télécharger :

$$proba\_telechargement(n, d, q, R) = \frac{qualite(n \to d, q)}{\sum_{d_i \in D} qualite(n \to d_i, q)}$$

4.4.2.3 Les arrivées des utilisateurs

Les arrivées des utilisateurs dans le système vont être d'autant plus rapide que la valeur du système augmente. Un réseau a d'autant plus de valeur qu'il y a d'une part, des utilisateurs connectés et donc de documents disponibles, et d'autre part, que l'utilisateur trouve d'autres utilisateurs ayant des profils similaires du sien et donc capable de lui fournir des réponses pertinentes. Or dans ce type de réseau où des groupes peuvent se former, la valeur d'un réseau suit la loi de Reed et croit de manière exponentielle avec le nombre d'utilisateurs.

VALEUR D'UN RÉSEAU

Plusieurs mesures sont proposées pour quantifier la valeur d'un réseau et dépendent de la nature des interactions entre les nœuds du réseau.

Loi de Sarnoff La loi de Sarnoff mesure la valeur d'un système de diffusion (radio, télévision, ...). La valeur du réseau augmente proportionnellement avec N le nombre d'auditeurs.

Loi de Metcalfe La loi de Metcalfe stipule que la valeur d'un système de communication (téléphone, email, ...) augmente avec le carré du nombre d'utilisateurs N du système. Cette valeur représente le nombre de pairs de connexions possibles dans le réseau, soit $N(N-1)$ connexions.

Loi de Reed La loi de Reed [116] mesure la valeur d'un réseau où des groupes peuvent se former (site de vente d'enchère, salon de discussion, ...). Sa valeur représente le nombre de groupes possibles soit $2^N - N - 1$.

4.4.3 Modèle réduit

La mémoire et le CPU étant limités, nous avons du restreindre le nombre d'utilisateurs simulés dans le système (*e.g.* 1500). En effet, chaque utilisateur gère une liste de profils de voisin et de document coûteux en mémoire. Mais cette restriction entraîne plusieurs problèmes. Les réseaux n'exhibent pas de propriétés clairement *small world*. Deux utilisateurs partageant un même fichier n'ont pas plus de chance que deux utilisateurs pris au hasard de partager un autre fichier en commun. De même que deux utilisateurs ayant un mot en commun dans leur vocabulaire n'ont pas une grande probabilité de partager un autre mot.

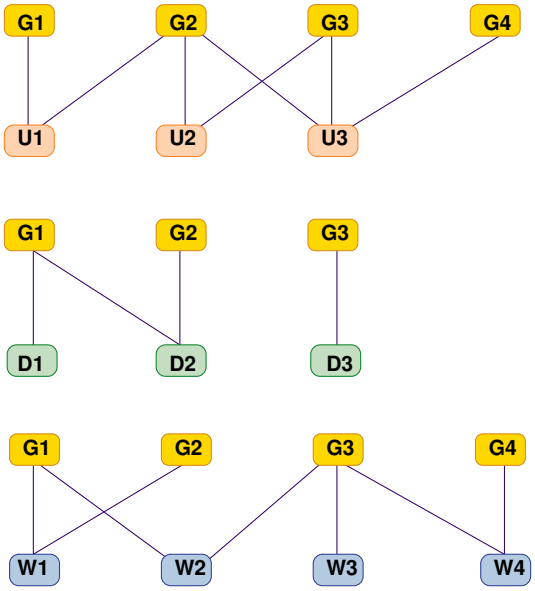


FIG. 4.16: Modèle des utilisateurs par groupe

Pour augmenter cette probabilité, on répartit, les documents, mots et utilisateurs par groupe. Les utilisateurs, documents et mots sont répartis dans plusieurs groupes. Ces groupes peuvent être vus comme des groupes sociaux, géographiques ou culturels ayant chacun leurs propre lexique, membres et documents.

Des groupes vont être plus important que d'autre. Sur le web, il y a beaucoup plus de documents écrits en anglais qu'en espagnol ou en français. C'est aussi le cas pour les utilisateurs d'Internet. Les groupes sont donc créés avec des tailles différentes. Ces groupes ne sont pas exclusifs, des utilisateurs, documents et mots peuvent appartenir à plusieurs groupes et vont favoriser prioritairement l'anglais. Les utilisateurs vont posséder des documents qui appartiennent à leurs groupes. Et les documents vont être composés de mots de leurs groupes.

Au sein de chaque groupe, les relations entre les utilisateurs, documents et mots ont les mêmes propriétés que dans le modèle précédent.

4.4.4 Mesures de personnalisation des résultats

En utilisant la valeur de qualité présentée précédemment, nous pouvons définir un ensemble de mesures pour évaluer l'efficacité du système à donner des réponses pertinentes et personnalisées à ses utilisateurs.

ENSEMBLE DES DOCUMENTS K-PERTINENTS : Nous définissons l'ensemble $P_k(u, q)$ des documents $k-pertinents$ à une requête de recherche q POUR un utilisateur u par l'ensemble des documents ayant les k meilleurs valeurs de qualité $qualite(u \to d, q)$.

Nous définissons les mesures de $k-precision$ et de $k-rappel$ pour étendre les mesures classiques de précision et de rappel et tenir compte de la personnalisation :

K-PRÉCISION : La $k-precision$ mesure le ratio de documents $k-pertinents$ retournés à l'utilisateur u par le système de recherche pour une requête q sur le nombre de résultats reçus :

$$k-precision(u, q) = \frac{P_k(u, q) \cap R(q)}{R(q)}$$

avec $P_k(u,q)$ l'ensemble des documents k-pertinents pour l'utilisateur u et une requête q, et $R(q)$ l'ensemble des documents retournés par le système de recherche.

K-RAPPEL : Le k-rappel mesure le ratio de documents k-pertinents retournés à l'utilisateur u par le système de recherche pour une requête q sur le nombre de documents k-pertinents :

$$k - rappel(u,q) = \frac{P_k(u,q) \cap P(q)}{P_k(u,q)}$$

avec $P_k(u,q)$ l'ensemble des documents k-pertinent pour l'utilisateur u et une requête q et $R(q)$ l'ensemble des documents retournés par le système de recherche.

L,K-SATISFACTION DU SYSTÈME : Nous définissons la $l,k - satisfaction$ du système qui mesure le ratio des requêtes dont les réponses contiennent au moins l documents $k - pertinents$ pour les utilisateurs qui les ont émises.

4.5 Conclusion

Nous avons décrit dans la première section, les caractéristiques des utilisateurs à travers des traces de système de recherche et nous avons étudié des modèles souvent utilisés pour simuler des utilisateurs d'échange de fichiers. Cependant ces modèles sont un cas particulier de système de recherche et se révèlent insuffisant pour simuler des utilisateurs de moteur de recherche textuelle.

Dans le seconde section, nous avons proposé un modèle des utilisateurs en se basant sur diverses études. Nous y avons tout d'abord présenté le modèle statique en définissant les relations entre les utilisateurs, documents et mots. Puis nous avons exposé les différentes actions d'un utilisateur de moteur de recherche, *i.e.* ses recherches, ses téléchargements et son arrivée dans le système tout en tenant compte des relations du modèle que nous avions défini. Nous avons introduit deux mesures pour évaluer l'efficacité d'un système de recherche à donner des résultats personnalisés à ses utilisateurs.

Tout cela nous permet de pouvoir simuler un ensemble d'utilisateurs de moteurs de recherche pour tester l'efficacité d'un système de recherche textuelle. C'est ce que nous allons faire en simulant des utilisateurs de notre système MAAY dans le prochain chapitre.

Chapitre 5

Simulation

Pour tester notre système, nous avons écrit un programme permettant de faire tourner des milliers de nœuds MAAY sur une machine. Les utilisateurs de ce système sont simulés en utilisant le modèle des utilisateurs que nous avons défini dans le chapitre précédent.

Dans ce chapitre, nous commencerons par décrire l'instanciation du modèle des utilisateurs et le fonctionnement du simulateur des utilisateurs. Puis nous détaillerons les paramètres utilisés pour la simulation des utilisateurs et pour le fonctionnement des nœuds MAAY. Ensuite nous étudierons les résultats que nous avons obtenus par les simulations, notamment sur la pertinence personnalisée des résultats. Enfin, nous terminerons en étudiant l'influence des paramètres du nœud MAAY sur l'efficacité du système.

5.1 Instanciation du modèle des utilisateurs

Instancier le modèle des utilisateurs, c'est construire un ensemble d'utilisateurs, de documents et de mots liés entre eux par des relations qui vérifient celles du modèle. Nous allons expliquer dans cette section une méthode permettant d'instancier le modèle d'utilisateur que nous avons défini précédemment.

Pour cela, nous commençons par construire les groupes, chacun contenant un ensemble d'utilisateurs, documents et mots. Ensuite nous définissons le contenu des documents en les associant à des mots appartenant à leurs groupes. Enfin nous établissons les relations entre les utilisateurs et les documents en tenant compte de leurs groupes respectifs.

Construction des groupes Connaissant le nombre de documents, utilisateurs et mots par groupe, on répartit les documents, utilisateurs et mots dans les différents groupes. Des utilisateurs, documents et mots peuvent appartenir à plusieurs groupes et certains groupes seront privilégiés.

On définit ensuite la popularité $p_{d,g}$ d'un document d dans le groupe g. Plus la popularité est élevée, plus un utilisateur du groupe g aura de chance d'avoir le document d dans ses connaissances. Cette popularité suit une loi exponentielle au sein de chaque groupe. Un document appartenant à deux groupes peut être populaire dans l'un et insignifiant dans l'autre. Pour générer ces popularités, nous utilisons un générateur de nombres aléatoires suivant une loi exponentielle (voir encadré).

De même, on définit la popularité $p_{m,g}$ d'un mot m dans le groupe g. Plus la popularité est élevée, plus le mot m aura de chance d'apparaître au sein d'un document appartenant au groupe g. La popularité des mots suit également une loi exponentielle.

Génération de variable aléatoire suivant une loi de puissance

Nous allons décrire comment effectuer un tirage d'un nombre suivant une loi de puissance. La loi de puissance est décrite par une valeur minimale m, une valeur maximale M et une valeur moyenne moy.

1. Nous commençons par déterminer la fonction de densité f de probabilité de la variable aléatoire. La fonction de densité f s'écrit sous la forme :

$$f(x) \approx a.x^p$$

avec a et p deux constantes à déterminer.
On sait que la fonction de répartition F vérifie ces propriétés :

$$\begin{cases} F(x) = \int_m^M f(x).dx \\ F(m) = 0 \\ F(M) = 1 \end{cases}$$

De ces propriétés, on en déduit que :

$$a = \frac{p+1}{M^{p+1} - m^{p+1}}$$

et donc on a :

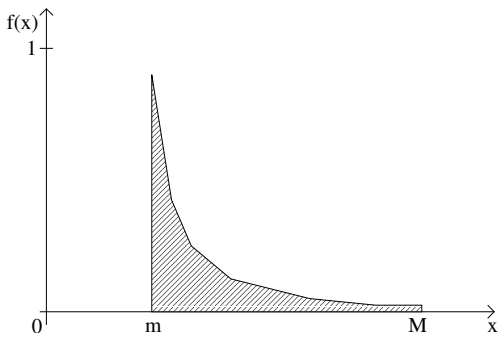
$$f(x) = \frac{p+1}{M^{p+1} - m^{p+1}} x^p$$

L'espérance E de f sur l'intervalle [m,M] donne la valeur moyenne de la v.a. :

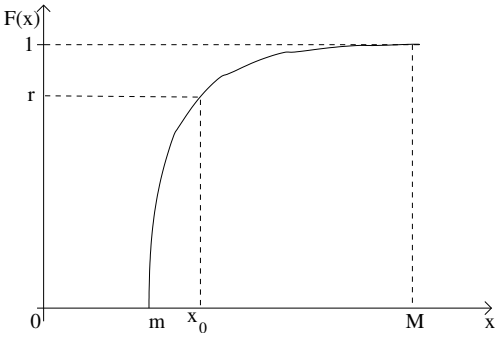
$$\begin{aligned} E &= \int_m^M x.f(x).dx \\ &= \frac{(p+1)(M^{p+2} - m^{p+2})}{(p+2)(M^{p+1} - m^{p+1})} \end{aligned}$$

Grâce à cette valeur moyenne, on peut déduire une valeur de p tel que $E = moy$.

2. Pour tirer aléatoirement un degré suivant une distribution de densité f, on utilise la méthode de transformation inverse.



Fonction de densité



Fonction de répartition

L'idée est de tirer un nombre r de manière uniforme entre 0 et 1 puis de déterminer la valeur de x tel que $F(x) = r$. Cela revient à choisir un point de manière uniforme dans l'aire délimitée par la fonction f et l'axe des abscisses puis de déterminer l'abscisse du point. Une valeur de x pour laquelle $f(x)$ est élevée a une plus grande probabilité

d'être choisie.
Le calcul nous donne :

$$x = \sqrt[p+1]{r.(M^{p+1} - m^{p+1}) + m^{p+1}}$$

Construction des relations document-mot Pour construire les relations entre les documents et les mots, nous associons les documents à des mots appartenant au même groupe que le document.

Les documents ont un nombre donné h de mots. Pour déterminer les mots du document d, on procède à h tirages. A chaque tirage, le groupe g est choisi de manière uniforme parmi ceux du document d. Puis un mot m du groupe g est choisi en fonction de sa popularité $p_{m,g}$ et est ensuite associé au document d.

Construction des relations utilisateur-document Pour construire les relations entre les utilisateurs et leurs documents, nous associons chaque utilisateur à des documents appartenant à un des groupes dont l'utilisateur est membre.

Pour cela, on commence par assigner à chaque utilisateur u le nombre de documents k_u qu'il va connaître. La distribution du nombre de documents par utilisateur suit une loi exponentielle. Puis pour choisir ces k_u documents, on procède à k_u tirages. A chaque tirage, le groupe g du document est choisi de manière uniforme parmi ceux auxquels appartient l'utilisateur. Puis un document d du groupe g est choisi suivant sa popularité $p_{d,g}$. Il est ensuite associé à l'utilisateur u.

5.2 Simulation

Chaque nœud MAAY est implémenté par une structure de donnée qui contient les données propres au nœud : les documents sauvegardés sur son disque, le profil de ses voisins, le profil de ses documents, les recherches en cours et une boite de réception contenant les messages reçus.

Lors de la simulation, les utilisateurs lancent des requêtes de recherche, de téléchargement, se joignent au système et traitent les messages reçues. Pour simuler ces différents évènements, le simulateur exécute les actions suivantes régulièrement avec une périodicité différente pour chacune d'entre elles :

Traitement des messages Un nœud est choisi aléatoirement pour traiter les messages reçus dans sa boite de messages. En traitant les messages, le nœud peut envoyer d'autres messages à ses voisins. Ainsi, par exemple, en traitant un message de recherche, le nœud peut le retransmettre à plusieurs de ses voisins et renvoyer un message de réponse à l'émetteur de la requête. Ces messages sont envoyés en les déposant dans la boîte de messages du destinataire. Après avoir traité un message, celui-ci est détruit.

Émission de requête de recherche Un utilisateur est choisi parmi ceux connectés au système pour lancer une recherche. Après avoir construit sa requête de recherche, il se l'envoie localement dans sa boîte de messages. La requête de recherche sera examinée lorsque le nœud traitera les messages qu'il a reçu.

Émission de requête de téléchargement Après avoir lancé une requête de recherche et reçu un nombre suffisant de réponses, l'utilisateur choisit parmi celles reçues, un document à télécharger suivant sa qualité.

Publication de documents Un utilisateur u est choisi aléatoirement pour publier un nouveau document. La publication s'effectue simplement en insérant le document dans la liste des documents de u.

Arrivée des utilisateurs Pour connecter un nœud au système, on lui attribue deux voisins choisis aléatoirement parmi les nœuds déjà présents dans le réseau.

5.3 Paramètres de la simulation

5.3.1 Paramètres du modèle des utilisateurs

Paramètres du modèle statique Les paramètres du modèles vont dépendre du nombre d'utilisateurs que l'on va pouvoir simuler sur une machine. Contrairement à un système à la Gnutella, les nœuds doivent maintenir le profil de leurs voisins, de leurs documents et classer leurs voisins et leurs documents en fonction de la requête de recherche et de son émetteur. Cela demande donc une charge de calcul et une quantité de mémoire plus importante. Nous avons fixé le nombre d'utilisateurs à 1500 et choisi les valeurs des autres paramètres en essayant de garder des proportions cohérentes.

