Contributions à la modélisation, évaluation et conception de systèmes de recherche d'information en pair-à-pair

Jérôme Sicard

TELECOM SudParis

Université d'Évry Val d'Essonne

Thèse de doctorat de Télécom SudParis, préparée dans le cadre de l'école doctorale S&I, en accréditation conjointe avec l'Universite d'Evry-Val d'Essonne

Spécialité Informatique

Par :

Jérôme SICARD

Contributions à la Modélisation, Évaluation et Conception de Systèmes de Recherche d'Information en Pair-à-Pair

Thèse présentée pour l'obtention du grade de Docteur de TELECOM SudParis

Soutenue le 23 novembre 2010 devant le jury composé de :

Rapporteurs :		
Mme.	Claudia RONCANCIO	Professeur à l'Institut Polytechnique de Grenoble - LIG
M.	Pascal MOLLI	Professeur à l'Université de Nantes - LINA
Examinateurs :		
M.	Bruno DEFUDE	Directeur de Recherche à TÉLÉCOM SudParis - SAMOVAR (Directeur de thèse et encadrant)
M.	Jean-Marc DELOSME	Professeur à l'Université d'Évry - IBISC (Président du jury)
M.	Michel BEIGBEDER	Maître-assistant à l'ENS des Mines de Saint-Etienne - LSTI

Thèse numéro : 2010TELE0026

Remerciements

Je souhaite en premier lieu remercier les membres du jury, Claudia RONCANCIO, Pascal MOLLI, Jean-Marc DELOSME et Michel BEIGBEDER, pour avoir accepté d'examiner mon travail ;

Merci à Marie pour son amitié et son entêtement désormais légendaire ; qui a d'ailleurs fini par me mettre à la salsa ;

Merci à Wissam pour son soutien toujours renouvelé et son amitié indéfectible - et les couscous de Sana... ;

Merci à Javier et Christina pour m'avoir régulièrement supporté pendant leurs pauses et pour les surprises qu'ils cachent au fond de leurs tiroirs ;

Je voudrais aussi remercier Yassin, mais je crois surtout devoir remercier tous ceux qui ont eu à supporter nos débats ;

Merci à Mohamed qui savait nous faire taire ;

Merci à Nomane le sage, notre ancien ;

Merci à Amine, Ben, Dorsaf, Hamid, Imen, Ines, Jallel, Kien, Ramzi, Sondes, Tony et tous ceux qui m'ont accompagné durant cette thèse, dans mes fonctions de président de Doc'INT et de représentant des doctorants ;

Merci à Anis, Aymen, Carlos, David, Mahmoud, Petros, Providence et vive le foot !

Merci à Abdul-Malik pour sa droiture ;

Merci à Léon pour sa générosité, ses cerises, ses barbecues...

Merci à Salah-Salim pour ses conseils en repassage ;

Merci à Zied pour sa gentillesse ;

Merci à Rami pour m'avoir consacré une après-midi mémorable, alors qu'il était en congés, bloqués dans les bouchons parisiens, pour trouver un canapé qui n'existait pas et courir après des locataires fantômes ;

Merci à Abdeltif pour m'avoir offert quelques discussions scientifiques vraiment intéressantes et fructueuses, trop rares durant cette thèse, donc précieuses ;

Merci aux permanents de l'équipe - Alda, Amel, Claire, Samir et Walid - pour leur soutien, leur écoute et leurs conseils ;

Merci à Brigitte pour sa bienveillance ;

Merci aux stagiaires qui ont travaillé avec moi et avec qui j'ai beaucoup appris : Bob et Zu, Muath, Abdessamad, Ines, en espérant que notre collaboration fut fuctueuse pour eux aussi ;

Merci à mes parents, Philippe et Béatrice, parce que c'est quand même bien un peu grâce à eux que j'en suis arrivé là ;

Un petit mot pour ma famille, Flo et Amandine, les cousins, les familles Cardeau, Arcos et leurs pièces rapportées ;

Je remercie la médiathèque parce que ça fera plaisir à Maëlle ;

Merci Maëlle pour le bonheur que tu m'apportes.

