
Un simulateur générique pour les systèmes de recherche d'informations en pair-à-pair

Jérôme Sicard* — **Bruno Defude*** — **Raja Chiky**** — **Georges Hébrail****
— **Sorin Moga*****

* : *Institut National des Télécommunications*
9 rue CharlesFourier
91011 Évry cedex
FRANCE
{Jerome.sicard,bruno.defude}@int-evy.fr

** : *Ecole Nationale Supérieure des Télécommunications*
46, rue Barrault
75013, Paris
FRANCE
{chiky,hebrail}@enst.fr

*** : *GET-ENST Bretagne*
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
sorin.moga@enst-bretagne.fr

RÉSUMÉ. Le passage à l'échelle des systèmes de recherche d'informations est un problème crucial aujourd'hui, avec l'avènement d'Internet et l'augmentation vertigineuse des sources d'informations disponibles. Une solution possible pour faire face à ce défi est de construire des systèmes de recherche d'informations selon une architecture pair-à-pair. Les systèmes pair-à-pair (peer-to-peer ou P2P) ont été popularisés par les systèmes de partage de fichier sur Internet comme Gnutella et Kazaa. Ils permettent notamment la montée en charge en nombre de noeuds et la dynamique du réseau. De nombreux travaux ont proposés ces dernières années d'adapter cette architecture aux systèmes de recherche d'informations. Le domaine reste encore neuf et il y a un besoin important de mieux appréhender les principes de conception à suivre pour de tels systèmes. Cet article apporte une double contribution sur ce sujet. D'une part nous proposons une architecture générique des systèmes de recherche d'informations en P2P. Nous montrons ensuite comment plusieurs propositions récentes peuvent se décrire à l'aide de cette architecture. D'autre part, nous présentons un simulateur générique de systèmes de recherche

d'informations P2P inspiré de cette architecture, qui permet à des concepteurs de nouveaux systèmes de pouvoir facilement simuler leurs algorithmes avant de les implanter.

ABSTRACT. The scalability of information retrieval systems is an important challenge today due to Internet development and the exponential increase of accessible data sources. A possible solution to tackle with this challenge is to define new information retrieval systems based on the peer-to-peer (P2P) paradigm. P2P systems have been popularized these last years with popular file-sharing systems on the Internet like Gnutella or Kazaa. They allow a good scalability together with a good support for dynamic networks. Several proposals have been done recently to adapt this paradigm to information retrieval systems, but the area is still in its infancy and there is a need to gain a thorough understanding of the design of such systems. In this article we present two contributions to this problem. First we propose a generic software architecture for P2P information retrieval systems. This architecture is validated with the description of Gnutella, PlanetP and Nakauchi's proposal. We also present a generic simulator for P2P information retrieval systems inspired from our generic architecture. This simulator allow designers of new systems to easily simulate their ideas and algorithms before implementing them.

MOTS-CLÉS : systèmes de recherche d'informations, pair-à-pair, simulation, architecture générique, évaluation de performances.

KEYWORDS: information retrieval systems, peer-to-peer, simulator, generic software architecture, performance evaluation.

1. Introduction

Avec l'avènement d'Internet et l'augmentation vertigineuse des sources d'informations disponibles, le passage à l'échelle des systèmes de recherche d'informations devient aujourd'hui un problème crucial. De nombreux travaux sont faits pour permettre la construction de "crawler" et de systèmes de recherche d'information centralisés toujours plus puissants. Google [BAR 03] est un exemple édifiant de cette approche, mais malgré toute la puissance déployée (une grappe de plus de 15000 machines), seule une partie du web est indexée et la montée en charge commence à être difficile. De plus, le coût d'une telle architecture est prohibitif et empêche donc la définition de nouveaux systèmes de ce type. Une autre solution possible pour faire face à ce défi est de construire des systèmes de recherche d'informations selon une architecture complètement décentralisée de type pair-à-pair.

Les systèmes pair-à-pair (peer-to-peer ou P2P) ont été popularisés par les systèmes de partage de fichier sur Internet comme Gnutella [Gnu] et Kazaa (voir [RIS 04] pour une présentation synthétique du P2P). Ils permettent notamment la montée en charge en nombre de noeuds et la dynamique du réseau à un coût faible puisque la charge est distribuée sur l'ensemble des participants. De nombreux travaux ont proposé ces dernières années d'adapter cette architecture aux systèmes de recherche d'informations : [CRE 02], [CUE 02], [NAK 03], [TAN 03].

[CRE 02] adapte les techniques développées pour les méta-moteurs de recherche au contexte P2P. Pour cela, chaque noeud dispose d'un index sur ses voisins qui est utilisé pour sélectionner les meilleurs noeuds vers lesquels propager une requête. PlanetP [?], suit le même principe avec une version distribuée du modèle vectoriel ainsi qu'une structure d'index à base de filtres de Bloom pour limiter les communications entre les noeuds. [NAK 03] propose d'utiliser des mécanismes d'expansion de requêtes et de *relevance feedback* dans un contexte P2P. [TAN 03] utilise des tables de hachage distribuées (CAN [RAT 01]) pour stocker le résultat d'une indexation de type LSI. La navigation sémantique dans les concepts se fait alors directement via la proximité dans l'espace de recherche CAN.