Utilisateurs	1500
Documents	3000
Mots	500
Mots par document	5
Documents par mot	1-1000 (moy : 30, dist. en loi de puissance)
Utilisateurs par document	1-200 (moy :2.5, dist. en loi de puissance)
Documents par utilisateur	1-40 (moy :5, dist. en loi de puissance)

Nous avons défini 5 groupes : A, B, C, D et E. Les groupes ont une popularité décroissante avec A le groupe ayant le plus de documents, utilisateurs et mots et le groupe E celui qui en a le moins. Des utilisateurs, documents et mots vont appartenir à deux groupes. 5% des utilisateurs, documents et mots des groupes B, C, D, E vont appartenir à un autre groupe. Le groupe additionnel choisi est majoritairement le groupe A qui est sélectionné dans 95% des cas. Les autres groupes se partagent de manière uniforme les 5% des cas restants.

En prenant 5 mots par documents, chaque mot apparaît en moyenne dans 30 documents. Le mot le plus populaire y apparaît 717 fois et le mot les moins populaire 1 fois.

La répartition des utilisateurs, documents et mots dans les groupes est décrite dans le tableau suivant.

Groupe	Nb. d'utilisateurs	Nb. de documents	Nb. de mots
A	500	1000	100
B	400	800	100
C	300	600	100
D	200	400	100
E	100	200	100

Paramètres du comportement des utilisateurs Le comportement des utilisateurs décrit leurs interactions avec le système, c'est-à-dire leurs recherches, leurs publications et leur arrivée dans le système. La fréquence d'émission de requête de recherche, de publication de document et d'arrivée des utilisateurs dans le système sont décrites dans le tableau ci-dessous.

Fréquence d'émission de requête de recherche	1 / 100 ticks / utilisateur
Fréquence de publication	1 / 25000 ticks / utilisateur
Fréquence d'arrivée des utilisateurs	1 / 50000 ticks / utilisateur
Mots utilisés par requête	1-3 [34% : 1, 41% : 2, 25% : 3]

(Un tick correspond au temps entre le moment où un nœud est choisi par le programme pour analyser les messages qu'il a reçu et la prochaine fois où il est choisi.)

5.3.2 Paramètres des nœuds Maay

Paramètres sur les profils Les nœud MAAY doivent être paramétrés avec soin puisqu'ils vont définir l'efficacité du système. D'un coté, des paramètres peuvent en ayant une valeur élevée favoriser l'efficacité du système. D'un autre coté, ces valeurs doivent garder un ordre de grandeur cohérent avec le nombre d'utilisateurs, documents et mots présents dans le réseau et donc rester relativement peu élevées.

Profils partiels de document	20
Doc. téléchargés mis en cache et leur profil complet	20
Profils de voisin géré par nœud	50
Politique de choix des documents publiés	aléatoire uniforme
Politique de choix des utilisateurs arrivants	aléatoire uniforme
Politique d'éviction de profils de voisin	plus petite valeur d'affinité avec le nœud
Politique d'éviction de profils de document	plus petit valeur d'adéquation avec le nœud

Ainsi, le nombre de profils de document gérés par nœud ne représente que 1.33% de tous les documents. De même que le nombre de profils de voisin gérés par nœud qui ne représente que 3.33% de tous les nœuds.

Propagation des requêtes de recherche La propagation d'une requête de recherche est contrôlée par les paramètres `TTL` et `FNC`. Le paramètre contrôle le nombre de retransmission du message : à chaque retransmission il est décrémenté de 1, quand celui-ci vaut 0, le message n'est plus retransmis. Le `FNC` contrôle le de voisins vers lesquels le nœud va retransmettre le message : le nœud transmet le message à `FNC` voisins puis les nœuds recevant le message divise sa valeur par 2 avant de retransmettre le message. En fixant le `TTL` initial à 2 et le `FNC` initial à 4, seuls 12 nœuds sont interrogés par requête : le nœud initiateur envoie sa requête à 4 voisins qui la retransmettent à leur tour à 2 de leurs voisins. Ces nœuds interrogés ne représente que 0.8% de tous les nœuds.

`TTL` initial	2
`FNC` initial	4

Nombre de réponses attendues Le nombre moyen de résultats par requête étant de 35, nous avons choisi de limiter le nombre de résultats (hits) reçu à 5. Cela permet de vérifier si le classement des résultats est efficace.

5.4 Résultats

Nous avons effectué nos simulations en utilisant un ordinateur doté de 2 processeurs Intel Xeon cadencés à 2.80 Ghz. Les simulations prennent chacune plus de 300 Mo en mémoire vive et demandent en moyenne 1 à 2 journées de calcul avant de fournir les résultats.

5.4.1 Voisinage

Topologie du réseau / Apprentissage Le premier résultat attendu est que les nœuds s'agrègent par profil commun.

Nous allons le montrer en utilisant l'outil GraphOpt[1]. Cet outil permet de mettre en évidence la présence d'amas dans des graphes en utilisant des principes physiques d'attractions et de répulsions. Si la topologie du réseau des nœuds MAAY liés par affinité présente des similitudes avec la topologie du réseau des utilisateurs liés par document en commun défini dans le modèle, alors on peut en déduire que le réseau MAAY a appris les relations entre les utilisateurs, documents et mots et a su s'adapter aux profils des différents utilisateurs.

Pour le vérifier, nous construisons le graphe des utilisateurs en liant chaque utilisateur avec les 5 autres utilisateurs avec lesquels il a le plus de documents en commun. La représentation graphique de ce graphe montre clairement la présence de 5 amas (voir Figure 5.1) correspondant aux 5 groupes. La taille des amas concorde avec la taille des groupes.

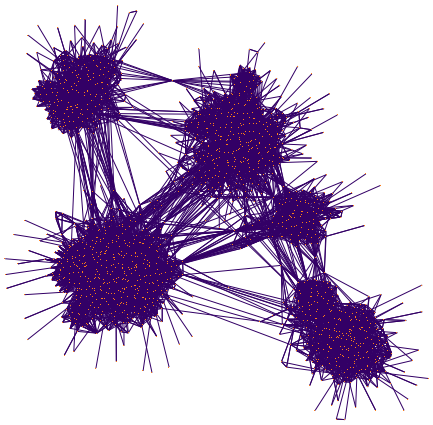


FIG. 5.1: Graphe des utilisateurs liés par document en commun

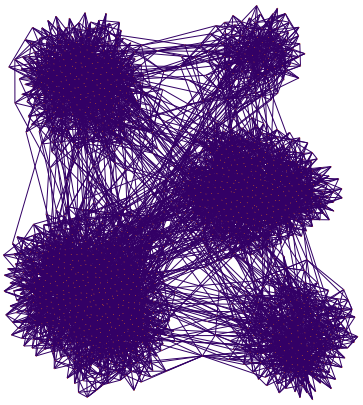


FIG. 5.2: Graphe des nœuds liés par affinité

A la fin de la simulation, quand tous les nœuds sont connectés au système, que tous les documents ont été publiés et que les nœuds ont pu apprendre le profil de leurs voisins, nous construisons le graphe d'affinité des nœuds en liant les nœuds avec leurs 5 voisins avec lesquels ils ont localement mesuré la plus grande valeur d'affinité. Ce graphe est représenté dans la Figure 5.2.

Les 5 amas que nous apercevons dans la figure 5.2 nous montrent que les nœuds se sont agrégés par groupe. La taille de ces groupes coïncident avec celles des groupes d'utilisateurs que nous avons définies. Les liens entre les nœuds appartenant à des groupes différents s'expliquent par le fait que des utilisateurs, documents et mots appartiennent à plusieurs groupes.

[1] GraphOpt. `http://www.schmuhl.org/graphopt`

Le système MAAY a donc réussi à apprendre les profils de ses utilisateurs et à adapter la topologie de son réseau pour agréger les nœuds des utilisateurs ayant un profil sémantique similaire.

Désambiguïsation de groupe L'observation précédente peut être confortée en mesurant pour chaque nœud le ratio des 10 voisins avec lesquels le nœud a mesuré le plus d'affinité et qui appartiennent au même groupe que le nœud. La mesure de 93% confirme bien le fait que les nœuds s'agrègent par profil sémantique proche.

5.4.2 Pertinence des réponses de recherche

Nous allons décrire les différentes mesures que nous avons effectuées pour mesurer la pertinence et la personnalisation des résultats.

Désambiguïsation de groupe Des mots peuvent appartenir à plusieurs groupes et peuvent être contenus dans des documents appartenant à différents groupes. Un utilisateur effectuant une requête de recherche sur un de ces mots sera satisfait si les réponses qu'il reçoit appartiennent au même groupe que le sien. Pour déterminer si les résultats sont appropriés à l'utilisateur, on mesure pour les requêtes de recherche portant sur des mots ambigus, *i.e.* appartenant à plusieurs groupes, le pourcentage de réponses reçues appartenant au même groupe que le demandeur. Nous avons trouvé que 85% des réponses retournées sont dans le même groupe que le demandeur ce qui confirme notre hypothèse.

5-précision / 5-rappel En utilisant la définition de $k - precision$ et de $k - rappel$ (voir 4.4.4), nous avons mesuré une $5 - precision$ de 0.6 et un $5 - rappel$ de 0.57. Cela signifie que toutes les réponses de recherche contiennent en moyenne 60% des 5 documents les plus pertinents pour l'utilisateur et qu'en moyenne 57% des 5 documents les plus pertinents sont présents dans les réponses de recherche.

l,5-satisfactions Pour mesurer plus précisément la satisfaction de l'utilisateur, nous utilisons la mesure de $l, k - satisfaction$ que nous avons définie dans la sous-section 4.4.4. La $l, k - satisfaction$ mesure le ratio des recherches où l'utilisateur a reçu au moins l résultats parmi les k plus pertinents pour lui. Nous avons mesuré, à la fin de la simulation, une 1,5-satisfaction de 84% des requêtes, une 2,5-satisfaction de 64% et une 3,5-satisfaction de 41%.

Comparaisons Pour mesurer l'efficacité de la sélection de voisins et du classement des résultats, nous avons comparé le système MAAY avec une version modifiée où le classement des documents et/ou la sélection des voisins étaient désactivés. Le tableau 5.1 résume les $l, 5 - satisfactions$ obtenus avec les versions modifiées pour $l = 1$ et $l = 2$.

politique de classement de documents	politique de sélection de voisins	$1,5 - sat.$	$2,5 - sat.$	gain
aucune	aucune	0.20	0.02	-
basé sur le profil	aucune	0.27	0.03	x 1.35
aucune	basé sur le profil	0.70	0.45	x 3.50
basé sur le profil	basé sur le profil	0.85	0.64	x 4.25

TAB. 5.1: Comparaison entre MAAY et ses versions modifiés sans classement des documents et/ou sélection de voisins

On observe que le classement des documents est important puisqu'il permet d'obtenir des gains de 1.35 par rapport à un système qui en est dépourvu. Cependant, la sélection des voisins est plus primordial puisqu'il permet d'avoir 3.5 fois plus de résultats 1-satisfait que sans. Enfin, il est intéressant de remarquer que les deux sont complémentaires.

Réactivité Pour mesurer si MAAY est capable de s'adapter à un changement d'intérêt de ses utilisateurs, nous avons changé la politique de sélection des mots pour 10% des utilisateurs et nous avons mesuré les valeurs de $l, 5 - satisfactions$ pour $l = 1$, $l = 2$ et $l = 3$. Ces mesures sont résumées dans le tableau 5.2.

La politique de sélection de mots dans les requêtes de recherche dépendent de deux facteurs : la fréquence du mot dans le corpus de document de l'utilisateur et la fréquence du mot dans tous les documents. Ainsi, un utilisateur peut privilégier les mots qui apparaissent couramment dans son corpus de document ou au contraire privilégier les mots peu fréquents, ou encore choisir ces mots de manière équiprobable. Il peut aussi effectuer son choix en fonction de la fréquence du mot dans l'ensemble des documents.

fréquence du mot dans le corpus de l'utilisateur	fréquence du mot dans tous les documents	$1,5 - sat.$	$2,5 - sat.$	$3,5 - sat.$
-	-	0.74	0.45	0.24
populaire	-	0.76	0.42	0.22
rare	-	0.75	0.47	0.26
-	populaire	0.66	0.27	0.10
-	rare	0.86	0.64	0.39
populaire	populaire	0.82	0.48	0.21
populaire	rare	0.86	0.61	0.38
rare	populaire	0.69	0.28	0.11
rare	rare	0.85	0.63	0.39

TAB. 5.2: Réactivité du système MAAY suivant la politique de sélection de mots

Ces résultats montrent que le système MAAY apprend les relations entre les utilisateurs, documents et mots et arrive à les exploiter pour fournir à ses utilisateurs des résultats personnalisés, c'est-à-dire qui tiennent compte du contexte social et sémantique de l'utilisateur. Cependant, du fait que les paramètres des nœuds aient été choisis de manière intuitive, il peut être intéressant de tester leurs influences sur la pertinence des réponses.

5.4.3 Influences des paramètres de recherche

Pour mesurer l'influence des paramètres du nœud MAAY sur l'efficacité du système, nous allons effectuer plusieurs simulations en modifiant la valeur de quelques paramètres du nœud MAAY.

Nombre maximal de résultats par réponse de recherche (EHC) Le tableau ci-dessous montre la pertinence des réponses suivant le nombre maximum `EHC` de résultats qui peuvent être renvoyés à un nœud.

EHC	3	5	10	20
$5 - precision$	0.63	0.60	0.63	0.61
$5 - rappel$	0.52	0.57	0.71	0.79
Ratio de voisins appartenant au même groupe	0.91	0.93	0.92	0.92
Ratio de réponses appartenant au même groupe	0.85	0.85	0.84	0.85
$1,5 - satisfaction$	0.76	0.84	0.90	0.93
$2,5 - satisfaction$	0.53	0.64	0.75	0.83
$3,5 - satisfaction$	0.31	0.41	0.60	0.71
Nb. moyen de résultats reçus	2.5	3.5	5.8	9
Nb. total de doc. convenant à la req.	36	36	36	36
Nb. de messages de req. par recherche	13	13	13	13

On remarque que plus cette valeur est élevée, plus la satisfaction est élevée. En effet, l'utilisateur récupère plus de résultats et a donc plus de chance de voir apparaître les résultats pertinents. En revanche, il est intéressant de remarquer que le ratio de réponses pertinentes reçues par requête (la précision) n'a pas baissé malgré un nombre moyen de résultat reçu plus important.

Nombre de retransmissions de la requête (TTL) Le tableau ci-dessous montre la pertinence des réponses suivant les valeurs de `TTL` initiaux.

TTL initial	1	2	3	4	5	6
$5 - precision$	0.46	0.60	0.63	0.63	0.63	0.63
$5 - rappel$	0.36	0.57	0.60	0.61	0.61	0.62
Ratio de voisins appartenant au même groupe	0.87	0.93	0.93	0.93	0.92	0.92
Ratio de réponses appartenant au même groupe	0.80	0.85	0.84	0.84	0.83	0.83
$1,5 - satisfaction$	0.63	0.84	0.85	0.85	0.84	0.85
$2,5 - satisfaction$	0.49	0.64	0.63	0.64	0.64	0.64
$3,5 - satisfaction$	0.30	0.41	0.44	0.45	0.45	0.45
Nb. moyen de résultats reçus	2.6	3.5	3.5	3.5	3.6	3.6
Nb. total de doc. convenant à la req.	36	36	36	36	36	36
Nb. de messages de req. par recherche	4.9	13	20.3	27	33	39

On peut remarquer que le `TTL` a peu d'influence quand celui-ci devient supérieur à 3. Ceci s'explique notamment par le fait que plus une requête de recherche s'éloigne, moins le profil du nœud interrogé est proche de celui de l'initiateur de la requête et donc a moins de chance d'avoir des documents intéressants pour l'initiateur.

Nombre de voisins vers lesquels un nœud retransmet la requête (`FNC`) Le tableau ci-dessous montre la pertinence des réponses suivant les valeurs de `FNC` initiaux.