Résumé

Cette thèse se base sur une vision originale des systèmes de Recherche d'Information en pair-à-pair (RI-P2P) ; pour proposer un ensemble d'outils pour mieux les évaluer et mieux les concevoir.
Synthétiquement, la problématique d'évaluation consiste à comprendre quel système fonctionne bien dans quelles conditions ; comment les différents composants/aspects d'un système contribuent à ses performances ; comment des modifications du système peuvent faire évoluer ses performances. D'un point de vue conception, on voudrait savoir choisir une approche P2P en fonction du contexte de l'application à implémenter (jeu de données, propriétés du réseau...) ; comprendre comment mixer des systèmes ; mieux maîtriser l'impact d'une modification de l'implémentation sur le comportement et les performances d'un système.
Nous pensons que ces deux problématiques d'évaluation et de conception sont fortements liées. Un même fondement est nécessaire pour progresser dans ces deux sens : mieux comprendre l'architecture des systèmes RI-P2P. À la base de notre réflexion, nous considérons qu'un système RI-P2P est l'implémentation en P2P d'un modèle RI. Nous nous appuyons sur cette idée pour donner un ensemble de définitions qui permettent de mieux appréhender les différents composants RI et composants P2P d'un même système et la façon dont ils s'organisent. Nous montrons ensuite comment on peut utiliser cette vision originale du domaine pour spécifier la démarche d'évaluation d'un système P2P et pour aider à la conception de nouveaux systèmes.

Mots clefs : Pair-à-Pair, Recherche d'Information, Indexation.

Abstract

Contributions for Modeling, Evaluating and Designing Peer-to-Peer Information Retrieval Systems

This PhD thesis develops an original vision of Peer-to-Peer Information Retrieval Systems (P2P-IR). Based on this vision, we propose a set of tools to better evaluate and design them.
Synthetically, on the one hand, evaluating means understanding which system behaves well in which context ; how different components/aspects of a single system contribute to its performance ; how modifications of the system can make its performance evolve. On the other hand, considering design of P2P-IR systems, we would like to be able to choose a P2P approach based on what is known about the context the application will execute in (data set, network properties...) ; understand how to mix P2P systems ; better control the impact of a modification of the implementation of a system on its behavior and performance.
We believe those concerns of evaluation and design are strongly related. There is a common need to improve on both directions : better understand the architecture of P2P-IR systems. On the basis of our reflexion, we state that a P2P-IR system is the implementation in P2P of an IR system, which contrasts with the traditional two-layered vision. We use this idea to give a set of definitions that help better understand the different IR and P2P components of a single system and their relationships. We then show how this vision can be used to help design new systems.

Keywords : Peer-to-Peer, Information Retrieval, Indexation.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

(Ratajczak and Hellerstein, 2003) débute avec un petit paragraphe au sujet des DHT (Distributed hash Tables, tables de hachage distribuées) qui nous semble résumer parfaitement l'esprit de cette thèse. C'est pourquoi nous vous en proposons, pour démarrer ce volume, une retranscription suivie d'une traduction rapportée à notre objet d'études, les systèmes pair-à-pair (systèmes P2P) :

There is a slew of DHT designs and implementations, each claiming relavance on theoretical or practical grounds. But while the area of DHT design has been the focus of considerable research, there remains no consensus on the question, "What is a good DHT ?". In this paper, we survey the field and address the more basic question, "What is a DHT ?", with the hope that a better understanding of the anatomy of DHT designs will engender a more systematic comparison between them, as well as a more principled approach to future designs.

Il existe toute une série de modèles et implémentations de systèmes P2P, chacun revendiquant sa pertinence sur des bases théoriques ou pratiques. Mais alors que la conception de systèmes P2P a été le sujet d'un effort de recherche considérable, on demeure sans consensus sur la question : « Qu'est-ce qu'on bon système P2P ? ». Dans cette thèse, nous faisons un point sur ce champ d'études et nous traitons la question plus fondamentale « Qu'est-ce qu'un système P2P ? », avec l'espoir qu'une meilleure compréhension de l'anatomie des modèles P2P en permettra une comparaison plus systématique, ainsi qu'une approche plus raisonnée des futurs modèles.

Nous cherchons donc à mieux comprendre les technologies P2P, pour mieux les évaluer, les comparer et les concevoir.

1.1 Contexte et motivations

1.1.1 Qu'est-ce qu'un système P2P ?

D'après les recherches que nous avons menées, il semble que la définition du terme « système P2P » ne fasse pas consensus. Plutôt que d'en donner une définition formelle, nous préférons donc donner une caractérisation du domaine, construite à partir de notre observation de systèmes se disant P2P. Ensuite, nous verrons comment quelques exemples de systèmes (P2P ou non) se situent par rapport à ces caractéristiques. Enfin, nous ciblerons le sous-ensemble de systèmes qui nous intéresse dans le cadre de cette thèse.