Le domaine reste cependant neuf et il y a un besoin important de mieux appréhender les principes de conception à suivre pour de tels systèmes. Cet article apporte une double contribution sur ce sujet. D'une part nous proposons une architecture générique des systèmes de recherche d'informations en P2P. Cette architecture s'inspire des premières propositions de [ABE 04], mais va plus loin dans la définition des fonctions et structures de données utilisées. Nous montrons également comment les systèmes existants évoqués ci-dessus peuvent se décrire et se comparer à l'aide de cette architecture. D'autre part nous présentons un simulateur générique de systèmes de recherche d'informations P2P inspiré de notre modèle architectural. La simulation présente l'avantage de pouvoir plus facilement expérimenter des architectures très réparties (plusieurs milliers de noeuds). En effet, il est très difficile de déployer une application de cette taille sur un environnement réel. Ce simulateur est construit au-dessus d'un simulateur de systèmes P2P appelé PeerSim [Pee] et peut se voir comme une "spécialisation" de PeerSim pour la recherche d'informations. Ce simulateur va

permettre à des concepteurs de nouveaux systèmes de pouvoir facilement simuler leurs algorithmes avant de les implanter grâce à son architecture et à ces fonctionnalités dédiés à la recherche d'informations. Ce développement a été réalisé dans le cadre du projet de recherche RARE (pour Routage optimisé par Apprentissage de REquêtes) qui fédère des équipes de l'ensemble des écoles du GET. Ce projet vise à proposer un ensemble d'algorithmes permettant de construire des connaissances et de les échanger afin d'améliorer la sélection des noeuds vers lesquels propager une requête. Nous étudions deux pistes : d'une part acquérir des connaissances sur le contenu des noeuds voisins et d'autre part acquérir des connaissances par apprentissage sur les requêtes passées en utilisant du feedback positif des utilisateurs.

Le reste de l'article est organisé comme suit. Dans la section 2, nous présentons notre architecture générique pour la recherche d'informations P2P et la validons dans la section 3 sur trois exemples typiques (Gnutella, PlanetP et [NAK 03]). La section 4 décrit ensuite le simulateur générique réalisé au-dessus de PeerSim. Nous décrivons également comment l'architecture générique de la section 2 se reflète dans le simulateur. Enfin, la conclusion fait le point sur l'avancement de notre travail et présente les pistes de recherche que nous allons explorer prochainement.

2. Architecture générique des systèmes de recherche d'informations en P2P

La définition d'un système de recherche d'informations P2P résulte d'un certain nombre de choix que nous voulons ici mettre en évidence pour définir une architecture générique. Cette architecture doit pouvoir modéliser l'ensemble des systèmes existants et à venir dans ce domaine. Notre définition s'inspire de la proposition de [ABE 04] qui est à notre connaissance le seul travail allant dans ce sens.

Nous imposons un certain nombre de restrictions dans ce travail. Tout d'abord nous ne prenons en compte que les systèmes P2P et pas les approches type méta-moteurs ou médiation. Pour aller plus loin, nous ne considérons que des systèmes P2P où tous les participants jouent le même rôle. Nous ne modélisons donc pas les systèmes de type P2P hiérarchique où certains participants, les super-pairs, jouent un rôle plus important que les autres. Le point de départ de notre architecture générique est la découpe en quatre couches proposée par [ABE 04] :

niveau réseau physique (le plus bas niveau) : on trouve ici les notions de machines et d'adressage bas niveau (niveau IP) ;

niveau réseau logique : on va trouver ici les notions de pair ainsi que les mécanismes de gestion du réseau de pairs (adressage, algorithmes de diffusion, requête P2P, propagation de requête P2P) ;

niveau gestion de documents et du contenu : on va trouver ici les notions de pair gérant un ensemble de documents (avec toutes les fonctions de gestion, d'indexation et d'organisation associées) ;

niveau recherche d'informations : nous trouvons ici la définition du modèle de recherche d'information choisi.

De manière plus précise, nous définissons l'architecture générique comme un ensemble de fonctions et de structures de données structurées en trois couches (la couche réseau physique n'est pas pertinente dans notre contexte). En fonction des systèmes P2P, certaines fonctions peuvent être absentes ou au contraire avoir plusieurs instantiations. Le tableau 1 illustre notre définition.

niveau	structure de données	fonction
recherche d'informations	requête utilisateur : UQ	
	résultat utilisateur : UR	
	thésaurus local : LT	
	thésaurus "global" : GT	
		resolveGlobalQuery(UQ) -> UR
		terminationGlobalQuery(UQ, UR) -> booléen
		resolveLocalQuery(UQ) -> UR
		mergeGlobalQuery(UR, UR) -> UR
		preGlobalQuery(UQ, GT) -> UQ'
		postGlobalQuery(UQ, UR)
pair documentaire	descripteur de document : DD	
	descripteur de pair : PD	
	index local : LI	
	index "global" : GI	
		addDocument(DD)
		deleteDocument(DD)
		peerchoice(GI) -> (PD, rank)
		propagateQuery(UQ)
		preLocalQuery(UQ, LT) -> UQ'
		postLocalQuery(UQ, UR)
réseau logique	ressource : R	
	descripteur de noeud : ND	
	table de voisinage du noeud : NRT	
		join(ND)
		leave(ND)
		put(clé, R)
		get(clé, R)
	propagate(R, NRT)	

Tableau 1. modules et structures de données dans un système P2P-RI.