FNC initial	1	2	4	8	16	32
$5 - precision$	0.38	0.54	0.60	0.65	0.65	0.65
$5 - rappel$	0.24	0.45	0.57	0.63	0.64	0.64
Ratio de voisins appartenant au même groupe	0.85	0.92	0.93	0.94	0.94	0.94
Ratio de réponses appartenant au même groupe	0.79	0.84	0.85	0.85	0.87	0.87
$1,5 - satisfaction$	0.46	0.70	0.84	0.87	0.87	0.87
$2,5 - satisfaction$	0.22	0.47	0.64	0.66	0.66	0.66
$3,5 - satisfaction$	0.12	0.31	0.41	0.47	0.48	0.48
Nb. moyen de résultats reçus	1.7	2.9	3.6	3.6	3.7	3.7
Nb. total de doc. convenant à la req.	36	36	36	36	36	36
Nb. de messages de req. par recherche	3	5	13	41	143	510

On remarque qu'à partir de 8, augmenter la valeur de `FNC` est inutile. Ceci s'explique par le fait que le nombre de nœuds ayant un profil proche de celui de l'initiateur donc qui ont les documents qui l'intéressent, est limité et qu'il ne sert à rien d'interroger plus de nœuds.

Taille du cache des documents Nous avons limité le nombre de documents téléchargés qu'un nœud peut stocker sur son disque ainsi que le nombre de profils partiels de document que peut gérer un nœud. Quand le cache est plein, les documents et les profils partiels des documents ayant les valeurs d'adéquation les moins élevées avec le profil du nœud sont évincés en priorité.

Nous avons fait varier la taille du cache des documents et celui des profils partiels de document de 20 à 160.

Taille du cache de doc. + profils partiels	20+20	40+40	80+80	160+160
$5 - precision$	0.60	0.65	0.68	0.68
$5 - rappel$	0.57	0.63	0.67	0.67
Ratio de voisins appartenant au même groupe	0.93	0.94	0.94	0.94
Ratio de réponses appartenant au même groupe	0.85	0.85	0.85	0.85
$1,5 - satisfaction$	0.84	0.88	0.90	0.91
$2,5 - satisfaction$	0.64	0.67	0.73	0.73
$3,5 - satisfaction$	0.41	0.48	0.52	0.52
Nb. moyen de résultats reçus	3.5	3.5	3.7	3.7
Nb. de doc. répondant à la req.	36	36	36	36
Nb. de messages de req. par recherche	13	13	13	13

Nous observons qu'augmenter la taille du cache permet d'augmenter la satisfaction des recherches pour les utilisateurs. Cependant l'espace disque alloué pour le stockage des documents et l'espace mémoire utilisé est plus important.

5.4.4 Bilan

Nous avons mesuré l'influence des paramètres de propagation des requêtes et l'influence de la taille du cache des documents sur l'efficacité du système MAAY.

Nous avons ainsi pu observer que le paramètre décrivant le nombre maximum de résultats par réponse (`EHC`) permettait d'augmenter la qualité des résultats quand on augmentait sa valeur, cependant elle se fait au prix d'une taille des messages de réponse plus importante.

Les paramètres décrivant le nombre de retransmissions de la requête et le nombre de nœuds vers lesquels un nœud retransmet une requête de recherche n'augmentaient pas la qualité des résultats quand ils étaient au dessus d'un certain seuil. Cela peut s'expliquer par

deux facteurs. Premièrement, une requête qui s'éloigne de son initiateur i a moins de chance d'atteindre un nœud ayant un profil similaire à i et donc d'avoir des documents pertinents pour lui. Deuxièmement, le nombre de nœuds ayant un profil similaire à i est limité, il est donc inutile d'interroger un plus grand nombre de nœuds.

Enfin, nous avons observé que la qualité des résultats augmentait avec la taille du cache des documents. Les nœuds ayant un profil similaire à l'initiateur de la requête peuvent lui fournir plus de réponses donc l'initiateur aura plus de chance de voir sa requête satisfaite. Cependant, cela se fait au détriment d'un espace mémoire et d'un espace de stockage utilisés plus importants.

5.4.5 Modèle basé sur le graphe des articles DBLP

Nous avons utilisé dans les simulations précédentes, des utilisateurs, documents et mots que nous avons générés à partir du modèle des utilisateurs que nous avons défini.

Pour avoir un modèle plus « réaliste », nous avons construit des utilisateurs à partir de la base de donnée DBLP contenant la liste des articles publiés dans diverses conférences. Nous avons sélectionné un ensemble de 60 conférences appartenant à 7 domaines : intelligence artificielle, hardware et architecture, applications, technologie système, langage de programmation et conception de logiciels, algorithmes et théorie, base de données. Les conférences ont été sélectionnées à partir d'une liste présentant un classement non officiel par domaine des conférences en informatique [2], seules les conférences de rang 1 ont été choisies.

Chaque article est décrit par son titre, ses auteurs et la conférence à laquelle il a été publié. Les utilisateurs, documents et mots du système sont construits en considérant les auteurs de publications comme les utilisateurs du système, leurs publications comme leurs connaissances, le titre de leurs articles comme les documents et enfin les domaines des conférences comme les groupes. Les documents appartiennent aux mêmes groupes que les conférences dans lesquelles ils sont publiés. Les auteurs appartiennent aux groupes auxquels appartiennent les documents qu'ils ont écrits. Enfin, les mots appartiennent aux mêmes groupes que les documents dans lesquels ils apparaissent.

Cependant, les nombres d'utilisateurs, de documents et de mots ainsi obtenus sont trop importants : 45158 utilisateurs, 51954 documents et 18917 mots. Nous avons donc réduit ces ensembles en excluant d'une part les auteurs ayant moins de 2 publications, les documents ayant moins de 2 auteurs, les documents ayant moins de 2 mots et les mots apparaissant dans moins de 5 documents. Nous sommes finalement parvenu à un ensemble faisant intervenir 1994 utilisateurs, 3494 documents et 859 mots, ce qui correspond à peu près au jeu de données que nous avons généré. La répartition des utilisateurs, documents et mots dans ces groupes est représenté dans le tableau suivant :

Groupe	Nb. d'utilisateurs	Nb. de documents	Nb. de mots
Intelligence Artificielle	501	371	808
Hardware et architecture	581	649	770
Application	496	179	791
Technologie système	377	126	730
Langage de programmation	353	126	757
Algorithmes et théories	738	998	817
Base de données	807	1046	811

Les résultats que nous avons obtenus sont présentés ci-dessous.

[2] *Computer Science Conference Rankings.* `http://www.cc.gatech.edu/people/home/guofei/CS_ConfRank.htm`

$5 - precision$	0.66
$5 - rappel$	0.60
Ratio de voisins appartenant au même groupe	0.76
Ratio de réponses appartenant au même groupe	0.57
$1,5 - satisfaction$	0.83
$2,5 - satisfaction$	0.69
$3,5 - satisfaction$	0.46
Nb. moyen de résultats reçus	2.6
Nb. total de doc. convenant à la requête	13.5
Nb. de messages de req. par recherche	13

Les résultats que nous obtenons sont comparables à ceux que nous avons mesurés avec nos utilisateurs générés. Cela permet de montrer qu'avec un modèle plus « réaliste », le système MAAY arrive à s'adapter aux profils de ses utilisateurs et à leur fournir des réponses pertinentes et personnalisées.

5.5 Conclusion

Nous avons présenté dans ce chapitre les différentes étapes qui nous ont permises de simuler les utilisateurs du système MAAY. Tout d'abord, nous avons détaillé la construction des utilisateurs, des documents et des mots à partir du modèle que nous avons défini précédemment. Puis nous avons décrit les paramètres utilisés pour effectuer la simulation, *i.e.* les paramètres du modèle et les paramètres de fonctionnement des nœuds MAAY. Nous avons ensuite simulé des utilisateurs de MAAY lançant des requêtes de recherche, de téléchargement, publiant des documents et se joignant au réseau. Enfin nous avons effectué une série de mesures pour évaluer l'efficacité du système MAAY. Les résultats que nous avons obtenus sont prometteurs et montrent que le système arrive à fournir majoritairement des résultats pertinents et personnalisés à ses utilisateurs.

Le dernier test auquel nous allons confronter notre système est le déploiement lui-même. C'est ce que nous avons commencé à étudier en implémentant un prototype que nous allons décrire dans le chapitre suivant.

Chapitre 6

Mise en oeuvre

Le projet MAAY [1] est né au sein du laboratoire France Télécom Division R&D et a débuté en Octobre 2002. Le projet s'est déroulé en 3 phases. La première phase a consisté en une étude pour concevoir le système et valider les algorithmes utilisés. La deuxième phase a été le développement d'un prototype fonctionnel pour définir les fonctionnalités du système et son architecture. Enfin la dernière phase est le développement du système final, soit à partir du prototype fonctionnel, soit en le reprogrammant entièrement en profitant de l'expérience acquise, puis son déploiement sur l'Internet. Nous avons fini la deuxième phase et nous nous préparons à attaquer la troisième. Le développement de MAAY se fera sous licence GPL (GNU General Public License), cette licence stipule d'une part que le code source du programme et de toutes ses modifications soient librement accessibles et d'autre part que tout programme utilisant en partie ou intégralement le programme précédent soit assujetti à la même licence.

Nous avons utilisé le langage de programmation Python[2] pour implémenter notre système. Ce langage de script, très complet, possède une panoplie de bibliothèques qui le rend comparable en fonctionnalités aux autres langages comme Java, C ou C++. C'est surtout la clarté du langage, comparable à du pseudo-code, qui a motivé notre choix. Bien que ce langage soit encore peu connu du grand public, il commence à être de plus en plus utilisé, même au sein des entreprises. Par exemple, l'entreprise Google l'utilise dans son système GMail, Google Maps et dans son moteur de recherche. On peut aussi citer le célèbre programme BitTorrent[3] qui est développé en Python. Python est disponible gratuitement en *open source* et est disponible sur les principales plateformes dont Windows, Linux et Mac OS X.

Nous décrirons dans ce chapitre le prototype fonctionnel du système MAAY que nous avons implémenté. Nous commencerons par décrire les fonctionnalités du prototype. Puis nous détaillerons l'architecture du nœud MAAY en décrivant le fonctionnement de chacun des modules qui la compose. Ensuite, nous verrons comment l'utilisateur peut interagir avec son nœud MAAY en décrivant l'interface homme machine. Enfin, nous décrirons les méthodes qui permettront une installation aisée du système et son déploiement.

6.1 Fonctionnalités

Les utilisateurs gèrent deux espaces d'information : un espace privé et un espace public. L'espace privé correspond aux données que l'utilisateur ne désire pas partager avec les autres. Alors que l'espace public correspond aux données qu'il rend disponible aux autres.

[1] `http://maay.netofpeers.net/`

[2] Python. `http://www.python.org/`

[3] BitTorrent. `http://www.bittorrent.com`

Recherche locale (Desktop Search) La recherche locale permet à un utilisateur de rechercher dans son espace d'information local. Cet espace englobe l'espace privé et public de l'utilisateur et comprend les fichiers présents sur son disque dur.

Recherche collaborative (P2P Search) La recherche collaborative permet à un utilisateur de rechercher dans l'espace public des autres utilisateurs. Cette recherche est décrite dans le chapitre 3. L'espace public d'un utilisateur comprend les fichiers de son disque dur qu'il a intentionnellement rendu public, mais aussi les fichiers qu'il a téléchargés chez les autres utilisateurs.

Indexation des ressources du web Quand un utilisateur visite une page web, il peut choisir de l'indexer grâce à une extension du navigateur web. La page est sauvegardée sur son disque puis partagée. La mise en cache de ces pages permet de les rendre disponibles même lorsqu'elles ne sont plus accessibles sur leur serveur web. Autre avantage, plusieurs versions d'une même page peuvent être disponibles permettant ainsi la recherche dans le passé du Web. Enfin, les pages du *deep web* qui ne sont pas accessibles aux moteurs de recherche peuvent être trouvées sur le réseau MAAY.

Annotation (Tagging) L'annotation permet d'associer un ensemble de mots à une donnée dont le contenu textuel ne contient pas ces mots ou de renforcer les liens entre les mots et la donnée. L'annotation est particulièrement utile pour les fichiers multimédias : images, musiques, vidéos qui ne contiennent pas ou très peu de méta-données.

6.2 Travaux approchés

6.2.1 Recherche collaborative

Nous allons décrire dans cette section les systèmes largement utilisés qui présentent des similarités avec notre système.

YouServ YouServ[4] [24] est un système de recherche qui a été déployé dans l'Intranet d'IBM et utilisé par 3055 utilisateurs en fin 2002. Chaque utilisateur a un client YouServ qu'il peut utiliser pour publier des documents et effectuer des recherches. Les fichiers publiés sont accessibles *via* une URL de la même manière qu'une page web.

La recherche dépend d'un serveur de registre qui répertorie pour chaque client la liste des mots pour lesquels il a au moins un document publié qui le contient.

Pour effectuer une recherche sur des mots Q, un client YouServ commence par interroger le serveur de registre qui lui envoie un ensemble de clients possédant des fichiers convenant à Q. Puis, il envoie à chaque client une requête de recherche sur Q et récupère les résultats.

YACY YACY[5][48] est un moteur de recherche web distribué. Ce système utilise une table de hachage distribuée où chaque nœud est responsable de gérer une partie des index inversés des mots. Les utilisateurs peuvent *via* une interface web définir les sites web à parcourir et à indexer. Le parcours des pages et leur indexation peut se faire de manière locale ou distribuée. L'utilisateur peut choisir d'indexer les pages web qu'il visite en utilisant le proxy web local de YACY. La recherche s'effectue en interrogeant les nœuds responsables des mots de la requête.

[4] YouServ : a P2P (peer-to-peer) web hosting/File Sharing System. `http://www.almaden.ibm.com/cs/people/bayardo/userv/`.

[5] YaCy : P2P Web Search Engine. `http://yacy.net/yacy/`.

socialized.net socialized.net[6][48, 47] est un système aujourd'hui utilisé notamment pour la recherche de fichiers de vidéo et de musique. Ce système présente de nombreuses similarités avec le système MAAY, notamment pour ses mécanismes de routage. Le choix des voisins interrogés se base sur plusieurs critères : sémantique (le nœud a-t-il manifesté avoir une donnée contenant les mots de la requête ?), activité du nœud (a-t-on reçu un message/réponse de la part du nœud il y a peu de temps ?), la contribution du nœud (a-t-il répondu à beaucoup de nos requêtes, ... ?). Les différents critères proposés sont intéressants et pourraient être étudiés pour être utilisés ultérieurement dans les prochaines versions de MAAY. Cependant, l'indexation ne porte pas sur le contenu textuel du document mais seulement sur les mots associés par l'utilisateur sur la donnée. De plus, le système ne propose pas de classement des résultats par pertinence.

6.2.2 Bookmark collaboratif

del.icio.us del.icio.us[7] est un gestionnaire de bookmark collaboratif. Les utilisateurs peuvent soumettre une URL et l'annoter avec des mots clés. Chacun des mots clés peut être considéré comme une catégorie. Ces mots clés ne sont pas prédéfinis et sont choisis à la libre appréciation des utilisateurs. Cette approche, appelée folksonomy[8], désigne le fait de catégoriser les données par les utilisateurs eux-mêmes et non par les auteurs ni par un groupe restreint de personnes (experts, éditeurs, ...) comme il est souvent coutume. Le système propose une interface pour rechercher par mot clé les pages soumises par les autres utilisateurs.

StumbleUpon StumbleUpon[9] est une application qui permet de découvrir des pages web de manière collaborative. Le logiciel se présente sous la forme d'une extension pour le navigateur web. Cette extension permet d'une part à l'utilisateur d'indiquer s'il apprécie la page web qu'il est en train de visiter et d'y ajouter éventuellement un commentaire. D'autre part, il lui permet de découvrir des pages web en appuyant sur un bouton. Un serveur central stocke les appréciations des utilisateurs et leurs profils. Lorsqu'un utilisateur s'inscrit à StumbleUpon, il crée un compte et renseigne le système sur son profil en sélectionnant les catégories qui l'intéressent : films, nouvelles technologies, sciences, ... Lorsque l'utilisateur appuie sur le bouton de découverte, il est redirigé vers une page web choisie d'une part parmi celles appartenant aux catégories qu'il a sélectionnées et d'autre en privilégiant les pages web qui ont été appréciées par les utilisateurs ayant aimé les mêmes pages que lui.