Les participants à un réseau P2P sont généralement appelés *pairs* ou *servants*. La définition d'un système P2P peut, selon nous, s'aborder sous trois angles. La difficulté est que, quel que soit l'angle considéré, on trouve des systèmes dits « P2P » qui prennent bien en compte cet aspect mais négligent les deux autres. On peut donc définir les systèmes P2P en considérant trois aspects :

- **technologique**. Les systèmes P2P sont fortement distribués. C'est-à-dire que les pairs tendent à avoir tous le même comportement ; en particulier, aucun participant ne joue le rôle de serveur unique.
- **contraintes d'exécution**. Les systèmes P2P sont conçus pour fonctionner dans un environnement de type Internet, usuellement caractérisé comme dynamique et grande échelle. « Environnements dynamiques et grande échelle » signifie que les pairs entrent et quittent fréquement le système et qu'ils sont très nombreux ; par exemple, on considère souvent qu'il n'est pas possible de mettre-en-place une procédure de déconnexion.
- **applicatif**. À travers un système P2P, les utilisateurs partagent des ressources - des index de ressources documentaires dans le cas de la Recherche d'Information, de la puissance CPU dans le cas de Seti@home, des fichiers pour Bittorrent et KaZaA... Un principe veut que tout utilisateur soit aussi contributeur. C'est-à-dire que celui qui utilise les ressources offertes par les autres est aussi celui qui fourni des ressources à la communauté.

Il nous semble que l'usage veuille qu'un système soit P2P lorsqu'il remplit au moins deux de ces critères. On remarque aussi que dans le langage courant, le P2P est assimilé partage de fichiers, alors qu'on peut faire d'autres choses en P2P (voir Seti@home) et qu'il y a d'autres moyens de partager des fichiers (FTP...). Enfin, la communauté recherche a plus tendance à favoriser l'aspect technologique pour cataloguer un système comme P2P, tandis que la communauté des développeurs favorise plus l'aspect applicatif.

Avant de donner quelques exemples, on peut remarquer que ces trois aspects sont très liés. Par exemple, les contraintes du réseau Internet, son échelle notamment, rendent coûteuse une implémentation centralisée. De même, la grande majorité des systèmes de partage de fichiers en P2P fonctionne sur le principe que les ressources sont fortement répliquées. Cela signifie qu'une même ressource est présente sur un grand nombre de pairs. En sélectionnant aléatoirement un nombre suffisamment grand de pairs, on a donc une forte probabilité de trouver la ressource recherchée. Le fait qu'une ressource soit fortement répliquée est la conséquence d'un comportement des utilisateurs, et est donc directement lié à la caractéristique que tout utilisateur est aussi contributeur. Enfin, on peut remarquer que le troisième aspect traîte des ressources directement partagées entre les utilisateurs, tandis que le premier traîte des ressources systèmes nécessaires au partage. Mais le même esprit domine : les pairs doivent contribuer de façon équitable à ces deux types de ressources.

Le terme « équité » est ici à prendre avec des princettes. Nous avons remarqué deux façons de formuler l'objectif d'égalité entre les nœuds : soit on vise à ce qu'ils participent tous à la même hauteur, soit on vise à ce qu'ils participent proportionnellement à leurs capacités. De même, on trouve des systèmes hybrides centralisé/distribué ; et la frontière entre un système dynamique ou non, grande échelle ou non, est floue.

Il nous semble donc raisonnable de parler en termes d'objectifs : *un système P2P est un système qui tend à être fortement distribué, dynamique et grande échelle, dans lequel les utilisateurs sont aussi les contributeurs.*

Nous pouvons maintenant situer quelques exemples de systèmes par rapport aux trois aspects introduits dans notre définition.

BitTorrent est un protocole de téléchargement multisource, par morceaux, en parallèle. Chaque pair qui télécharge devient à son tour source (pair serveur) dès qu'il a récupéré au moins un morceau du fichier. Chaque pair est adressé par son adresse IP, on n'a donc pas ici de procédure de désanonymisation. En parallèle, on a un système de récompense pour inciter les utilisateurs à partager des ressources (ressources informatiques) : plus on contribue en tant que source, plus on peut utiliser les autres comme sources.

Pour pouvoir télécharger un fichier par BitTorrent, il faut deux choses :
- un descripteur, qui permet d'identifier le fichier de façon unique et contient les métadonnées, dont celles utiles au téléchargement - nombre de morceaux, somme de contrôle...
- une liste de sources à partir desquelles télécharger le fichier.