La définition des deux niveaux supérieurs (recherche d'information et pair documentaire) n'a pas été facile. On peut tout d'abord se demander s'il est pertinent d'avoir deux niveaux et non pas un seul : nous avons préféré deux niveaux car il nous semble intéressant de déconnecter l'aspect répartition proprement dit de l'aspect recherche d'informations. Un autre découpage possible aurait été de découper entre ce qui relève d'un pair isolé et ce qui relève du réseau. L'inconvénient de cette découpe est de segmenter la recherche d'informations.

Au niveau recherche d'informations, nous mettons en évidence la notion de requête utilisateur et celle de résultat. Le langage de requête est bien évidemment un élément important du modèle de recherche d'informations choisi. Selon les systèmes, le résultat peut intégrer une mesure de pertinence ou non. Le modèle de recherche d'informations en P2P est décrit dans la fonction principale *resolveGlobalQuery* qui va s'appuyer sur trois fonctions du même niveau : *resolveLocalQuery* qui résout la requête sur le pair local, *terminationGlobalQuery* qui détermine l'arrêt d'une recherche et *mergeGlobalResults*, qui fusionne les résultats obtenus à partir des autres pairs avec

ceux obtenus localement. Nous intégrons également à ce niveau l'usage d'autres mécanismes de recherche d'informations, comme du relevance feedback ou de l'expansion de requêtes. Nous modélisons ces mécanismes par les fonctions *preGlobalQuery* (expansion via une base de connaissance globale de la requête), *preLocalQuery* (expansion via une base de connaissance locale), *postGlobalQuery* (feedback utilisateur), *postLocalQuery* (feedback "système"). Ces fonctions utilisent un thésaurus local et "global" (c'est à dire qui représente de l'information en provenance de plusieurs pairs).

Au niveau d'un pair de document, nous mettons en évidence deux index : l'index local décrit le contenu du pair (fichier inverse par exemple) alors que l'index "global" décrit le contenu d'autres pairs de documents (pas forcément tous). Les fonctions proposées sont des fonctions documentaires basiques (ajout et retrait d'un document), mais surtout les fonctions qui vont gérer l'aspect réparti d'une recherche en P2P. La fonction *peerchoice* va choisir les "meilleurs" pairs vers lesquels propager une requête. Cette fonction va utiliser l'information provenant de l'index global pour cela. La fonction *propagateQuery* va ensuite réaliser la propagation proprement dite. Enfin, la fonction *globalconsistency* va gérer la mise à jour de manière distribuée des structures de données globales (index global et thésaurus global).

Au niveau d'un noeud, nous voulons considérer ici aussi bien une approche non structurée comme Gnutella [Gnu] que des approches structurées de type DHT (Distributed Hash Tables, voir par exemple [RAT 01]). Nous supposons donc une interface abstraite avec des fonctions *join* et *leave* pour matérialiser l'aspect dynamique du réseau, *put* et *get* pour prendre en compte le stockage et l'accès aux ressources (pas seulement des documents mais aussi des structures de données internes au système) et *propagate* qui permet de diffuser une ressource à un ensemble de noeuds.

Pour valider cette proposition d'architecture générique, nous décrivons dans la section suivante les systèmes Gnutella [Gnu], PlanetP [CUE 02] et [NAK 03].

3. Validation de l'architecture générique

Nous avons choisi de modéliser trois systèmes de recherche d'informations en P2P. Le premier est Gnutella [Gnu] qui, même s'il est faible sur la dimension recherche d'informations, est très représentatif du P2P. Le second est PlanetP [CUE 02], qui implante une version P2P du modèle vectoriel de recherche d'informations et le dernier est [NAK03], qui utilise des fonctions spécifiques de recherche d'information comme l'expansion de requêtes et la prise en compte du feedback. Nous décrivons tout d'abord succinctement ces trois systèmes avant de montrer leur modélisation dans notre architecture générique.

3.1. Description des systèmes de recherche d'informations en P2P

Gnutella est un système de partage de fichiers sur Internet ; est aujourd'hui l'archétype des systèmes P2P non structurés. Dans Gnutella tous les noeuds peuvent stocker

des documents et peuvent émettre des requêtes de recherche. Une requête est une équation booléenne portant sur les méta-données décrivant les fichiers partagés. Une requête est d'abord évaluée sur le noeud d'émission puis propagée récursivement sur un sous-ensemble aléatoire des noeuds voisins (principe d'inondation). La terminaison de la recherche est assurée par une détection de cycles ainsi qu'une profondeur maximale de recherche (appelée TTL ou Time To Live). Les noeuds possédant une réponse renvoient un descripteur vers le noeud émetteur et celui-ci peut alors choisir quel fichier il va pouvoir télécharger. Au niveau recherche d'informations, Gnutella est assez simpliste. Il n'y a aucune garantie dans la recherche (même si un fichier répondant à la requête existe, il peut ne pas être retrouvé) et le nombre de messages échangés est très grand. Le modèle de recherche d'informations est également simple (équation booléenne).