Searchius Searchius[10][10] est un outil qui permet aux utilisateurs de rechercher par mot clé un site web parmi les bookmarks des autres utilisateurs. Cet outil est utilisé par plus de 36000 personnes et comptabilisent plus de 1.4 millions d'URLs. Les utilisateurs installent un logiciel qui envoie leurs bookmarks à un serveur central. Ce serveur propose une interface web permettant aux utilisateurs de rechercher une URL parmi les bookmarks soumis par les utilisateurs. Une URL convient à une requête s'il existe au moins un bookmark pour cette URL dont la description contient les mots de la requête. Les réponses sont classées suivant plusieurs critères dont la popularité du site mesurée par le nombre d'utilisateurs qui l'ont dans leurs bookmarks. D'autres mesures s'inspirant du PageRank de Google sont proposées.

[6] The Socialized.Net. `http://www.socialized.net`.
[7] del.icio.us : social bookmarks. `http://del.icio.us`.
[8] Folksonomy From Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Folksonomy`.
[9] StumbleUpon. `http://www.stumbleupon.com`
[10] Searchius. `http://searchius.ceid.upatras.gr/`

6.2.3 Desktop Search

Plusieurs systèmes commerciaux proposent un outil de recherche locale. Parmi ces outils, on peut citer Google Desktop[11], Copernic[12], Yahoo Desktop Search[13] et MSN Desktop Search[14]. Ces trois outils indexent, pendant que le processeur de la machine de l'utilisateur est inoccupé, les mails, le contenu du disque dur et les pages web visitées par l'utilisateur. L'interface homme-machine permet à l'utilisateur de rechercher parmi ses mails et ses fichiers ceux dont le nom ou le contenu textuel contiennent les mots de la requête.

6.2.4 Bilan

Nous avons décrit un ensemble d'applications de recherche collaborative, de bookmarks partagés et de *desktop search* (recherche de données locales). Ces applications ne sont apparues que récemment mais connaissent déjà un succès grandissant et intéressent de nombreux acteurs dont Microsoft, Google et Yahoo.

Cependant, à notre connaissance, il n'existe pour le moment aucun système de recherche textuelle permettant de rechercher de manière collaborative des documents publiés par ses utilisateurs tout en leur fournissant des réponses pertinentes et personnalisées. C'est ce que nous allons proposer avec notre système Maay. Nous nous sommes inspirés du bookmark collaboratif et du desktop search proposés par les différentes applications pour inclure ces fonctionnalités dans notre système.

6.3 Architecture du nœud Maay

L'architecture de Maay est composée de plusieurs modules qui sont représentés sur la figure 6.1 :

- le **crawler** parcourt les fichiers du disque dur en vue de leur indexation ;
- l'**indexeur** analyse le contenu textuel des documents et enregistre leur description (titre, taille, ...) et leurs associations document-mot dans la base de donnée ;
- la **base de données** sert à stocker les informations sur les documents (titre, URL, taille, profil, ...) et sur les nœuds (IP, port, profil, ...) ;
- le **gestionnaire de téléchargement** gère les téléchargements en cours lancés par l'utilisateur ;
- le **gestionnaire de profils** met à jour les profils des documents et des utilisateurs ;
- le **moteur de requêtes** répond aux requêtes de recherche provenant de l'utilisateur ou des autres nœuds Maay ;
- le **gestionnaire de communications** assure la communication entre le nœud et l'utilisateur et entre le nœud et les autres nœuds Maay.

Nous allons décrire dans cette section les différents modules et leurs interactions.

6.3.1 Terminologie

Nous définissons un *document* comme une donnée. Cette donnée peut être un texte, une image, une vidéo ou une ressource quelconque. Un document est identifié de manière unique par une clé de hachage sur son contenu.

Nous utiliserons le terme *visibilité* d'un document pour désigner si celui-ci est *public* ou *privé*.

[11] Google Desktop. `http://desktop.google.com/`.
[12] Copernic Desktop Search. `http://www.copernic.com/en/products/desktop-search/`.
[13] Yahoo Desktop Search. `http://desktop.yahoo.com`.
[14] Windows Desktop Search. `http://desktop.msn.com/`.

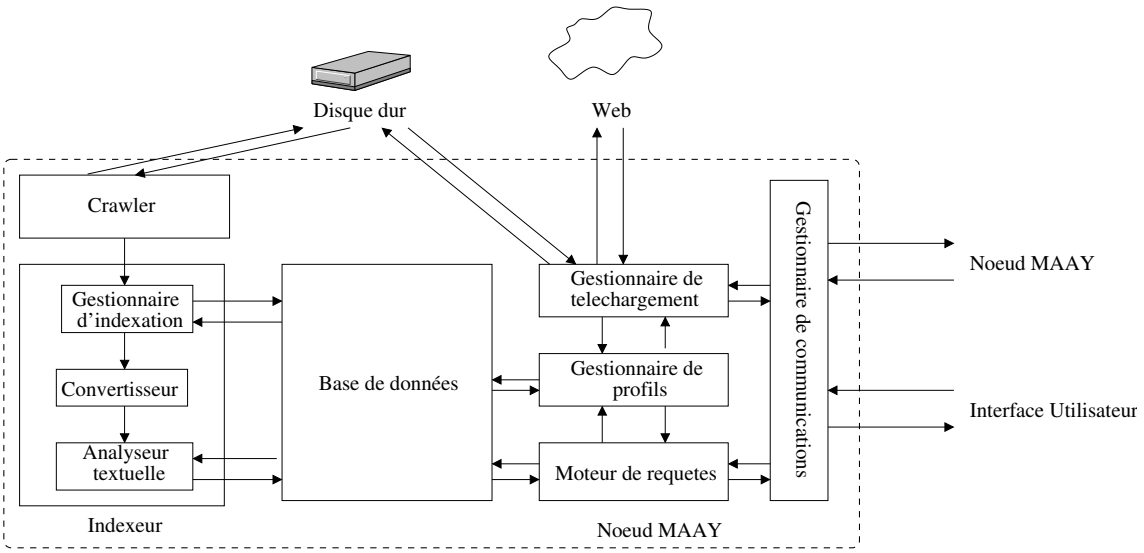


FIG. 6.1: Architecture d'un nœud MAAY

Un document a un *état* qui peut prendre trois valeurs :
- *intentionnel* : le document a été intentionnellement mis par l'utilisateur dans les répertoires indexés par l'indexeur.
- *mis en cache* : le document provient d'un téléchargement effectué par l'utilisateur en utilisant son nœud MAAY. Il est présent sur le disque dur mais peut être supprimé si la place vient à manquer.
- *connu* : le document est connu à partir des réponses de recherche reçues. Son profil est partiel et seuls quelques mots du document sont connus.

Un *fichier* est l'enregistrement physique d'un document sur le disque dur. Un même document peut être enregistré dans plusieurs fichiers se trouvant à des endroits différents du disque dur. Il peut aussi être présent sur le disque dur de plusieurs utilisateurs. Nous appellerons *fournisseur* de document un nœud qui possède un enregistrement du document sur son disque dur.

6.3.2 Crawler

Le crawler parcourt une liste de ressources à indexer et envoie à l'indexeur la liste des fichiers à analyser.

Règles d'indexation Les règles d'indexation permettent à l'utilisateur de spécifier les ressources qu'il veut indexer et rendre publiques ou privées. L'utilisateur définit quatre ensembles :
- les répertoires à indexer (`IDX`)
- les répertoires à ne pas indexer (`NIDX`)
- les répertoires à publier (`PUB`)
- les répertoires à ne pas publier (`NPUB`)

En paramétrant ces ensembles, un utilisateur peut spécifier un répertoire qu'il veut indexer en excluant quelques sous-répertoires, ou rendre privés un répertoire et ses sous-répertoires à l'exception d'un sous-répertoire qui est rendu public.

Les deux premiers ensembles `IDX` et `NIDX` permettent de définir les fichiers qui seront indexés. Un fichier f est indexé si et seulement si :

$$f \in (IDX \setminus NID)$$

Les deux derniers ensembles `PUB` et `NPUB` déterminent, parmi les fichiers indexés, ceux qui seront privés ou publics.

Les fichiers privés sont ceux qui ne sont pas dans les répertoires publiés `PUB` ou dans les répertoires à ne pas publier `NPUB`. Autrement dit, un fichier f est indexé et privé si et seulement si :

$$f \in (IDX \setminus NID) \cap (NPUB \setminus PUB)$$

Les fichiers publics sont ceux qui sont dans les répertoires publiés `PUB` et qui ne sont pas dans les répertoires à ne pas publier (`NPUB`). En d'autres mots, un fichier f est indexé et public si et seulement si on a :

$$f \in (IDX \setminus NID) \cap (PUB \setminus NPUB)$$

Parcours de la liste des fichiers On utilise deux crawlers qui parcourent chacun l'ensemble des fichiers publics et l'ensemble des fichiers privés. Le crawler parcourt sa zone à indexer en utilisant un parcours en profondeur dans l'ordre alphabétique et alimente le gestionnaire d'indexation avec les fichiers parcourus.

Une fois que le crawler a fini de parcourir sa zone, il attend une période donnée avant de la parcourir à nouveau. Cela permet de détecter si des fichiers ont été ajoutés, supprimés ou modifiés et de mettre à jour les index.

6.3.3 Indexeur

Gestionnaire d'indexation Le gestionnaire d'indexation reçoit la liste des fichiers à indexer des deux crawlers mais aussi du gestionnaire de téléchargements qui le notifie des nouveaux fichiers rapatriés sur le disque dur. Ces nouveaux fichiers peuvent provenir soit des fichiers téléchargés chez les autres utilisateurs soit des pages web visitées par l'utilisateur.

Pour détecter les ajouts, suppressions et modifications qui ont eu lieu dans les espaces parcourus par les crawlers, l'indexeur n'a pas d'autre choix que de parcourir périodiquement ces espaces et de détecter les changements. A l'exception du premier parcours, la majorité des fichiers envoyés par le crawler à l'indexeur sont déjà indexés et seule une petite partie a besoin de l'être.

C'est pourquoi le gestionnaire d'indexation attribue une priorité différente aux différents fichiers qui lui sont fournis en entrée. Les fichiers fournis par le gestionnaire de téléchargement sont traités en priorité par rapport aux fichiers fournis par les deux crawlers car ils correspondent à des documents nouveaux. Les fichiers envoyés par le crawler en charge de l'espace public sont traités avec une priorité plus élevé que ceux envoyés par le crawler en charge de l'espace privé pour que le nœud puisse fournir des informations à jour sur ses documents publics aux autres nœuds.

Ces deux ensembles sont traités séparément par le gestionnaire d'indexation.

Algorithme de détection d'ajout, suppression et modification de fichiers L'algorithme présenté (voir Algorithme 3) permet de détecter les fichiers qui ont été ajoutés, supprimés ou modifié lors du dernier parcours.

L'algorithme consiste à parcourir conjointement deux listes triées par ordre alphabétique, d'une part la liste des fichiers parcourus par le crawler et d'autre part la liste des fichiers

Algorithme 3 : Algorithme de détection d'ajout, modification et suppression de fichiers

```
crawled_files ← liste de fichiers parcourus par le crawler classée par ordre alphabétique
Si le crawler est responsable de l'espace public alors
    db_files ← liste des fichiers indexés publics enregistrés dans la BD et classée par ordre alphabétique
sinon
    db_files ← liste des fichiers indexés privés enregistrés dans la BD et classée par ordre alphabétique
i ← 0
j ← 0
Tant que i < len(crawled_files) ou j < len(db_files)
    cf ← crawled_files[i]
    df ← db_files[j]
    Si cf.name < df.name ou j ≥ len(db_files) alors
        # le fichier cf a été créé
        index_file(cf)
        i = i + 1
    sinon si cf.name > df.name ou i ≥ len(crawled_files) alors
        # le fichier df a été supprimé
        unindex_file(df)
        j = j + 1
    sinon
        Si cf.time ≠ df.time alors # le fichier a peut être été modifié
            Si cf.sha1 ≠ df.sha1 alors
                reindex_file(cf)
        i = i + 1
        j = j + 1
```

indexés lors du dernier crawl et référencés dans la base de données. Soit cf le curseur pointant sur la liste des fichiers renvoyés par le crawler. Soit df le curseur pointant sur la liste des fichiers référencés lors du dernier crawl dans la base de donnée. Ces deux curseurs cf et df sont initialisés au début de leur liste respective.

Si le nom du fichier pointé par cf est plus petit que celui pointé par df, cela signifie que le fichier pointé par cf est un nouveau fichier qui doit être indexé. On avance le curseur cf.

Par contre, si le curseur cf pointe sur un fichier dont le nom est alphabétiquement plus grand que celui pointé par df, cela signifie que le fichier indexé et pointé par df a été supprimé du disque dur depuis le dernier crawl. On enlève les informations relatives à ce fichier dans la base de données et on avance le curseur df.

Si le fichier pointé par cf est égal à celui pointé par df, cela signifie que le fichier a déjà été indexé. Pour vérifier que le contenu du fichier n'a pas été modifié, on commence par comparer la date de dernière modification du fichier avec celui indexé dans la base de donnée. Si elle est différente, on calcule la clé SHA-1 du fichier et on le compare avec celle de la base de donnée. Si la clé SHA-1 est différente, le fichier doit être réindexé. On avance les curseurs cf et df.

Pour déterminer la visibilité d'un document, il suffit de regarder parmi ses différentes instances de fichier si au moins l'une d'entre elles est publique. Si c'est le cas, le document est public et est partagé avec les autres, sinon il est privé. De même, pour déterminer son état, il suffit de regarder parmi ses différentes instances de fichier. Si une d'entre elles a un état *intentionnel*, alors le document a un état *intentionnel*, sinon le document a l'état *mis en cache*. Si le document n'a pas d'instances de fichier, cela signifie qu'il est connu grâce à une réponse de recherche reçue antérieurement, son état est donc *connu*.

Convertisseur Pour extraire le contenu textuel du fichier, l'indexeur commence par déterminer le type du fichier en se basant sur l'extension du nom du fichier ou sur le type MIME[15] retourné par le serveur web dans le cas d'une ressource du Web.

Notre analyseur textuel gère le format texte et le format HTML. Nous avons rencontré plusieurs problèmes avec les parseurs HTML existants. Ceux-ci étaient peu robustes aux erreurs syntaxiques et ne permettaient pas d'extraire dans plusieurs cas le texte visible de la page. Nous avons donc programmé un parseur HTML plus tolérant aux erreurs de syntaxes.

Pour indexer les fichiers postscripts, PDF, Word et RTF, nous avons utilisés des programmes externes permettant leur conversion en HTML ou en texte.

Les convertisseurs utilisés sont décrits dans le tableau suivant :

Format source	Format de destination	Programme
PDF	HTML	xpdf[16] (pdftotext)
Word	HTML	antiword[17]
RTF	HTML	rtf2html[18]
postscript	texte	pstotext[19]

Ces convertisseurs sont sous licence GPL[20] et peuvent donc être utilisés librement.

Analyseur textuel Une fois le titre et le contenu textuel extraits du document, l'analyseur textuel extrait les mots du document. Comme la majorité des moteurs de recherche, notre analyseur textuel ne tient pas compte de la casse ni des signes diacritiques. Un signe diacritique (accent, cédille, ...) est un signe placé a coté d'une lettre pour en modifier la valeur phonétique, permettre une lecture plus précise ou pour éviter une ambiguïté entre des homographes (mots qui s'écrivent de la même manière). Ainsi, notre analyseur considérera les mots suivants comme équivalents : Mais, maïs, mais.

Ces associations ainsi que les informations relatives au document : identifiant, titre, taille, date, contenu textuel, URL dans le cas d'une ressource web sont enregistrées dans la base de données.

6.3.4 Base de données

Toutes les informations sur les voisins, documents, mots et les relations entre eux sont stockés dans une base de données relationnelle et sont répartis dans plusieurs tables. Nous avons utilisé la base de donnée MySQL[21] sous licence GPL qui est disponible sur la plupart des systèmes d'exploitations.

Les tables sont représentées sur la figure 6.2 par des boîtes dont les noms sont écrits en blanc sur noir. La liste de ces champs est décrit en-dessous. Les champs en gras précisent les clés par lesquelles sont accédées les entrées des table. Par exemple les entrées de la table `Document` seront principalement accédées par son champ `Document ID`.