Les métadonnées se trouvent dans un fichier *.torrent*. Elles sont générées par le client qui publie le fichier. Les sources sont ajoutées et mises à jour par un *tracker*, donc un serveur centralisé. L'adresse du tracker est donnée dans le .torrent. Certains clients permettent de rechercher des fichiers .torrent de façon décentralisée, mais il n'existe par à notre connaissance de système décentralisé de mise à jour sources.

Il faut donc bien distinguer trois aspects dans bitTorrent :
- **le fichier** en tant que tel est partagé en pair-à-pair, dans tous les sens - technologique, contraintes d'exécution et applicatif.
- **les métadonnées** sont créées de façon centralisée par le pair publiant la ressource. À l'origine, la maintenance est elle aussi centralisée ; elles sont maintenues en P2P dans les clients les plus récents (qui sont aussi probablement les plus utilisés).
- **la liste des sources** est à notre connaissance toujours maintenue de façon centralisée.

Napster était un serveur d'indexation de fichiers partagés. Il maintenait les métadonnées associées aux fichiers et la liste des sources. Les pairs pouvaient ensuite s'échanger les fichiers directement. Il jouait donc le rôle des *tracker* BitTorrent. Dans Napster, l'indexation était centralisée, tandis que l'accès aux ressources était P2P. Nous parlerons dans cette thèse de service d'indexation et service d'accès. Il n'est donc pas tout à fait juste de dire que Napster est un système P2P ; il serait plus juste de dire que Napster utilise un service d'accès P2P.

Lorsqu'on recherche une page Web avec **Google**, on n'utilise pas service d'indexation P2P. En effet, la technologie est centralisée ; on ne peut pas faire le lien entre utilisateur et contributeur, la publication d'une page web ne se faisant pas via Google ; par contre, on est dans un environnement grande échelle qu'on peut considérer comme dynamique.

Pour accéder ensuite aux données (la page Web sur laquelle on a cliqué), on utilise le service **DNS**. Même si la technologie utilisée est fortement distribuée, l'ensemble des machines qui rendent le service est nettement disjoint de l'ensemble des machines qui l'utilisent - et pour cause, il s'agit de serveurs. Il y a donc une nette différenciation des rôles des pairs, tant au niveau technologique qu'applicatif. Le DNS n'est donc pas un système P2P selon notre définition.

Les systèmes DHT que nous décrirons dans la section 2.2.3 de notre état de l'art sont souvent décriés pour mois bien supporter la dynamicité et le passage à l'échelle que d'autres types de systèmes P2P. Quelle que soit l'issue de la polémique à ce sujet, ils mettent bien en place une technologie fortement distribuée égalitaire entre les pairs, et l'ensemble des pairs posant des requêtes est bien confondu avec l'ensemble des pairs y répondant. Ce sont donc bien des systèmes P2P au sens de notre définition.

Il est à noter que beaucoup de systèmes P2P rendent simultanément les services d'indexation et d'accès. Par exemple, avec Gnutella, en même temps que la réponse à la recherche de fichier (description du fichier, métadonées), le système retourne toutes les informations permettant de récupérer chaque fichier réponse.

Dans la suite de cette thèse, nous nous intéressons aux
Services d'Indexation en P2P.

1.1.2 La Recherche d'Information

Dans cette thèse, nous nous intéressons aux services d'indexation en P2P. Pour cela, nous avons choisi une application cible : la Recherche d'Information (RI). Nous étudions donc les systèmes de Recherche d'Information en P2P (RI-P2P) parce que la RI est un domaine d'application complexe, qui nous permet de pousser les systèmes P2P dans leurs retranchements. La complexité des systèmes de Recherche d'Information nous laisse penser que ce qu'on peut apprendre de la conception et de l'évaluation des systèmes RI-P2P sera réutilisable dans un grand nombre d'autres domaines applicatifs.

Définition 1.4 : Recherche d'Information La Recherche d'Information est une branche de l'Informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche et la sélection d'information. (Salton, 1968)

La Recherche d'Information est un domaine scientifique désormais ancien, qui a atteint une certaine maturité. On ne présente plus *Google* (GoogleWebsite,), un système RI sur Internet. Il en est sans doute le représentant le plus célèbre ; c'est une société cotée en bourse et répandue dans le monde entier. Il illustre bien la tendance dominante qui est de concevoir des systèmes centralisés.