PlanetP essaye, lui, d'adapter le modèle vectoriel de recherche d'informations dans un contexte P2P. Chaque noeud indexe classiquement par le modèle vectoriel l'ensemble des documents qu'il stocke. Pour limiter l'inondation des requêtes et améliorer les performances du système, PlanetP propose de construire un index global des noeuds du système. Pour que cet index soit de taille raisonnable, PlanetP utilise une représentation compacte de l'index, appelée filtre de Bloom [BLO 70] (basés sur l'utilisation de fonctions de hachage sur les termes des documents). Pour chaque noeud on dispose donc d'un filtre de Bloom qui va être ensuite utilisé pour choisir les meilleurs noeuds correspondant à une requête donnée (on compare le filtre de Bloom de la requête avec les filtres des noeuds). Cette comparaison se base sur une variante du modèle vectoriel où la fréquence globale des termes dans l'ensemble des documents est remplacée par le nombre total de documents dans lesquels le terme apparaît (dans un contexte P2P le maintien de la fréquence totale est trop coûteux). Le problème est d'assurer la cohérence globale des index sur les différents noeuds (comment propager les mises à jour entre les différents noeuds). PlanetP propose d'utiliser un algorithme épidémique inspiré de ceux utilisés dans la gestion des copies multiples dans les bases de données répliquées. L'algorithme permet d'assurer une convergence vers un état global stable si le taux de mises à jour n'est pas trop important. PlanetP est beaucoup plus représentatif des systèmes de recherche d'informations; par contre, le fait de maintenir un index global à tous les noeuds empêche le passage à l'échelle du système (rien n'interdit cependant de ne maintenir qu'un index partiel).

[NAK 03] propose un système de recherche d'informations simple proche de Gnutella mais qui intègre un mécanisme d'expansion de requêtes. L'idée est que chaque noeud maintient un ensemble de relations binaires pondérées entre termes dans un thésaurus. Une requête avant d'être évaluée localement est expansée en utilisant le thésaurus. Par contre, c'est toujours la requête initiale qui est propagée vers les voisins, pour éviter de trop dégrader l'information initiale. Pour faire évoluer le thésaurus, les auteurs proposent deux mécanismes. Le premier est basé sur le feedback positif de l'utilisateur et permet d'augmenter la valeur des pondérations des relations qui ont joué un rôle positif dans l'obtention du résultat. Le deuxième permet à deux noeuds d'échanger leur thésaurus et de chercher à "augmenter" leur thésaurus local par des relations en provenance de l'autre noeud. Cet échange ne se fait que si l'intersection

niveau	structure de données ou fonction	Gnutella	planetP	Nakauchi
recherche d'informations	requête utilisateur : UQ	sur métadonnées	vecteur de termes	sur métadonnées
	résultat utilisateur : UR	ensemble de descripteurs de documents	ensemble des k meilleurs documents	ensemble de descripteurs de documents avec pondération
	thésaurus local : LT	NON	NON	ensemble de relations binaires entre termes avec pondération
	thésaurus "global" : GT	NON	NON	ensemble de relations binaires entre termes avec pondération
	terminaisonGlobalQuery (UQ, UR) -> booléen	TTL	plus d'obtention de bons résultats	TTL
	resolveLocalQuery(UQ) -> UR	modèle booléen	modèle vectoriel	modèle booléen
	mergeGlobalQuery(UR, UR) -> UR	union	sélection des k meilleurs résultats	union
	preGlobalQuery(UQ, GT) -> UQ'	NON	NON	NON
	postGlobalQuery(UQ, UR)	NON	NON	feedback utilisateur
	preLocalQuery(UQ, LT) -> UQ'	NON	NON	expansion en utilisant les thésaurus (local et global)
	postLocalQuery(UQ, UR)	NON	NON	NON
pair documentaire	descripteur de document : DD			
	descripteur de pair : PD			
	index local : LI	OUI	OUI	OUI
	index "global" : GI	NON	fi ltre de Bloom de l'ensemble des pairs	NON
	addDocument (DD)	OUI	OUI	OUI
	deleteDocument (DD)	OUI	OUI	OUI
	peerchoice(GI) -> (PD, rank)	aléatoire	"meilleurs" pairs, comparaison entre fi ltres de Bloom (requête et pairs)	aléatoire
	propagateQuery(UQ)	OUI	OUI	OUI
	globalconsistency(structure données globale)	NON	gossiping index global	amélioration des pondérations du thésaurus local par feedback, augmentation du thésaurus global par échange entre pairs

Tableau 2. *Instanciation de l'architecture générique sur les trois systèmes.*

entre les deux thésaurus est suffisamment grande. Ce système montre que l'on peut adapter au contexte P2P des mécanismes spécifiques à la recherche d'informations comme l'expansion de requêtes et le feedback utilisateur.

3.2. Modélisation des systèmes de recherche d'informations en P2P

Le tableau 3.2 montre comment nous avons instancié l'architecture générique sur chaque système. Nous n'avons pas fait apparaître le niveau réseau logique car il n'y a pas de différences entre ces systèmes à ce niveau (ils sont tous de type P2P non structuré).

Tous les concepts introduits par les différents systèmes rentrent dans notre architecture générique. Sans surprise, Gnutella qui est faible sur la recherche d'informations remplit peu les fonctions du niveau haut. [NAK 03] utilise un thésaurus que nous avons choisi de séparer entre partie locale et globale car les fonctions de mise à jour associées sont différentes. PlanetP a besoin d'un index global pour améliorer la propagation des requêtes en les guidant vers les "meilleurs" pairs. On peut noter

que certaines fonctions sont non-instanciées pour l'ensemble des systèmes (preGlobalQuery et postLocalQuery), ce qui peut remettre en cause leur utilité.