[15] Le type MIME est un standard pour identifier des contenus d'Internet. Les différents format d'image, vidéo et texte ont chacun leur type MIME associé. Par exemple le type MIME d'une page HTML est `text/html`.

[16] Xpdf : A PDF Viewer for X. `http://www.foolabs.com/xpdf/`.

[17] Antiword : a free MS Word document reader. `http://www.winfield.demon.nl/`.

[18] RTF to HTML Converter. `http://www.w3.org/Tools/HTMLGeneration/rtf2html.html`.

[19] The pstotext program. `http://www.research.compaq.com/SRC/virtualpaper/pstotext.html`.

[20] GNU General Public Licence. `http://www.gnu.org/copyleft/gpl.html`

[21] MySQL. `http://www.mysql.com/`

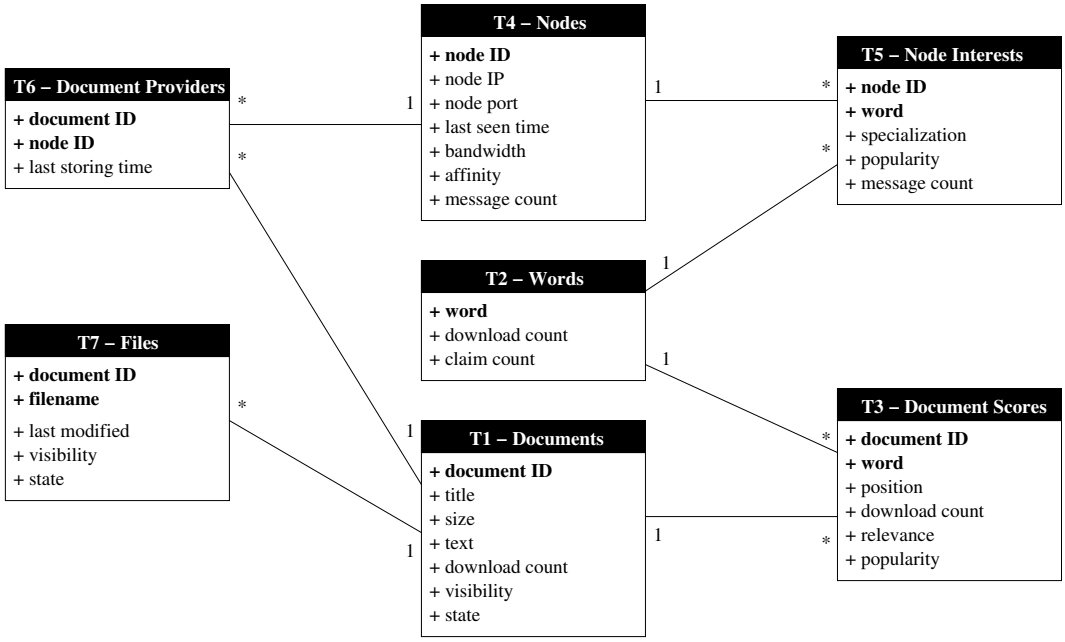


FIG. 6.2: Structure de données

Les tables sont liées par des relations matérialisées par un trait sur la figure. Les deux valeurs aux deux extrémités du trait représentent la cardinalité de la relation. Par exemple la table des `Documents` et la table des `Document Scores` sont liées par un trait ayant à ses extrémités `1` et `*`. Cela signifie qu'un document peut avoir plusieurs scores document-mot (pour chacun de ses mots), mais que le score document-mot n'est rattaché qu'à un seul document.

T1 - Table des documents La table des documents décrit les informations relatives au document : son identifiant généré par la fonction de hachage SHA-1 sur son contenu, son titre, la taille du document en octet. Il contient aussi le contenu textuel du document qui est utilisé pour afficher un extrait (contexte) du document autour des mots de la requête. Le champ *visibility* décrit si le document est public ou privé. Le champ *state* décrit les trois états possibles du documents : *intentionnel*, *mis en cache* ou *connu*. Enfin, *download count* décrit le nombre de fois où le document a été téléchargé.

T2 - Table des mots La table des mots associe à chaque mot le nombre de manifestations d'intérêt pour ce mot et le nombre de fois où il a été utilisé dans une requête de téléchargement.

T3 - Table des scores des documents La table des scores des documents associe aux documents, les mots qui le décrivent. Ces mots sont tous ceux présents dans le contenu textuel du document mais aussi ceux utilisés par les utilisateurs pour annoter le document. Chaque couple mot-document est associé à plusieurs valeurs : le nombre de fois où le document a été téléchargé pour le mot, et la valeur de pertinence et de popularité du mot pour le document.

T4 - Table des nœuds La table des nœuds décrit les informations relatives à un nœud : son adresse IP, son port, la date de dernière communication avec ce nœud, le nombre de fois

où il a manifesté son intérêt, sa valeur d'affinité avec le nœud de l'utilisateur et sa bande passante.

T5 - Table des intérêts des nœuds Cette table est composée d'un ensemble d'associations nœud-mot qui décrivent les intérêts des nœuds : les valeurs de spécialisation et d'expertise du nœud pour le mot, et le nombre de fois où le nœud a clamé son intérêt pour le mot.

T6 - Table des fournisseurs de documents Cette table décrit sur quels nœuds un fichier peut être téléchargé. A chaque association nœud-document est associée la date la plus récente à laquelle le nœud a affirmé avoir le fichier (dans une réponse de recherche).

T7 - Table des fichiers Cette table associe à chaque fichier indexé présent sur le disque le document auquel il est associé. Si deux fichiers ont un contenu identique, ils sont associés au même document. Chaque entrée de la table est composée de l'identifiant du document, du nom du fichier, de sa `visibilité` (qui peut être *public* ou *privé*) qui dépend du répertoire dans lequel il se trouve, de son *état* (qui peut être *intentionnel* ou *mis en cache*) et de la date de dernière modification du fichier.

6.3.5 Moteur de requêtes

Le moteur de requêtes accepte une requête de recherche en entrée et retourne une liste de documents triés suivant plusieurs critères.

Le moteur de requêtes répond à trois types de requête de recherche :

- **Recherche locale** qui permet à l'utilisateur d'effectuer des recherches sur les données présentes sur son disque dur, les fichiers qu'il a téléchargée et les pages web qu'il a visitées.
- **Recherche collaborative** qui englobe tous les documents partagés par les utilisateurs. La recherche s'effectue en interrogeant plusieurs voisins. Les résultats reçus sont agrégés puis classés avant d'être retournés à l'initiateur de la requête.
- **Recherche web** L'idée est de proposer à l'utilisateur une recherche classique sur le web ou sur l'intranet en enrichissant les résultats retournés par le moteur de recherche par diverses informations comme les mots pertinents de la page, les différentes versions disponibles de la page web sur le réseau MAAY. Des actions sont aussi associées aux résultats et permettent à l'utilisateur d'indexer et d'annoter des page web ou encore d'en rechercher d'autres versions sur le réseau MAAY.

Le moteur de requêtes garde en mémoire des informations sur les recherches qu'il a émises (ou retransmises) dans une table de recherches.

Chaque recherche est associée à :

- un *query ID* : cet identifiant est forgé par l'initiateur de la requête et identifie les messages associés à la recherche
- un *mots de la requête*
- un *identifiant du nœud émetteur de la requête*
- une *liste de réponses*
- *liste des documents déjà renvoyés* : elle est utilisée pour ne pas retransmettre plusieurs fois la description d'un même document à l'émetteur de la requête (dans le cas où le nœud n'est pas l'initiateur de la requête).

En recevant une requête de recherche ayant un identifiant *qid*, le nœud vérifie dans sa table qu'il n'a pas déjà lancé une recherche d'identifiant *qid*. Si c'est le cas, elle est rajoutée à

la table de recherches. Sinon la requête de recherche n'est pas prise en compte, ce qui permet d'éviter les cycles.

Prédicats de recherche La recherche peut s'effectuer sur trois critères possibles :
- présence de mots dans le document ;
- URL du document (dans le cas d'une page récupérée du web). Cela permet de pouvoir rechercher différentes versions d'une page web ;
- identifiant du document. Typiquement pour rechercher des fournisseurs d'un document pour le télécharger.

Fonctionnement de la recherche locale Elle s'effectue en cherchant parmi les documents publics et privés référencés dans la base de données, celles ayant un *état intentionnel* ou *mis en cache*.

Fonctionnement de la recherche collaborative Elle est initiée en envoyant une requête de recherche aux voisins pour leur demander s'ils ont ou connaissent des documents convenant à la requête. Les voisins retransmettent la requête à leurs voisins et ainsi de suite. Les voisins ayant des réponses sur des documents publics les retransmettent au nœud qui lui a retransmis la requête. Les réponses remontent ainsi de proche en proche jusqu'à l'initiateur de la requête. Les mécanismes de propagation des requêtes et du classement des résultats sont décrits plus en détail dans le chapitre 3.

Fonctionnement de la recherche web La recherche s'effectue en interrogeant un moteur de recherche externe. Dans notre prototype, nous avons proposé deux recherches : un utilisant Google pour rechercher sur le Web et l'autre utilisant un moteur de recherche du web de l'intranet. Pour interroger le moteur de recherche, notre programme lui envoie une requête HTTP contenant la requête de recherche en l'adaptant au moteur. Celui-ci lui renvoie ses résultats dans une page HTML qui est analysée par notre parseur HTML. Les titres des documents, leurs extraits, leur URL sont extraites de la page HTML et rajoutée à la liste de résultats associés à cette recherche.

6.3.6 Gestionnaire de téléchargement

Le gestionnaire de téléchargement maintient une table de téléchargements pour manager les téléchargements en cours. A l'instar des programmes d'échange de fichiers, le nombre de téléchargements simultanés peut être limité par l'utilisateur.

La table de téléchargements associe à chaque téléchargement :
- le document ID du document à télécharger
- une liste de fournisseurs potentiels du document

Quand le nœud reçoit de la part de l'utilisateur un téléchargement pour le document d, il vérifie dans sa table de téléchargements que le document d n'est pas déjà en cours de téléchargement. Si ce n'est pas le cas, il est rajouté dans la table de téléchargements.

Le gestionnaire maintient pour chaque téléchargement une liste de fournisseurs potentiels du document. Cette liste est initialisée par la liste des fournisseurs du document qui sont connus grâce aux réponses de recherche reçues (ces informations sont stockées dans la base de donnée dans la table `Document Providers`).

Pour chacun de ces fournisseurs, le gestionnaire maintient la date à laquelle il l'a interrogé sur le document pour la dernière fois et son état qui peut être :
- *occupé* : le fournisseur a renvoyé une réponse décrivant qu'il ne pouvait pas à cet instant transmettre le fichier. Dans ce cas, le nœud va être réinterrogé ultérieurement,

- *non joignable* : le nœud ne sera plus interrogé,
- *n'a pas le document* : le document a été supprimé ou modifié par ce fournisseur. Le nœud ne sera pas interrogé ultérieurement,
- *ok* : le fournisseur a le fichier,
- *non contacté* : le fournisseur n'a pas encore été contacté.

Les fournisseurs sont successivement interrogés jusqu'à ce que l'un d'eux accepte d'envoyer le document. Si aucun d'entre eux n'accepte d'envoyer le document, les fournisseurs ayant l'état *occupé* et n'ayant pas été interrogés depuis un certain temps sont réinterrogés. Parallèlement et périodiquement, une recherche par identifiant de document est lancée pour trouver de nouveaux fournisseurs.

6.3.7 Gestionnaire de profils

Le gestionnaire de profils est responsable de mettre à jour le profil des documents et des voisins.

Profil des documents Les profils des documents sont mis à jour après chaque téléchargement. Ces téléchargements peuvent être distants ou locaux. Même la consultation d'un document localement par l'utilisateur est considérée comme un téléchargement.

Les profils de document sont aussi mis à jour lors de l'annotation d'un document. Le profil du document annoté est modifié de la même façon que dans le cas d'un téléchargement associé aux mots utilisés pour l'annotation.

Profil des voisins Les profils des voisins sont mis à jour après la réception de messages. Le profil d'un voisin est modifié lors de la réception d'un message si le voisin l'a émis ou s'il apparaît dans les fournisseurs d'un document dans un message de réponse.

6.3.8 Gestionnaire de communications

Le gestionnaire de communications gère l'envoi et la réception de messages de manière asynchrone. Il est responsable de l'encodage et du décodage des messages.

Le gestionnaire reçoit les messages provenant soit d'autres nœuds MAAY soit du navigateur web de l'utilisateur.

Protocole entre nœuds Maay La communication entre les nœuds s'effectuent par des messages encapsulés dans des requêtes HTTP.

Nous avons choisi d'encoder les messages de la manière la plus compacte et économique possible. Le message est codé sans sa structure ni le type de ses données. Les entiers, flottants sont codés sur 4 octets et les chaînes de caractères sont codés par un octet indiquant sa taille suivi du contenu de la chaîne.

Nous aurions pu utiliser *XML-RPC*[22] ou *SOAP*[23] qui sont deux protocoles de communications utilisant XML pour l'encodage des messages et reposant souvent sur HTTP. Mais ils sont coûteux en bande passante et demandent une charge de calcul non négligeable pour encoder et décoder les messages en XML. Les premiers tests avec ce protocole ont vite montré qu'ils n'étaient pas adaptés à notre système.

Les messages sont envoyés sous la forme :

[22] XML-RPC Home Page.`http://www.xmlrpc.com/`.
[23] W3C. *SOAP Specification.* `http://www.w3.org/TR/soap/`.

```
POST /maay/messages [en-tête HTTP]

[contenu du message MAAY]
```

La composition des messages est détaillée dans la figure 3.1 de la section 3.2

Nous prévoyons d'utiliser le codage *bencode* qui permettra de faire évoluer plus facilement le protocole. Le codage *bencode* permet de coder et de décoder de façon compacte des données de type entier, chaîne de caractère, séquence et dictionnaire. La structure du message et les types des données utilisées sont codés dans le message ce qui permet au décodeur de ne pas avoir à connaître, à priori, la structure du message.

Protocole entre nœud Maay et le navigateur de l'utilisateur Le navigateur dialogue avec le nœud MAAY en utilisant le protocole HTTP. Le navigateur envoie des requêtes HTTP au nœud MAAY qui lui répond en lui renvoyant une page HTML.

Pour générer les pages HTML retournées par le nœud MAAY au navigateur, nous avons utilisé le moteur de template Cheetah[24]. Son système de template permet de pouvoir créer des pages HTML dynamiquement sans inclure de code HTML dans notre programme.

6.4 Interface homme machine

L'interface homme-machine (IHM) permet à un utilisateur de communiquer avec son nœud MAAY pour effectuer des recherches, visualiser les résultats et télécharger des documents.

Nous avons choisi une IHM accessible par un navigateur web pour deux raisons. La première est de s'affranchir des problèmes de compatibilité liés aux interfaces graphiques sur les différents systèmes d'exploitation. Et la seconde, de fournir à l'utilisateur la possibilité d'accéder à son nœud MAAY à partir de n'importe quel ordinateur distant sans y avoir à installer de logiciels.

Interface de recherche L'interface de recherche se présente comme celle d'un moteur de recherche classique. L'utilisateur tape ses mots clés dans une boîte de saisie puis envoie sa recherche en appuyant sur un des 4 boutons prévus à cet effet. L'utilisateur a le choix d'effectuer sa recherche :

- dans son espace local uniquement
- sur les fichiers partagés sur le réseau MAAY
- sur le web
- sur le web de l'intranet

L'utilisateur peut effectuer des recherches plus avancées (figure 6.4). L'interface qui lui est proposée lui permet de rechercher des documents par mots clés et/ou par URL ou par identifiant de document.

Cependant, le temps de réception des réponses est plus élevé que dans les moteurs de recherche centralisés et d'autre part, des nouveaux résultats sont continuellement reçus après l'envoi de la requête, changeant l'ordre de présentation des résultats.

Nous nous sommes donc inspirés des IHM des systèmes pair à pair d'échange de fichiers et avons proposés une interface permettant d'effectuer plusieurs recherches en parallèle. Une barre d'onglet affiche les recherches en cours et permet à l'utilisateur de sélectionner les résultats des recherches qu'il souhaite observer.

Les résultats se présentent sous la forme d'une liste de descriptions de documents.