1.1.3 Pourquoi concevoir des systèmes P2P ?

La technologie P2P présente un certain nombre d'atouts. Du point de vue du chercheur en systèmes répartis, on a un grand nombre de nœuds disponibles ; on a donc potentiellement des capacités accrues en termes de puissance de calcul et de capacité de stockage. Mais le passage à la technologie P2P propose aussi une nouvelle donne au chercheur en Recherche d'Information.

La Recherche d'Information centralisée a tendance à favoriser les informations sur le corpus global, en minimisant les informations personnalisées pour chaque utilisateur. En effet, dans un système centralisé, les informations globales sont plus facilement disponibles - et moins coûteuses à maintenir (place mémoire) que les informations individualisées. En P2P, les contraintes s'inversent. Cette fois, il est possible d'aller très loin dans la personnalisation des informations ; ce sont les informations globales qui sont difficiles à obtenir et maintenir.

Plus pragmatiquement, lorsqu'on fait de la RI en P2P (RI-P2P) **tous** les objets manipulés sont :

Répartis parce qu'on est en pair-à-pair, ce qui impose de prendre en compte de nouvelles contraintes pour maintenir la cohérence globale des informations.

Clustérisés parce que regroupés sur différents pairs.

Désanomymisés parce que derrière un pair, on retrouve un utilisateur, ou du moins une unité d'utilisateurs (ex. habitants d'un même domicile).

En P2P, on a donc a priori connaissance d'une relation *(utilisateur, ressources, requêtes)*. La partager ou l'utiliser peut être coûteux, mais contrairement aux conditions centralisées la collecte en est triviale. Certains systèmes de notre état de l'art ignorent cette information, d'autres l'utilisent intensivement.

1.2 Problématique

1.2.1 Comment la problématique se pose au monde de la recherche

Avec l'émergence massive des systèmes de partage de fichiers en pair-à-pair (P2P) dans les années 2000, la communauté scientifique s'est beaucoup intéressée aux technologies P2P. La littérature scientifique présente aujourd'hui un grand nombre de propositions et études de systèmes. Derrière ces systèmes, ce sont un grand nombre de techniques différentes qui sont aujourd'hui disponibles.

Pourtant, un nombre très restreint sont effectivement utilisées. La majorité des systèmes P2P aujourd'hui déployés font du partage de fichiers en P2P : ce sont KaZaA, Bit-Torrent, e-Mule... À de rares exceptions près, tous utilisent une technologie P2P hiérarchique (cf. section 2.2.2). On trouve aussi des systèmes de stockage, partage d'espace disque voir système de fichier distribué en P2P ; cette fois plutôt basés sur des technologies DHT (Distributed Hash Tables, tables de hachage distribués), connues pour permettre de stocker efficacement des données de façon fortement distribuée. Nous verrons que l'état de l'art propose bien d'autres types de techniques P2P. Enfin, on trouve des applications de type réseaux sociaux ; dans ce cas, l'appui technologique P2P est plus faible, puisque ce sont les utilisateurs qui gèrent les connexions entre eux, et que l'essentiel des applications effectue uniquement des échanges entre un pair et ses relations. Ce dernier type d'application reste plutôt confidentiel (en P2P).

C'est justement un problème récurrent des technologies P2P : elles sortent difficilement du cadre de l'application de partage de fichiers. Hors de ce domaine, les logiciels P2P apparaissent systématiquement très confidentiels par rapport à leurs homologues centralisés. Il nous semble que la cause est toujours semblable : les logiciels basés sur des technologies P2P présentent toujours des lacunes de fonctionnalités par rapport aux systèmes centralisés. Nous l'expliquons par ce qu'on manque encore d'outils de conception des systèmes P2P :

- Visiblement, les techniques développées dans les laboratoires de recherche trouvent difficilement écho dans l'industrie, probablement parce qu'elles sont mal connues : on a des difficultés à comparer précisément les nouvelles technologies entre elles, donc on a du mal à « vendre » ces nouvelles techniques.

- On sait mal décrire les possibilités offertes par les techniques P2P. On n'a pas aujourd'hui de réponse claire à des questions telles que : "Quelles sont les limites des systèmes P2P ? Quand peut-on utiliser une technologie P2P, quand doit-on programmer en centralisé ?
- On n'a pas bien formalisé, on a peu d'outils pour construire un système P2P, implémenter une fonctionnalité en P2P. On observe d'ailleurs souvent des confusions entre application et technologie. Par exemple, Bit-Torrent fait du partage de fichiers - c'est l'application - en P2P hiérarchique, la résolution inter-clusters fonctionnant sur un mode structuré, la résolution intra-clusters fonctionnant sur un mode centralisé - c'est la technologie utilisée.