Cette architecture est utile pour comprendre et comparer des systèmes existants ainsi que pour guider le processus de conception d'un nouveau système. À notre avis, le point crucial dans la définition d'un système P2P concerne les connaissances partagées entre les pairs. Pour introduire de l'efficacité dans la propagation des requêtes, il faut de la connaissance sur les autres pairs. Mais maintenir cette connaissance a un coût. Toute la difficulté est de trouver un bon équilibre entre la qualité des informations disponibles et les coûts de maintenance. Des études expérimentales sont nécessaires pour bien calibrer ce compromis.

4. Le simulateur générique

Ce travail se situe dans le cadre du projet RARE (Routage optimisé par Apprentissage de REquêtes) qui fédère des équipes des trois écoles du GET. RARE s'intéresse à la recherche d'informations P2P et plus précisément à améliorer la propagation des requêtes en utilisant deux types de connaissances. D'une part une connaissance sur le contenu des pairs de manière analogue à PlanetP (voir [CHI 06] pour une présentation de nos premiers travaux sur ce sujet) et d'autre part une connaissance des requêtes passées : l'idée est de réutiliser les "bons" pairs mis en évidence dans la résolution d'une requête passée proche de la requête courante.

Pour tester nos idées, nous avons été amené à développer un simulateur. Parmi les outils de simulation existants, nous avons choisi PeerSim [Pee], qui est un outil *opensource* écrit en Java qui présente l'avantage d'être déjà spécialisé pour l'étude des systèmes P2P et qui dispose d'une architecture ouverte et modulaire qui permet de l'adapter et de le spécialiser. De plus, il est soutenu par une équipe de développement active, ce qui permet d'avoir à notre disposition nombre de modules de base déjà implémentés, notamment pour la construction d'une topologie de réseau physique réaliste.

Cette étude se décompose en cinq parties. Nous allons tout d'abord décrire comment nous construisons le jeu de données sur lequel portent les simulations. Ensuite, après avoir décrit le fonctionnement du simulateur PeerSim, nous verrons comment nous faisons le lien avec l'architecture décrite en section 2. Enfin, nous pourrons étudier la classe principale, le *GeneralP2PExchange*, et montrer ce que nous produisons en sortie du simulateur.

4.1. Les entrées du simulateur

Le simulateur a besoin qu'on lui fournisse un jeu de données. Ce jeu de données doit décrire les noeuds du système et les documents qu'ils possèdent, ainsi que les requêtes qui seront lancées sur le réseau.

A partir d'une analyse statistique réalisée sur des données collectées d'un système pair-à-pair Gnutella [GOH 05], nous avons enrichi le simulateur PeerSim d'une couche de génération de données dans le but de réaliser des simulations en conditions réelles. La figure 1 décrit le positionnement du générateur par rapport au simulateur. Le générateur prend en entrée deux fichiers décrivant le contenu de la collection

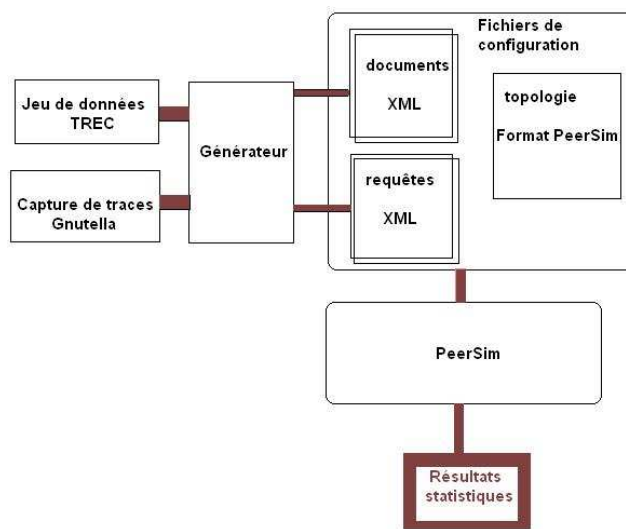


Figure 1. Couplage entre le générateur et le simulateur

utilisée (documents et requêtes) et produit deux fichiers XML correspondant aux associations des documents et des requêtes aux pairs. Un document est défini par un identifiant et une liste de mots clés. Une requête est définie par un identifiant, une liste de mots clés et une liste des documents pertinents avec pour chacun une mesure de similarité. Nous fournissons au générateur, en plus de la collection de données, des paramètres correspondant aux types de distributions des documents et des requêtes sur les pairs du réseau :

- Type de répartition (Uniforme ou Zipf) des occurrences de documents (resp. requêtes) sur les documents distincts (resp. requêtes distinctes) ;
 - Type de répartition (Uniforme, Zipf, Log-normale) des occurrences de documents sur les pairs ;
 - Type de répartition (Uniforme, Zipf) des occurrences des requêtes sur les pairs.
- Nous avons rajouté aussi le cas où nous voulons une répartition des requêtes proportionnelle à celle des documents : les pairs contenant un grand nombre de documents sont ceux qui envoient le plus de requêtes et vice versa.

En sortie, le générateur crée deux fichiers XML d'associations documents-pair et requêtes-pair qui constituent, avec les deux fichiers initiaux de définitions de la col-

lection et le fichier de topologie, les fichiers de configuration du simulateur PeerSim.