[24] Cheetah - The Python-Powered Template Engine. `http://www.cheetahtemplate.org/`

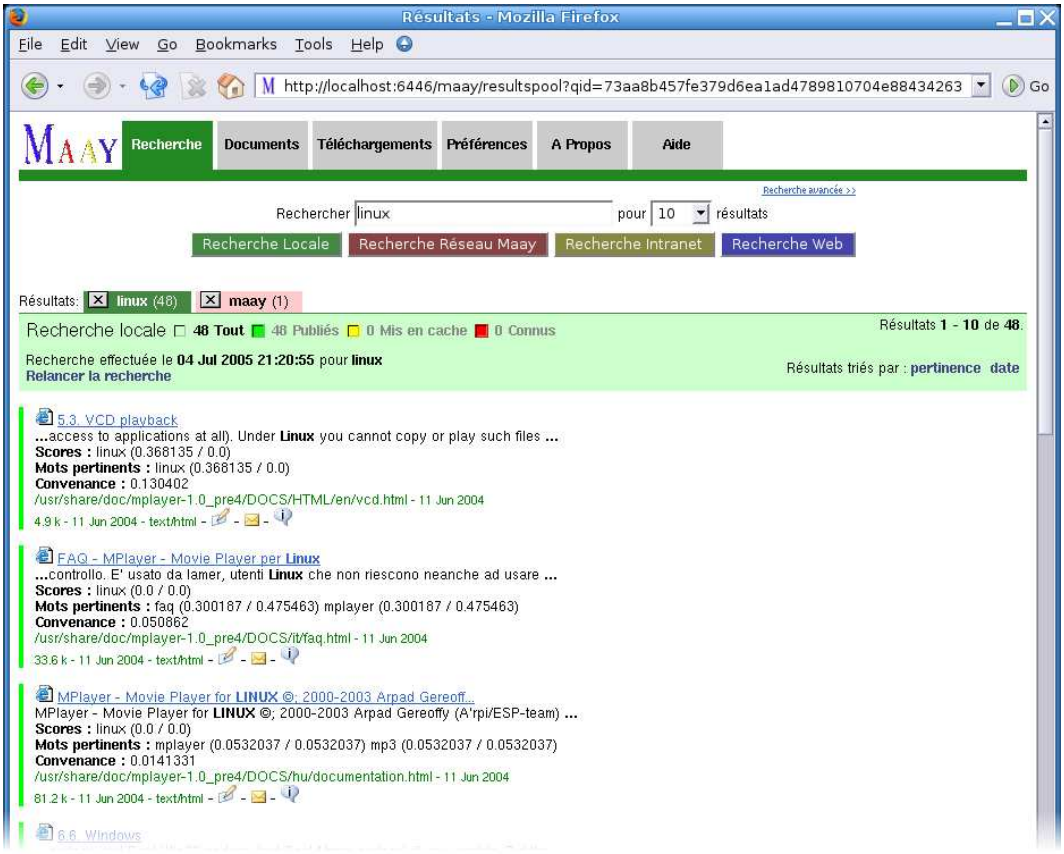


FIG. 6.3: Interface de recherche

Un document est décrit par son titre, un extrait textuel du document autour des mots de la requête, les scores de chaque mot de la requête, les mots les plus pertinents dans le document. Des actions sont associées à chaque résultat et permettent de le télécharger, l'annoter, visualiser des informations supplémentaires sur le document et effectuer des recherches sur d'autres versions du document.

FIG. 6.4: Recherche avancée

Annotation / Tagging L'annotation consiste à associer de nouveaux mots à un document ou à renforcer l'association entre des mots et le document.

Pour annoter un document, l'utilisateur peut soit appuyer sur le bouton `Annoter cette page` présent dans la barre d'adresse (figure 6.6), soit appuyer sur une icône affichée à coté des résultats de recherche. Ces deux actions entraînent l'ouverture d'une fenêtre d'annotation (figure 6.5) qui permettent à l'utilisateur de taper les mots clés qu'il veut associer au document. L'utilisateur peut taper des mots quelconques, même ceux qui n'apparaissent pas dans le contenu textuel du document.

FIG. 6.5: Fenêtre d'annotation de document

Indexer une page web L'utilisateur peut choisir d'indexer une page web de la même manière qu'il le met dans ses marque pages. En appuyant sur le bouton `Indexer cette page` situé dans la barre d'adresse du navigateur (figure 6.6), le navigateur lui ouvre une fenêtre qui lui permet de confirmer l'indexation de la page web et de l'annoter.

Une fois la page indexée, celle-ci est rendue publique et peut être cherchée de la même manière que les autres documents. Elle peut de plus être cherchée par URL afin d'offrir aux utilisateurs la possibilité de rechercher différentes versions d'une page web.

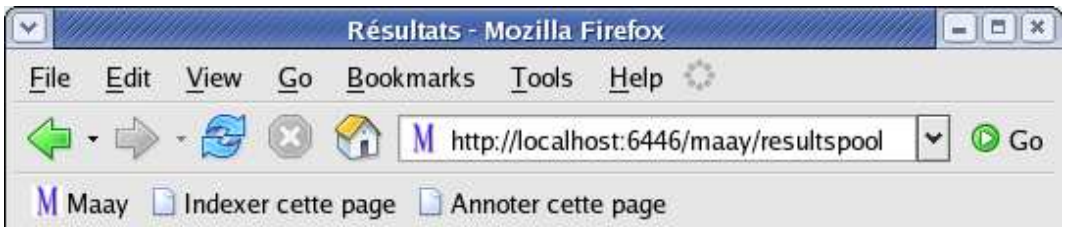


FIG. 6.6: Bouton dans la barre d'adresse du navigateur permettant d'indexer la page web courante

Version d'une page web Les pages provenant du web et apparaissant dans les résultats de recherche sont associées à une action qui permet à l'utilisateur de voir les différentes versions d'une page web connue par le nœud (figure 6.7). L'utilisateur peut choisir de visualiser une version de la page à la date indiquée. Il peut aussi lancer une recherche pour trouver de nouvelles versions de la page présentes dans le réseau MAAY.

Interface de téléchargement L'interface de téléchargement permet à l'utilisateur de surveiller les téléchargements en cours. Il peut regarder le niveau d'avancement des téléchargements et en arrêter.

FIG. 6.7: Versions disponibles d'une page web

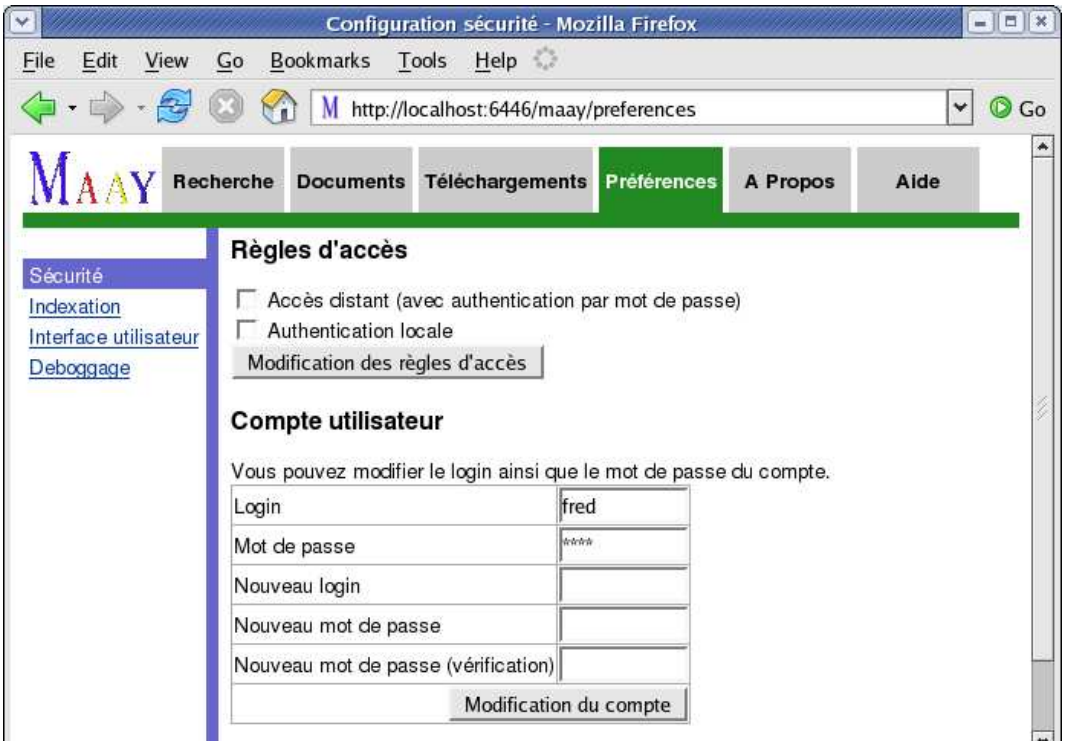


FIG. 6.8: Configuration des règles d'accès

Interface de configuration L'interface de configuration permet à l'utilisateur de configurer trois paramètres.

Le premier est l'accès à l'interface Web de MAAY (figure 6.9). Les règles d'accès permettent à l'utilisateur d'autoriser ou non l'accès à l'interface web de manière distante. Les avantages de l'accès distant est de pouvoir, à partir de n'importe quel ordinateur relié à l'Internet, accéder à l'interface web pour rechercher ses fichiers et les télécharger. L'utilisateur peut choisir de protéger l'accès à l'interface web par mot de passe, même lorsqu'il y accède localement et de le modifier.

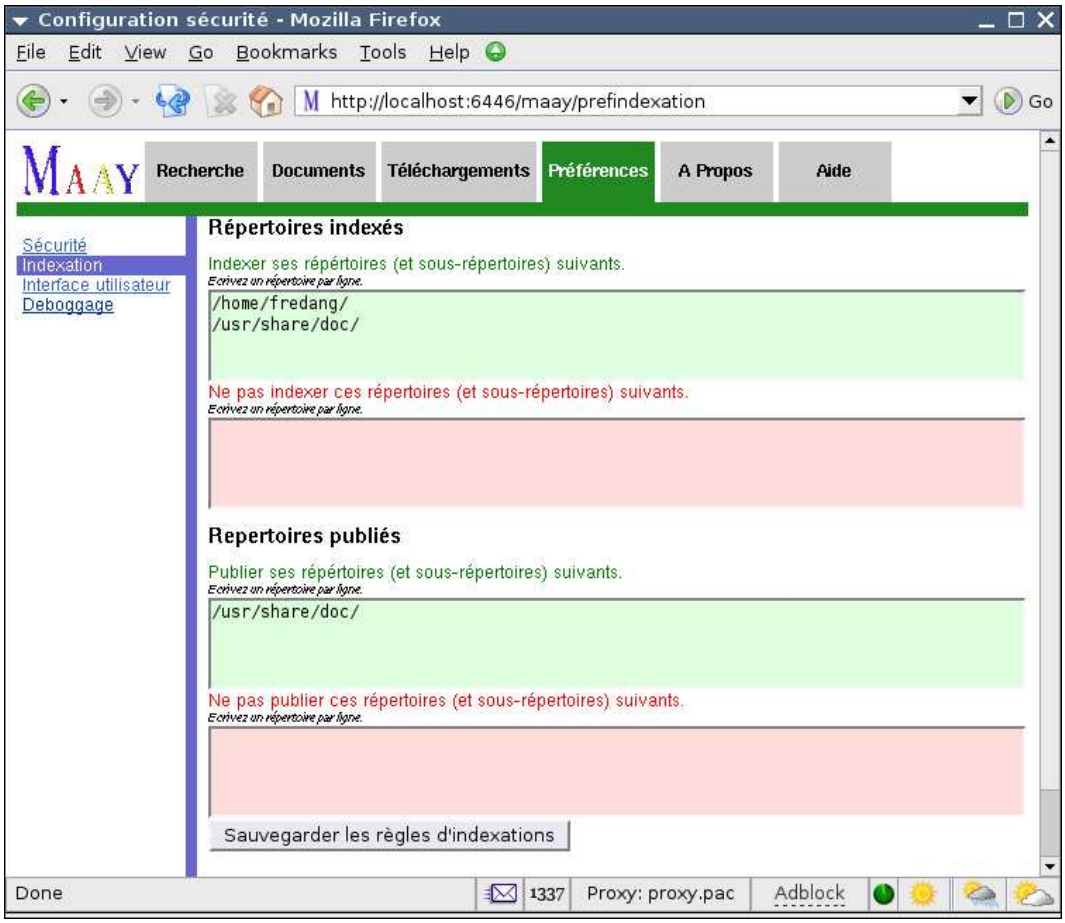


FIG. 6.9: Configuration de l'indexation des répertoires

Le deuxième concerne les règles d'indexation. L'utilisateur peut spécifier les répertoires de son disque dur qu'il souhaite indexer et ceux qu'il souhaite rendre public (figure 6.9). Les règles d'indexations ont été décrites dans la sous-section 6.3.2.

Le troisième est l'interface utilisateur. L'utilisateur peut paramétrer le nombre de résultats qu'il souhaite afficher par pages de résultats.

6.5 Méthodes pour faciliter son déploiement

6.5.1 Installeur

Pour faciliter le déploiement du prototype, nous avons d'abord utilisé le module py2exe[25] qui permet de générer sous Windows des fichiers exécutables à partir du code écrit en Python. Un utilisateur souhaitant installer MAAY sur sa machine n'a qu'à copier ces fichiers exécutables sans avoir à installer Python sur sa machine.

Puis nous avons utilisé NSIS qui est un outil pour générer des auto-installeurs sous Windows. Un auto-installeur est un programme qui, lorsqu'il est exécuté, installe automatiquement un autre programme sans assistance de l'utilisateur.

Le déploiement sous Linux pourra s'effectuer en adaptant l'installation à ses différentes distributions.

[25] py2exe. `http://starship.python.net/crew/theller/py2exe/`

6.5.2 Bootstrap

Connaître des nœuds avec lesquels se connecter pour se joindre au système (ou boostrap) est un problème récurrent dans les systèmes pair à pair. Une solution naïve consisterait à utiliser un serveur qui maintiendrait une liste d'adresses de quelques nœuds connectés sur le réseau et qui la fournirait aux nouveaux nœuds entrants. Cette liste serait mise à jour par les nœuds qui enverraient leur adresse IP au serveur lors de leur entrée dans le système ou de manière périodique. Cependant, cette solution ne passe pas l'échelle avec des milliers d'utilisateurs qui engendreraient un trafic considérable au niveau du serveur.

GWebCache

GWebCache[26] est un script installé sur un serveur web permettant à un client Gnutella de récupérer une liste de nœuds Gnutella auxquels se connecter. Les GWebCaches sont accessibles *via* une URL et communiquent en HTTP.

Le protocole GWebcache spécifie deux types de requêtes : les requêtes GET pour récupérer une liste d'adresse IP de nœuds Gnutella et d'URLs pointant sur d'autres serveurs GWebCache. Et les requêtes UPDATE permettant à un nœud d'informer le GWebCache qu'un ultrapeer ou un serveur GWebCache est présent dans le réseau.

Le serveur maintient une liste limitée d'adresses IP des ultrapeers et d'URLs de GWebCache et ne garde que les dernières reçues. Cela permet de garantir que les adresses des ultrapeers et des GWebCaches retournées à un client soient toujours fraîches et donc ont de grande chance d'être encore accessibles.

Nous nous sommes inspirés du GWebCache (voir encadré) de Gnutella pour implémenter notre MaayWebCache.

Chaque nœud Maay peut agir comme un MaayWebCache en acceptant deux types de requêtes. Le premier type de requête UPDATE permet à un nœud de lui transmettre ses coordonnées (identifiant, IP, port). Le second type de requête GET permet à un nœud de recupérer une liste de voisins connus par le MaayWebCache. Cette liste comprend les derniers voisins avec lesquels le nœud Maay a communiqué.

6.6 Conclusion

Nous avons décrit le prototype de Maay que nous avons implémenté en décrivant ses fonctionnalités et son architecture. Nous avons ensuite présenté l'interface homme machine permettant à l'utilisateur de communiquer avec son nœud Maay pour lancer des recherches et des téléchargements. Enfin nous avons décrit son implémentation et les différentes méthodes qui permettront une installation aisée du système et son déploiement.