4.2. Le simulateur PeerSim

PeerSim est un environnement de simulation pour les réseaux pair-à-pair - non spécifique à une application - qui permet une simulation événementielle ou par cycles. Nos développements s'orientent vers la simulation par cycles ; nous expliquons ici brièvement son fonctionnement.

Une simulation est définie par un fichier de configuration. Le fichier de configuration définit quatre types d'éléments :

Les *Protocols*. C'est l'unité principale de PeerSim. Un *Protocol* est une fonctionnalité, associée à un noeud. On définit donc, dans le fichier de configuration, la pile de protocoles à mettre en place sur chaque noeud. À chaque cycle, le simulateur appelle sur chaque noeud, pour chaque *protocol*, dans l'ordre décrit dans le fichier de configuration, la fonction *nextCycle()*.

On peut également définir un *protocol* pour maintenir des attributs associés aux noeuds. Par exemple, PeerSim définit *IdleProtocol* ; un *protocol* qui maintient le voisinage d'un noeud, et dont la fonction *nextCycle()* est vide.

Les *Initializers*. Ce sont des modules lancés en début de simulation (1^{er} cycle) pour initialiser les *Protocols*. En pratique, il servent à définir la topologie initiale du réseau (associer à chaque noeud ses voisins), à distribuer le jeu de données (requêtes et documents)... Nous utilisons trois initializers, pour les cinq fichiers de configuration décrits à la section précédente.

Les *Dynamics*. Ils permettent d'appliquer un comportement dynamique au réseau. Il peut s'agir de supprimer des liens en cours de simulation, d'allumer ou éteindre des noeuds...

Les *Observers*. Ils servent à collecter les données nécessaires à la construction du résultat de la simulation. Un *Observer* défini dans le fichier de configuration est appelé à la fin de chaque cycle et effectue une observation sur le réseau entier.

Implémenter un système P2P dans PeerSim comporte donc trois étapes :

- 1) Définir les composantes de la couche protocolaire que l'on veut simuler, ce qu'il faudra initialiser sur ces composantes et ce que l'on voudra observer.
- 2) Construire le fichier de configuration correspondant,
- 3) Implémenter les protocoles dont on a besoin - s'ils ne sont pas déjà rendus disponibles par PeerSim.

4.3. Liens avec l'architecture générique

L'architecture générique décrite dans la section précédente est orientée vers la conception de nouveaux systèmes mais pas vraiment vers leur simulation. En effet,

lorsque l'on simule on va modéliser la partie du système que nous voulons observer et simplifier au maximum le reste. Lorsqu'on simule un système de recherche d'informations en P2P, la partie délicate à comprendre (et donc que l'on veut observer) concerne surtout l'aspect échange d'informations entre les pairs (propagation des requêtes, maintien de la cohérence des connaissances globales, ...). Nous avons donc fait le choix de concevoir le simulateur générique via le mécanisme d'échange. Cela nous a conduit à mettre en avant une notion de message, qui est la structure générique qui va pouvoir représenter toute l'information échangée (requête initiale, requête propagée, résultat, index, ...).

La première étape de spécialisation du simulateur PeerSim a donc consisté à ajouter un *protocol* définissant la gestion d'un message appelé *GeneralP2PExchange*. Au niveau de la configuration PeerSim, nous pouvons mettre en place plusieurs instances de *GeneralP2PExchange* sur chaque noeud (correspondant aux différents types de messages) et choisir, pour chacune, les classes qui vont implémenter ses interfaces.

Dans ce contexte, un mécanisme de recherche est défini par deux *GeneralP2PExchange*, prenant en charge respectivement les requêtes et les réponses. A cela viennent s'ajouter les instances *GeneralP2PExchange* prenant en charge la maintenance de la topologie du réseau et le partage d'informations (si besoin est). La figure 2 montre comment, à partir de l'implémentation des interfaces de ces *GeneralP2PExchange*, on peut se ramener à l'architecture décrite en section 2 ; pour des raisons de clarté, nous nous sommes limités aux interfaces concernées par le mécanisme de recherche.

4.4. La classe *GeneralP2PExchange*

La figure 2 décrit de manière simplifiée le diagramme de classes du *GeneralP2PExchange*. La classe *GeneralP2PExchange* permet de mettre en place un protocole d'échange en P2P à partir de l'implémentation de cinq interfaces. Deux interfaces permettent de définir le contenu des requêtes : *ForwardCriteria* et *EvaluationCriteria* ; les trois autres définissent leur prise en charge sur le réseau : *LocalEvaluation*, *ForwardManager* et *Shipping* :

ForwardCriteria Elle regroupe l'ensemble des données utiles à la propagation de la requête. Un exemple de telles données est le *Time To Live* associé aux requêtes Gnutella. Ces informations seront utilisées par les modules *ForwardManager* et *Shipping*.

EvaluationCriteria L'implémentation de cette interface détermine le critère d'évaluation locale. Elle regroupe les données à partir desquelles la requête sera évaluée sur chaque noeud. Dans Gnutella, il s'agit des mots clés sur lesquels porte la requête. Ces données seront utilisées par le module *LocalEvaluation*.

LocalEvaluation Cette interface définit la méthode *void eval(Query query)*. Son implémentation détermine l'action à mener sur un noeud lorsque l'on reçoit une requête. Cette action est exclusivement **locale** et utilise le *EvaluationCriteria* de la requête.

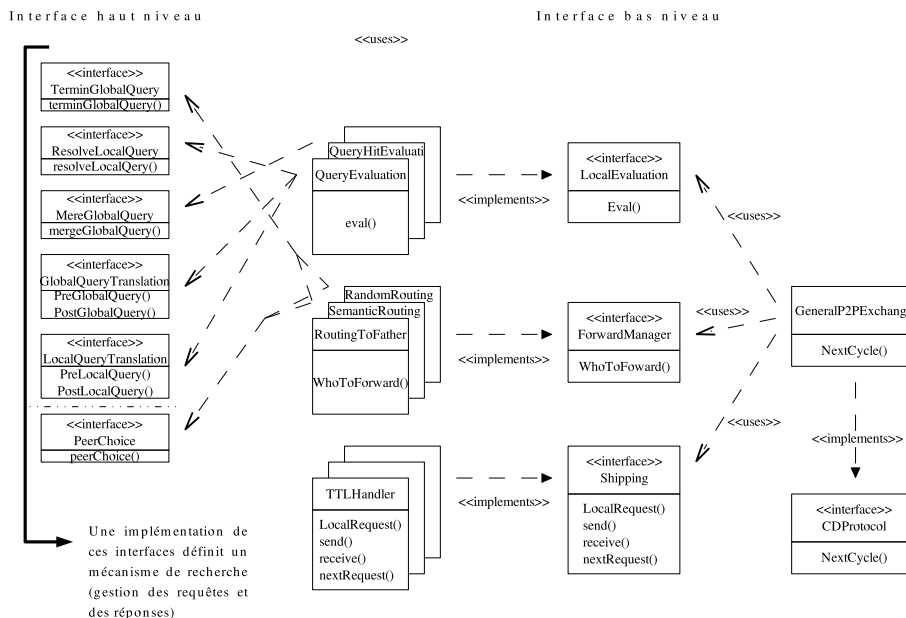


Figure 2. Diagramme de classes du *GeneralP2PExchange*

ForwardManager Le *ForwardManager* est en charge de mettre en place le routage des messages à travers le réseau pair-à-pair. Nous définissons cette fonctionnalité par la possibilité pour chaque noeud d'associer à chaque message reçu la liste des voisins à qui il sera transmis, par la méthode *LinkedList<Node> whoToForward(Query query)*.

Shipping Cette dernière interface prend en charge l'aspect réparti du protocole. Elle gère les tampons de requêtes et l'aspect non local de leur propagation, au travers de quatre fonctions :

- *void localRequest(Query query)* : Cette fonction permet l'ajout d'une nouvelle requête à émettre localement. Elle peut être utilisée par la fonction *eval* d'un autre *protocol GeneralP2PExchange* situé sur le même noeud - pour, par exemple, initier la réponse correspondant à l'évaluation d'une requête - ou par le protocole simulant un comportement utilisateur.

- *void receive(Query query)* : Il s'agit là aussi de recevoir une nouvelle requête, sauf que celle-ci a été reçue d'un autre noeud du réseau. Cette fonction est appelée par un *GeneralP2PExchange* distant. Elle met en place les mécanismes de gestion de la propagation. Un exemple typique est, dans *Gnutella*, de s'assurer que la requête n'ait pas déjà été reçue (détection des cycles). On situera également ici la mise en tampon des requêtes.

– *Query send(Query query)* : Au terme d'un cycle de traitement, on a donc reçu une requête, prise en charge par la méthode précédente. On a traité cette requête, et on a calculé vers quels voisins on devait la propager.

Avant d'envoyer effectivement la requête, pour chaque voisin, le *GeneralP2PExchange* appelle la fonction *send*. Cette fonction assure elle aussi une partie de la gestion de la propagation. Elle retourne une nouvelle requête, potentiellement la même, de façon à en permettre la modification.

Dans Gnutella, on assure ici l'incrémentation du TTL.

– *Query nextRequest()* : Cette fonction est utilisée par le *GeneralP2PExchange* pour appeler la prochaine requête à traiter. Classiquement, cette méthode extrait la prochaine requête du tampon.

4.5. Les sorties du simulateur

Nous obtenons en sortie un fichier XML décrivant l'évolution de la simulation, pour lequel nous avons construit nos propres *observers*. À partir de là, il est aisé d'intégrer les résultats dans un outil statistique.

Voici un extrait de fichier de sortie pour la simulation du système Gnutella :

```
<?xml version="1.0" encoding="UTF-8"?>
<queries xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <query identifi er="10.2.5" sender="10" init_cycle="5" >
    <words>
      <word>new</word>
      <word>world</word>
      <word>jasmine</word>
      <word>disney</word>
      <word>whole</word>
    </words>
    <neighbours>
      <node>11</node>
      <node>15</node>
      <node>9</node>
    </neighbours>
    <responses>
      <document name="1764" answering_node ="10" retrieval_time="8" class="none" distance="0.5" >
        <path>
          <node>10</node>
          <node>15</node>
          <node>10</node>
        </path>
      </document>
    </responses>
  <downloaded>
    <document name="1764" />
  </downloaded>
</query>
</queries>
```

Il consiste en une liste de requêtes - une seule est présentée. Il s'agit de la requête "10.2.5", lancée par le noeud 10, au cycle 5. On a ensuite la description du contenu de la requête (liste de mots clés). Le champ <neighbours> indique vers quels pairs la requête a été propagée.