Les expériences limitées que nous avons effectuées sur 3 machines, fonctionnant sous Windows et Linux, partageant chacune des milliers de documents HTML, PDF et textes montrent des résultats prometteurs.

Chapitre 7

Conclusion

Dans cette thèse nous nous sommes intéressés aux problèmes rencontrés par les moteurs de recherche. Les moteurs de recherche connaissent actuellement des difficultés à parcourir et à indexer les milliards de pages du web de manière exhaustive et régulière ; ainsi les résultats qu'ils retournent se révèlent incomplets et non à jour. De plus comme ils ne cherchent pas à adapter leurs résultats au profil de chacun de leurs utilisateurs, leurs résultats sont impersonnels et ont donc une pertinence moyenne. Avec un web sans cesse en croissance et en mutation, ces problèmes vont encore s'amplifier.

Nous avons proposé un système de recherche adaptatif visant à résoudre ces problèmes en décentralisant l'indexation et la recherche sur les machines des utilisateurs. Chaque utilisateur contrôle une machine agissant simultanément comme un fournisseur de contenu (*e.g.* serveur web) et un moteur de recherche sur son contenu. Son contenu local (faible comparé à la taille du web) peut être parcouru et indexé rapidement ce qui permet au moteur de recherche de fournir des résultats à jour. Les algorithmes que nous avons définis permettent à l'ensemble de ces machines de former un système de recherche global. Les mécanismes de routage et de classement que nous avons mis en place permettent aux utilisateurs de recevoir des réponses qui leur sont personnalisées.

Constatant qu'il n'existait pas de modèles des utilisateurs permettant de simuler des utilisateurs de moteur de recherche textuel, nous avons proposé un modèle des utilisateurs en s'appuyant sur diverses études sur des données réelles.

Nous avons ensuite implémenté un programme permettant de simuler des milliers de nœuds et nous avons simulé leurs utilisateurs à partir du modèle que nous avons défini. Les résultats que nous obtenons sont encourageants et prometteurs.

Enfin, nous avons développé un prototype implémentant plus de fonctionnalités dont la recherche locale et l'indexation des pages web visitées par l'utilisateur. Ce prototype nous a permis de spécifier l'architecture du système et de définir les fonctionnalités intéressantes pour les utilisateurs. Ce prototype nous servira de base pour la conception du système qui sera déployé prochainement.

7.1 Contributions

Nous allons résumer dans cette section les différentes contributions que nous avions présentées dans cette thèse.

Spécification d'un système de recherche sémantique par réseau adaptatif de pairs
Nous avons proposé un système de recherche de documents qui repose sur un réseau de pairs. Les machines des utilisateurs jouent simultanément le rôle de fournisseur de contenu (*e.g.*

serveur web) de moteur de recherche et des clients des moteurs de recherche. Nous avons définis des techniques permettant aux nœuds d'apprendre le profil des autres nœuds et des documents. Nous avons utilisé ces profils pour définir les règles de sélection des voisins à interroger et les règles de classement des résultats. En interrogeant des nœuds ayant un profil similaire au sien, un nœud a une probabilité plus élevée de trouver des documents pertinents pour lui. De plus en classant ses documents en fonction de leur profil et de celui du demandeur, le demandeur reçoit des réponses qui lui sont pertinentes et personnalisées.

Framework pour simuler des utilisateurs de moteurs de recherche Les modèles des utilisateurs de système d'échange de fichiers se révèlent incomplets pour simuler des utilisateurs de moteur de recherche. Nous avons proposé un modèle des utilisateurs de moteur de recherche en décrivant les les distributions entre les utilisateurs, documents et mots en s'appuyant sur des études d'utilisateurs de système de recherche, sur des études bibliométriques et sur des études sémantiques.

Nous avons ensuite défini le comportement des utilisateurs : émissions de requêtes de recherche (choix des mots utilisés), téléchargements des documents apparaissant dans les réponses de recherche.

Puis nous avons défini une mesure de qualité décrivant l'appréciation d'un document d pour un utilisateur et une requête de recherche. Nous avons pour cela supposé que cette qualité dépendait des liens sémantiques et des liens sociaux entre le document d et les documents connus par l'utilisateur. Nous avons ensuite utilisé cette qualité pour définir un ensemble de mesures pour évaluer l'efficacité du système à fournir des résultats pertinents et personnalisés à ses utilisateurs.

Avec ce modèle des utilisateurs, la description de leurs comportements et les mesures d'efficacité, nous avons pu simuler notre système et l'évaluer.

Spécification de l'architecture et des fonctionnalités d'un système distribué Nous avons proposé une implémentation de notre système de recherche. Nous avons défini l'architecture utilisée ainsi que les interactions entre l'utilisateur et le système MAAY. Le système offre la possibilité aux utilisateurs d'effectuer des recherches locales sur les données présentes sur leur disque dur ou de rechercher de manière collaborative les données partagées par les autres utilisateurs. Les utilisateurs peuvent aussi indexer les pages web qu'ils visitent et les annoter, ce qui permet d'indexer les pages du deep web, d'améliorer la disponibilité des pages du web et d'effectuer des recherches dans le passé du web.

7.2 Perspectives

Nous allons décrire dans cette section, l'ensemble des travaux qui pourront être entrepris pour compléter et améliorer notre système.

Dans une première partie, nous verrons les améliorations possibles des algorithmes utilisés dans MAAY. Nous décrirons quelques critères qui pourront être utilisés dans le routage des requêtes et le classement des résultats. Nous proposerons ensuite dans une deuxième partie les fonctionnalités qui pourront être implémentées dans le système MAAY afin de le rendre plus complet. Enfin, nous verrons comment améliorer le modèle des utilisateurs grâce aux traces que nous pourrons obtenir après le déploiement du système.

7.2.1 Amélioration des algorithmes de recherche de Maay

7.2.1.1 Critères de classement des documents

Le classement de documents dans MAAY s'appuient sur trois critères que sont la pertinence, la popularité et l'adéquation avec le profil du demandeur. Mais d'autres critères peuvent être utilisés à la place ou en combinaison avec ceux déjà mis en place.

Utilisation des fréquences des mots On peut utiliser des critères se basant sur la fréquence des mots comme TFxIDF présenté dans la section 2.1.4.1 comme indicateur de pertinence d'un document pour un mot.

Popularité d'un document Un document est d'autant plus populaire qu'il est lu par beaucoup de personnes. Un nœud peut estimer localement la popularité d'un document en comptant le nombre de ses fournisseurs qu'il connaît grâce aux réponses de recherche reçues.

Liens hypertextes entre page web Il peut être intéressant d'utiliser un critère se basant sur les liens hypertextes, à l'instar de PageRank ou de HITS, pour classer les pages web indexées par les utilisateurs.

7.2.1.2 Critères de sélections de voisins

Le choix de voisins s'appuient sur trois critères que sont la spécialisation, l'expertise et l'affinité avec le profil du demandeur. D'autres critères peuvent être utilisés à la place ou en combinaison avec ceux déjà mis en place.

Liste choisie de voisins Dans MAAY les voisins sont cachés à l'utilisateur, le profil attribué aux voisins est effectué de manière automatique sans intervention de l'utilisateur. On pourrait cependant laisser le choix à l'utilisateur de définir - manuellement ou par un système de feedback explicite - sa liste de voisins privilégiés ou une liste d'amis. La recherche s'effectuerait alors un interrogeant ces voisins privilégiés ou se restreindrait aux seuls voisins amis.

Caractéristiques réseaux et matériels des voisins Le CPU, l'espace de stockage, le nombre de voisins, le nombre de fichiers partagés, sa stabilité, ... pourraient être pris en compte dans le choix des voisins à interroger. Ainsi, les nœuds ayant un grand espace de stockage pourraient jouer le rôle de dépôt de documents, alors que les nœuds ayant une bande passante élevée auraient un rôle de connecteur ou d'ultrapeer comme ceux utilisés dans Gnutella.

Comme nombre de moteurs de recherche, MAAY se base sur plusieurs critères pour définir les voisins et les documents les plus pertinents pour l'utilisateur. Trouver parmi l'infinité de critères disponibles ceux qui contribueront à améliorer la recherche est une tâche très difficile. Pour exemple, le moteur de recherche Google en utilise 150 et ces critères sont sans cesse mis à jour.

En proposant un système ouvert qui permette à tout développeur de rajouter un critère, de les combiner et d'étendre le protocole, on peut espérer que parmi les différentes versions du système, quelques unes sortiront du lot et à la manière de la sélection darwinienne, seules les versions les plus efficaces du systèmes survivront.

7.2.2 Fonctionnalités supplémentaires

7.2.2.1 Gestion des droits

Espace personnel Pour le moment, l'espace local d'un utilisateur se limite aux fichiers présents sur sa machine. Mais on peut imaginer étendre cet espace local à un espace personnel qui regroupe les données présentes sur les différentes machines de l'utilisateur.

L'utilisateur pourra effectuer des recherches dans cet espace personnel et y télécharger des documents. Cela permettra à un utilisateur d'accéder à distance à toutes ses ressources personnelles éparpillées sur plusieurs machines en quelques clics.

Droits de groupe On peut étendre la visibilité des documents en rajoutant la notion de groupe. Des documents publiés pourront n'être accessibles qu'aux membres de quelques groupes. Par exemple dans le cas d'une entreprise, des documents peuvent être soit publics, soit internes à l'entreprise ou soit personnels.

7.2.2.2 Indexation d'autres sources

Interaction avec les réseaux Gnutella, eDonkey, ... Il existe une multitude de systèmes de recherche et chacun ont leur propre espace d'information : Google a le web, Gnutella, eDonkey et Freenet ont chacun l'ensemble des fichiers partagés par leurs utilisateurs.

Il peut être intéressant d'interagir avec ces réseaux de manière à avoir l'espace d'information le plus vaste possible.

Autres sources de données possibles D'autres sources de données peuvent être indexées : les e-mails, les conversations écrites dans les logiciels de messagerie instantanée, les flux RSS qui décrivent les articles les plus récents postés sur un site, ...

7.2.2.3 Fonctionnalité web

Archivage du web En améliorant le système d'indexation des pages web de notre système, nous pourrions obtenir un système où les pages web visitées par les utilisateurs pourraient être archivées efficacement et cherchées par date et par URL à la manière du système d'archivage Internet Archive WayBack Machine[1].

Gestion des liens hypertextes Les liens hypertextes présents dans les pages HTML pointent vers d'autres pages HTML. Or les pages pointées par une page a peuvent avoir été modifiées ou avoir disparu au moment de la lecture de la page a. De plus, on ne connaît pas l'intention de l'auteur de la page qui peut aussi bien avoir voulu créer un lien vers la version la plus à jour de la page que vers sa version au moment de la création du lien.

Pour résoudre ce problème, on pourrait proposer à l'utilisateur de choisir la version de la page qu'il veut visiter lorsqu'il clique sur un lien hypertexte ou lorsqu'il visite une page web.

Publication de sites web Offrir la possibilité aux utilisateurs de publier un ensemble de pages reliées entre elles par des liens hypertextes peut être intéressant. Des travaux sur le format des liens, du nommage des ressources et l'accès aux pages pointées doivent être effectués.

[1] Internet Archive WayBack Machine. `http://www.archive.org`.

7.2.3 Statistiques

Récupérer des statistiques sur le système MAAY pourra être intéressant pour pouvoir caractériser les usages qu'en font les utilisateurs et leur proposer de nouvelles fonctionnalités. Elles pourront aussi être utilisées pour améliorer notre modèle des utilisateurs. Notamment pour établir des corrélations entre les documents partagés, les requêtes de recherche tapés par les utilisateurs et entre les documents téléchargés. Grâce aux traces, nous pourrions proposer un benchmark décrivant les documents partagés par les utilisateurs, les requêtes de recherche qu'ils ont effectuées et les documents qu'ils ont téléchargés.

Bibliographie

Moteurs de recherche

[1] Google. `http://www.google.com`.

[2] Yahoo. `http://www.yahoo.com`.

[3] Michael K. Bergman. The Deep Web : Surfacing Hidden Value. *The journal of electronic publishing (JEP). Published by the University of Michigan Press*, July 2001.

[4] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1–7) :107–117, 1998.

[5] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph Structure in the Web. *Computer Networks*, 33(1-6) :309–320, 2000.

[6] Charles H. Ferguson. What's Next for Google. *MIT Technology Review*, 26 Dec 2004.

[7] Maria Halkidi, Benjamin Nguyen, Iraklis Varlamis, and Michalis Vazirgiannis. THESUS : Organizing Web Document Collections Based on Link Semantics. *The VLDB Journal*, 12(4) :320–332, 2003.

[8] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Proceedings of the 9th ACM Symposium on Discrete Algorithms (SIAM), 1998.*, 46(5) :604–632, 1999.

[9] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking : Bringing Order to the Web. *Stanford Digital Library Technologies Project*, 1998.

[10] Athanasios Papagelis and Christos Zaroliagis. On Building Search Engines Based on User Information Spaces. *Unpublished manuscript*, 2005.

[11] Chris Riding and Moke Shishgin. Pagerank uncovered. 2002.

[12] Ian Rogers. The Google Page Rank Algorithm and How It Works. 2002.

[13] Scott Deerwester and Susan T. Dumais and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science, 41, 391-407*, 1990.

[14] A. Spink, D. Wolfram, B. J. Jansen, and T. Saracevic. Searching the Web : the Public and their Queries. *Journal of the American Society of Information Science and Technology*, 52 :226–234, 2001.

[15] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive Web Search Based on User Profile Constructed Without any Effort from Users. *Proceedings of the 13th international conference on World Wide Web (WWW 2004), New York, NY, USA*, pages 675–684, May 17-20 2004.

— Etudes de traces —

[16] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000.

[17] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like Distributions : Evidence and Implications. *Proceedings IEEE INFOCOM '99, The Conference on Computer Communications, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, The Future Is Now, New York, NY, USA*, March 21-25 1999.

[18] Zihui Ge, Daniel R. Figueiredo, Sharad Jaiswal, James F. Kurose, and Donald F. Towsley. Modeling Peer-Peer File Sharing Systems. *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA*, March 30 - April 3 2003.

[19] Bernard J. Jansen and Amanda Spink. An Analysis of Web Documents Retrieved and Viewed. *Proceedings of the International Conference on Internet Computing, IC '03, Las Vegas, Nevada, USA, June 23-26, 2003, Volume 1*, pages 65–69, 2003.

[20] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real Life Information Retrieval : A Study of User Queries on the Web. *ACM SIGIR Forum*, 32(1) :5–17, 1998.

[21] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real Life, Real Users, and Real Needs : A Study and Analysis of User Queries on the Web. *Information Processing and Management*, 36(2) :207–227, 2000.

[22] Craig Silverstein, Monika Rauch Henzinger, Hannes Marais, and Michael Moricz. Analysis of a Very Large Web Search Engine Query Log. *ACM SIGIR Forum*, 33(1) :6–12, 1999.

Systèmes Pair à Pair

[23] M. Bawa, R. J. Bayardo Jr., S. Rajagopalan, and E. Shekita. Make it Fresh, Make it Quick – Searching a Network of Personal Webservers. *Proceedings of the Twelfth ACM International World Wide Web Conference (WWW2003), Budapest, Hungary*, May 20-24 2003.

[24] Roberto J. Bayardo Jr., Rakesh Agrawal, Daniel Gruhl, and Amit Somani. YouServ : a Web-Hosting and Content Sharing Tool for the Masses. *Proceedings of the Eleventh International ACM World Wide Web Conference (WWW2002) Honolulu, Hawaii, USA*, May 7-11 2002.

[25] O'Reilly. *Peer-to-peer : Harnessing the Power of Disruptive Technologies.* 2001.

— Non structurés —

[26] The Gnutella Developer Forum(GDF). `http://groups.yahoo.com/group/the_gdf/`.

[27] Michael D.Smith Atip Asvanund, Ramayya Krishnan and Rahul Telang. Interest-Based Self-Organizing Peer-to-Peer Networks : A Club Economics Approach. 2004.

[28] Clip2 Distributed Search Solutions. The Gnutella Protocol Specification v0.4. 2001.

[29] Tyson Condie, Sepandar D. Kamvar, and Hector Garcia-Molina. Adaptive Peer-to-Peer Topologies. *IEEE Computer Society. 4th International Conference on Peer-to-Peer Computing (P2P 2004), Zurich, Switzerland*, pages 53–62, August 15-17 2004.