La balise <responses> contient les réponses qui ont été reçues pour cette requête, avec le chemin parcouru pour l'obtenir.

Enfin, on a la liste des réponses téléchargées. Ici, une réponse pertinente a été définie dans le fichier de configuration comme ayant un champ "distance" inférieur ou égal à 5. La seule réponse obtenue a donc été jugée pertinente, elle a été téléchargée (et intégrée au contenu du noeud).

5. Conclusion

Nous avons présenté dans cet article une architecture générique pour la recherche d'informations qui nous permet de pouvoir mieux comprendre et comparer des systèmes de recherche d'informations P2P ou d'en concevoir de nouveaux. Cette architecture reprend en partie celle proposée par [ABE 04] en l'étendant et en faisant mieux apparaître les points cruciaux que sont à notre avis la définition de structures de données "globales" et la façon de les maintenir à jour. Cette architecture a été validée sur plusieurs systèmes existants.

Dans le contexte complètement décentralisée qu'est le P2P, il est souvent pertinent de simuler un système avant de l'implanter. A partir d'un simulateur existant PeerSim, nous avons développé une spécialisation qui doit permettre à un concepteur de systèmes de simuler plus rapidement et plus facilement ses nouveaux algorithmes. Ce simulateur est pour l'instant instancié sur Gnutella et PlanetP et doit être étendu pour offrir plus de fonctions de base (support de différents types d'index, intégration du feedback, ...). Il nous a permis de tester des systèmes comportant de l'ordre du millier de pairs.

A partir de ce travail, de nombreuses pistes de recherche s'ouvrent à nous. Nous allons travailler sur l'intégration des mécanismes d'apprentissage sur les requêtes passées développés par ailleurs dans le projet RARE. Il sera alors intéressant de comparer les informations obtenues par cette méthode par rapport à une approche basée sur le contenu des pairs. L'architecture générique et le simulateur supposent que tous les pairs jouent le même rôle. Il est intéressant de relâcher cette hypothèse pour permettre de décrire et de simuler des systèmes hiérarchiques mixant client/serveur et P2P. Enfin, le passage à l'échelle reste un problème puisque nous restons limité à des systèmes de l'ordre du millier de noeuds. Nous commençons à étudier une version parallèle de PeerSim qui devrait pouvoir nous faire gagner un ordre de grandeur.

6. Bibliographie

- [ABE 04] ABERER K., KLEMM F., RAJMAN M., WU J., « An Architecture for Peer-to-Peer Information Retrieval », *27th Annual International ACM SIGIR Conference (SIGIR 2004), Workshop on Peer-to-Peer Information Retrieval, Sheffield, UK, July 29, 2004.*
- [BAR 03] BARROSO L. A., DEAN J., HOLZLE U., « Web Search for a Planet : The Google Cluster Architecture », *IEEE Micro*, , 2003.

- [BLO 70] BLOOM H., « Space/Time Trade-offs in Hash Coding with Allowable Errors », *Commun. ACM*, vol. 13, n° 999, 7 Jul. 1970, p. 422-426.
- [CHI 06] CHIKY R., DEFUDE B., HÉBRAIL G., « Définition et diffusion de signatures sémantiques dans les systèmes pair-à-pair », *Extraction et Gestion des Connaissances*, , 2006.
- [CRE 02] CRESPO A., GARCIA-MOLINA H., « Routing Indices for Peer-to-Peer Systems », *Proc. ICDCS*, , 2002.
- [CUE 02] CUENCA-ACUNA F. C. PEERY P. M. E. D. T. N., « PlanetP : Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities », *IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, , 2002.
- [Gnu] « The Gnutella Protocol Specification v0.4 (Document Revision 1.2), accessible sur http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf, 2001 ».
- [GOH 05] GOH S., KALNIS P., BAKIRAS S., K. T., « Real Datasets for File-Sharing Peer-to-Peer Systems », *DASFAA*, , 2005.
- [NAK 03] NAKAUCHI K. E. A., « Peer-to-Peer Keyword search using Keyword Relationship », *Proc. Third International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems (GP2PC 2003)*, , 2003.
- [Pee] *Site officiel du simulateur PeerSim*, <http://peersim.sourceforge.net/>.
- [RAR] RARE, « Le projet RARE (Routage optimisé par Apprentissage de REquêtes), accessible sur <http://www-inf.int-evry.fr/defude/RARE>, accédé en décembre 2005 ».
- [RAT 01] RATSANAMY S., « A Scalable Content Addressable Network », *Proc. ACM SIGCOMM*, , 2001.
- [RIP 02] RIPEANU M., « P2P Architecture Case Study Gnutella Network », *Proceedings of the First International Conference on Peer-to-Peer Computing*, 999, 2002, 999, page 999.
- [RIS 04] RISSON J., MOORS T., « Survey of Research Towards Robust Peer-to-Peer Networks : Search Methods », *Technical report UNSW-EE-P2P-1-1*, , septembre 2004.
- [TAN 03] TANG C., XU Z., DWARDAKAS S., « P2P Information Retrieval Using Self-Organizing Semantic Overlay Networks », *Proc. ACM SIGCOMM*, , 2003.