[30] S. Daswani and A. Fisk. Gnutella UDP Extension for Scalable Search (GUESS) v0.1, August 2002.

[31] Pierre Fraigniaud, Philippe Gauron, and Matthieu Latapy. Combining the Use of Clustering and Scale-Free Nature of User Exchanges into a Simple and Efficient P2P System. *Proceedings of the 11th International Euro-Par Conference, Lisbon, Portugal*, August 30 - September 2 2005.

[32] Adriana Iamnitchi, Matei Ripeanu, and Ian T. Foster. Locating Data in (Small-World ?) Peer-to-Peer Scientific Collaborations. *First International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, Revised Papers.*

[33] T. Klingberg and R. Manfredi. Gnutella 0.6. June 2002.

[34] Kiyohide Nakauchi, Yuichi Ishikawa, Hiroyuki Morikawa, and Tomonori Aoyama. Peer-to-Peer Keyword Search Using Keyword Relationship. *3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2003), Tokyo, Japan*, pages 359–366, May 12-15 2003.

[35] Sharman Networks. Kazaa. `http://www.kazaa.com`.

[36] S. Osokine. Search Optimization in the Distributed Networks. available at `http://www.grouter.net/gnutella/search.htm`. 15 Oct. 2002.

[37] Yi Ren, Chaofeng Sha, Weining Qian, and Aoying Zhou. Explore the Small World Phenomena in Pure P2P Information Sharing System. *3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2003), Tokyo, Japan. IEEE Computer Society 2003, ISBN 0-7695-1919-9*, May 12-15 2003.

[38] Jordan Ritter. Why Gnutella Can't Scale. No, Really. February 2001.

[39] Christopher Rohrs. Query Routing for the Gnutella Network. Version 0.4. *Limewire LLC*, December 18 2001.

[40] Nima Sarshar, P. Oscar Boykin, and Vwani P. Roychowdhury. Percolation Search in Power Law Networks : Making Unstructured Peer-to-Peer Networks Scalable. *4th International Conference on Peer-to-Peer Computing (P2P 2004), Zurich, Switzerland*, pages 2–9, August 15-17 2004.

[41] Anurag Singla and Christopher Rohrs. Ultrapeers : Another Step Towards Gnutella Scalability. Version 1.0, November 11 2002.

[42] Chen Wang, Li Xiao, Yunhao Liu, and Pei Zheng. Distributed Caching and Adaptive Search in Multilayer P2P Networks. *4th International Conference on Distributed Computing Systems (ICDCS 2004), Hachioji, Tokyo, Japan*, pages 219–226, March 24-26 2004.

[43] Beverly Yang and Hector Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01), Roma, Italy*, September 11-14 2001.

[44] Beverly Yang and Hector Garcia-Molina. Improving Search in Peer-to-Peer Networks. *In Proceedings of the 22th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria*, pages 5–14, July 2002.

[45] Beverly Yang and Hector Garcia-Molina. Designing a Super-Peer Network. *Proceedings of the 19th International Conference on Data Engineering (ICDE), Bangalore, India*, March 2003.

[46] Beverly Yang, Patrick Vinograd, and Hector Garcia-Molina. Evaluating GUESS and Non-Forwarding Peer-to-Peer Search. In *24th International Conference on Distributed Computing Systems (ICDCS 2004), Hachioji, Tokyo, Japan*, pages 209–218, 24-26 March 2004.

— Routage sémantique —

[47] Njål T. Borch. Improving Semantic Routing Efficiency. *Second International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P'05), San Diego, USA*, July 2005.

[48] Njål T. Borch and Lars Kristian Vognild. Searching in Variably Connected P2P Networks. *Proceedings of the International Conference on Wireless Networks, ICWN '04, Las Vegas, Nevada, USA*, June 21-24 2004.

[49] Vicent Cholvi, Pascal Felber, and Ernst W. Biersack. Efficient Search in Unstructured Peer-to-Peer Networks. *Proceedings of the Sixteenth Annual ACM Symposium on Parallel Algorithms (SPAA 2004), Barcelona, Spain*, June 27-30 2004.

[50] Ian Clarke. Freenet's Next Generation Routing Protocol, July 2003.

[51] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1) :40–49, 2002.

[52] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 23–, July 2002.

[53] Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. PlanetP : Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. *12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), Seattle, WA, USA*, pages 236–249, June 22-24 2003.

[54] Sam Joseph. NeuroGrid : Semantically Routing Queries in Peer-to-Peer Networks. *Web Engineering and Peer-to-Peer Computing, NETWORKING 2002 Workshops, Pisa, Italy, Revised Papers*, pages 202–214, May 19-24 2002.

[55] Vana Kalogeraki, Dimitrios Gunopulos, and Demetrios Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA*, pages 300–307, November 4-9 2002.

[56] Joaquín Keller, Frédéric Dang-Ngoc, and Daniel Stern. Maay : A Self-Adaptive Peer Network for Efficient Document Search. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '03, Las Vegas, Nevada, USA, Volume 2. CSREA Press 2003, ISBN 1-892512-42-4*, June 23-26 2003.

[57] Amr Z Kronfol. FASD : A Fault-Tolerant, Adaptive, Scalable, Distributed Search Engine. *PhD Thesis*, 2002.

[58] K. Sripanidkulchai, B. M. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA*, March 30 - April 3 2003.

[59] Dimitrios Tsoumakos and Nick Roussopoulos. Adaptive Probabilistic Search for Peer-to-Peer Networks. *3rd International Conference on Peer-to-Peer Computing (P2P 2003), Linköping, Sweden*, September 1-3 2003.

[60] S. Voulgaris, A-M. Kermarrec, L. Massoulié, and M. Van Steen. Exploiting Semantic Proximity in Peer-to-Peer Content Searching. *10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004), Suzhou, China*, May 26-28 2004.

— Structurés —

[61] Francisco Matias Cuenca-Acuna and Thu D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. *Web Engineering and Peer-to-Peer Computing, NETWORKING 2002 Workshops, Pisa, Italy, May 19-24, 2002, Revised Papers. Lecture Notes in Computer Science 2376 Springer 2002, ISBN 3-540-44177-8*, May 19-24 2002.

[62] Pierre Fraigniaud and Philippe Gauron. Brief Announcement : an Overview of the Content-Addressable Network D2B. *Proceedings of the 22th ACM Symposium on Principles of Distributed Computing (PODC 2003), Boston, Massachusetts, USA*, July 13-16 2003.

[63] Michael J. Freeman and David Mazière. Sloppy Hashing and Self-Organizing Clusters. *Second International Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, CA, USA, Revised Papers*, February 21-22 2003.

[64] Anh-Tuan Gai and Laurent Viennot. Broose : A Practical Distributed Hashtable Based on the De-Bruijn Topology. *4th International Conference on Peer-to-Peer Computing (P2P 2004), 15-17 August 2004, Zurich, Switzerland*, pages 167–164, 2004.

[65] Omprakash D Gnawali. A Keyword-Set Search System for Peer-to-Peer Networks. *Master's thesis, Masachusetts Institute of Technology*, 2002.

[66] Sitaram Iyer, Antony I. T. Rowstron, and Peter Druschel. Squirrel : a Decentralized Peer-to-Peer Web Cache. *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing (PODC), July 21-24, 2002 Monterey, California, USA.*, pages 213–222, 2002.

[67] Jinyang Li, Boon Thau Loo, Joe Hellerstein, Frans Kaashoek, David R. Karger, and Robert Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. *Second International Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, CA, USA, Revised Papers. Lecture Notes in Computer Science 2735 Springer 2003, ISBN 3-540-40724-3 BibTeX*, February 21-22 2003.

[68] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. Semantic Small World : An Overlay Network for Peer-to-Peer Search. *12th IEEE International Conference on Network Protocols (ICNP 2004), 5-8 October 2004, Berlin, Germany*, pages 228–238, 2004.

[69] Lintao Liu, Kyung Dong Ryu, and Kang-Won Lee. Supporting Efficient Keyword-based File Search in Peer-to-Peer File Sharing Systems. *IEEE Global Internet and Next Generation Networks Symposium, Dallas, TX*, November 2004.

[70] Petar Maymounkov and David Mazières. Kademlia : A Peer-to-Peer Information System Based on the XOR Metric. *First International Workshop on Peer-to-Peer Systems (IPTPS'02) Cambridge, MA, USA, Revised Papers*, pages 53–65, March 7-8 2002.

[71] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario Schlosser, Ingo Brunkhorst, and Alexander Löser. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. *Proceedings of the Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary. ACM, 2003*, pages 536–543, May 20-24 2003.

[72] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A Scalable Content-Addressable Network. *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, San Diego, CA, USA*, pages 161–172, August 27-31 2001.

[73] Patrick Reynolds and Amin Vahdat. Efficient Peer-to-Peer Keyword Searching. *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003), Rio de Janeiro, Brazil*, pages 21–40, June 16-20 2003.

[74] Antony Rowstron and Peter Druschel. Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany. Lecture Notes in Computer Science 2218 Springer 2001, ISBN 3-540-42800-3*, 2218, November 12-16 2001.

[75] Shuming Shi. Making Peer-to-Peer Keyword Searching Feasible Using Multi-Level Partitioning. *In Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04) San Diego, CA, USA*, pages 151–161, February 26-27 2004.

[76] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord : A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transaction on Networking*, 11(1) :17–32, February 2003.

[77] Torsten Suel, Chandan Mathur, Jo wen Wu, Jiangong Zhang, Alex Delis, Mehdi Kharrazi, Xiaohui Long, and Kulesh Shanmugasundaram. ODISSEA : A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. *International Workshop on Web and Databases (WebDB'03), San Diego, California*, June 12-13 2003.

[78] Chunqiang Tang and Sandhya Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004), San Francisco, California, USA*, pages 211–224, March 29-31 2004.

[79] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. PeerSearch : Efficient Information Retrieval in Peer-Peer Networks. Technical Report HPL-2002-198, Hewlett-Packard Labs, 2002.

[80] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. pSearch : Information Retrieval in Structured Overlays. *First Workshop on Hot Topics in Networks (HotNets I). Princeton, NJ, October 2002. Also appears in Computer Communication Review, 33(1) :89-94*, January 2003.

[81] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry : An Infrastructure for Fault-tolerant Wide-Area Location and Routing. *Computer Communication Review (ACM SIGCOMM)*, 32(1) :81, 2002.

— Etudes de traces —

[82] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000.

[83] Fabrice Le Fessant, Sidath B. Handurukande, Anne-Marie Kermarrec, and Laurent Massoulié. Clustering in Peer-to-Peer File Sharing Workloads. *Third International Workshop on Peer-to-Peer Systems (IPTPS'04), La Jolla, CA, USA, Revised Selected Papers*, pages 217–226, February 26-27 2004.

[84] Zihui Ge, Daniel R. Figueiredo, Sharad Jaiswal, James F. Kurose, and Donald F. Towsley. Modeling Peer-Peer File Sharing Systems. *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA*, March 30 - April 3 2003.

[85] P. Krishna Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer

File-Sharing Workload. *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY USA*, pages 314–329, October 19-22 2003.

[86] Kunwadee Sripanidkulchai. The Popularity of Gnutella Queries and Its Implications on Scalability. *Featured on O'Reilly's www.openp2p.com website*, February 2001.

[87] Jian Liang, Rakesh Kumar, and Keith W. Ross. The KaZaA Overlay : A Measurement Study. *19th IEEE Annual Computer Communications Workshop (CCW'04). Bonita Springs, Florida*, October 17-20 2004.

[88] Przemmmyslaw Makosiej, German Sakaryan, and Herwig Unger. Measurement Study of Shared Content and User Request Structure in Peer-to-Peer Gnutella Network. *In proceedings of the Conference on Design, Analysis, and Simulation of Distributed Systems (DASD 2004)*, pages 115–124, April 2004.

[89] Matei Ripeanu and Ian T. Foster. Mapping the Gnutella Network : Macroscopic Properties of Large-Scale Peer-to-Peer Systems. *First International Workshop on Peer-to-Peer Systems (IPTPS 2002), Cambridge, MA, USA, Revised Papers*, pages 85–93, March 7-8 2002.

[90] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems*, 9(2) :170–184, 2003.

[91] Daniel Stutzbach and Reza Rejaie. Characterizing Today's Gnutella Topology. *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (ACM SIGMETRICS), Poster Session, Alberta Canada*, June 6-10 2005.

[92] Yinglian Xie and David R. O'Hallaron. Locality in Search Engine Queries and Its Implications for Caching. In *Proceedings IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA. ISBN 0-7803-7477-0*, June 23-27 2002.

— Modelisation —

[93] Brian F. Cooper. A Content Model for Evaluating Peer-to-Peer Searching Techniques. *Proceedings of the ACM/IFIP/USENIX 5th International Middleware Conference (Middleware 2004), Toronto. Lecture Notes in Computer Science 3231 Springer 2004, ISBN 3-540-23428-4*, October 18-20 2004.

[94] M. Jovanovic, F.S. Annexstein, and K.A. Berman. Modeling Peer-to-Peer Network Topologies through Small-World Models and Power Laws. *In Telecommunications Forum, Belgrade, Yugoslavia, November, 2001*, November 2001.

[95] Mario T. Schlosser, Tyson E. Condie, and Sepandar D. Kamvar. Simulating A File-Sharing P2P Network. *First Workshop on Semantics in P2P and Grid Computing*, December 2002.

Réseaux complexes

[96] Alexei Vazquez. Statistics of Citation Networks. *2001. cond-mat/0105031*, 2001.

[97] Albert-László Barabási. *Linked : The New Science of Networks.* Perseus Publishing, 2002.

[98] Albert-László Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science 286, 509-512 (1999)*, 1999.

[99] Ergin Elmacioglu and Dongwon Lee. On Six Degrees of Separation in DBLP-DB and More. *ACM SIGMOD Record, Vol. 34, No. 2*, June 2005.

[100] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-law Relationships of the Internet Topology. *Proceedings of the ACM SIGCOMM '99 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Cambridge, MA, USA*, August 30 - September 3 1999.

[101] J-L. Guillaume and M. Latapy. Bipartite Graphs as Models of Complex Networks. *Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'04)*, August 5-7 2004.

[102] R. Ferrer i Cancho and R. V. Solé. The Small World of Human Language. *The Royal Society of London*, 268(1482) :2261–2265, 2001.

[103] A. Iamnitchi, M. Ripeanu, and I. T. Foster. Small-World File-Sharing Communities. *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China*, March 7-11 2004.

[104] Konstantin Klemm and Victor M. Eguiluz. Growing Scale-Free Networks with Small-World Behavior. *Physical review E, Volume 65, 057102*, May 2002.

[105] Vaishnavi Krishnamurthy, Junhong Sun, Michalis Fabloutsos, and Sudhir Tauro. Sampling Internet Topologies : How Small Can We Go ? *Proceedings of the International Conference on Internet Computing (IC '03), Las Vegas, Nevada, USA*, 2 :577–580, June 23-26 2003.

[106] M.E.J. Newman. The Structure of Scientific Collaboration Networks. *Proceedings of the National Academy of Sciences*, 98(2) :404–409, 2001.

[107] Sidney Redner. How Popular is Your Paper ? An Empirical Study of the Citation Distribution. *European Physical Journal. B 4, 131-134 (1998)*, 1998.

[108] Mark Steyvers and Joshua B. Tenenbaum. The Large Scale Structure of Semantic Networks : Statistical Analyses and a Model of Semantic Growth. *Cognitive Science, 29(1), 41-78*, 2004.

[109] Duncan J. Watts. *Six Degrees : the Science of a Connected Age.* W. W. Norton, 2003.

[110] Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature, 393 :440-442 (1998)*, 1998.

Divers

[111] DBLP Computer Science Bibliography. `http://dblp.uni-trier.de/`.

[112] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7) :422–426, 1970.

[113] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 301 :13–30, 1963.

[114] Glen Jeh and Jennifer Widom. SimRank : A Measure of Structural-Context Similarity. *In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada*, July 2002.

[115] Yutaka Matsuo, Yukio Ohsawa, and Mitsuru Ishizuka. A Document as a Small World. *Japanese Society of Artificial Intelligence Workshops (JSAI 2001)*, 2253 :444–448, 2001.

[116] David P. Reed. That Sneaky Exponential - Beyond Melcalfe's Law to the Power of Community Building. *Context magazine*, 1999.