Nous pensons que cela traduit que ces problématiques et les réponses qui y sont apportées ne sont pas bien formalisées par la communauté recherche P2P.

Pour fonder ce propos sur un exemple, nous pensons que l'engouement de la communauté scientifique pour les systèmes P2P, comme l'intérêt de la communauté civile, qui a financé ces recherches, ont une cause très pragmatique : les systèmes de partage de fichiers en P2P ont montré qu'on peut trouver des applications pour lesquelles l'utilisation de technologies P2P est justifiée techniquement - bonnes performances - et présente des avantages par rapport à un système centralisé - moins cher à déployer, anonyme... Le partage de fichiers audios et vidéos en P2P fonctionne notamment parce que dans ces réseaux les données partagées sont fortement répliquées. Donc une recherche aléatoire fonctionne bien. Parallèlement, il est possible de compenser la faible bande-passante ascendante des pairs par des téléchargements en parallèle. Ce type de jeu de données et ce contexte d'utilisation du réseau sont une spécificité apportée par l'application de partage de fichiers.

Nous voyons un fort enjeu à pousser plus avant cette logique : comprendre pour quel type d'application et quel contexte d'utilisation les technologies développées dans les laboratoires de recherche sont efficaces ; mieux cerner les types de fonctionnalités qui peuvent être efficacement implémentées en P2P ; pouvoir décider à quel moment il est préférable d'utiliser une technologie P2P ou une technologie centralisée.

Globalement, on a besoin d'outils pour mieux comparer les systèmes P2P. Comparer deux systèmes revêt deux aspects : la comparaison technique (comment les systèmes sont faits) et la comparaison de performances. Dans la littérature, les techniques P2P sont utilisées pour implémenter différents types d'application. On a donc besoin d'une méthode pour séparer l'application des techniques utilisées pour l'implémenter.

1.2.2 Comment nous en sommes venus à étudier cette problématique

L'objectif initial de cette thèse était d'étudier les possibilités offertes par l'utilisation d'une information de type *feedback* (retour de l'utilisateur sur la qualité des réponses apportées par le système) pour améliorer le routage dans les systèmes P2P. L'idée qui

s'est rapidement imposée à nous, c'est que la question ainsi formulée est mal posée. En effet, on ne peut pas poser ainsi un lien direct entre le feedback et le routage. Échanger des informations de feedback permet d'améliorer la qualité des informations utilisées par le système : index de pair, index de requêtes, index de ressources. L'utilité d'utiliser une information de feedback, son impact sur la qualité des index et donc sur la qualité du résultat produit par le système est un problème purement Recherche d'Information. Parallèlement, dans un cadre P2P, il va falloir mettre en place des mécanismes permettant de transporter l'information de feedback depuis le lieu où elle est produite vers le lieu où elle est utile. Mais l'utilisation des index pour router les requêtes reste inchangée.

Cette réflexion nous amène à l'intuition qu'il existe une dichotomie entre deux problématiques différentes dans les systèmes P2P. On a d'un coté une problématique applicative, qui produit et utilise des informations dans l'objectif de remplir une (des) fonctionnalité(s). De l'autre, des technologies P2P sont sollicitées pour implémenter cette application en P2P ; et elles sont largement indépendantes des problématiques fonctionnelles.

Étudier l'utilisation du feedback en P2P, serait donc d'abord comprendre comment cette information est créée et utilisée dans un système RI classique, puis comprendre les enjeux de l'implémentation en P2P d'un modèle de Recherche d'Information et comment ce type d'implémentation impacte les performances.

1.3 Objectifs

C'est donc cette problématique de l'implémentation en P2P d'un modèle RI - et même, dans une certaine mesure, d'une application en général - que nous essayons d'expliciter dans cette thèse. Le but en est de fournir des outils pour aider à concevoir et mieux évaluer les systèmes RI-P2P. Nous commençons par nous donner une idée de la problématique RI, de l'étendue des techniques et de ses enjeux. Nous faisons ensuite une étude bibliographique des systèmes P2P en général et des systèmes RI-P2P en particulier. Puis nous finissons notre état de l'art par une étude des propositions et pistes pour un méta-modèle des systèmes P2P. La confrontation entre ces méta-modèles et les systèmes étudiés précédemment nous permettent de mettre en évidence des pistes prometteuses, sans pour autant qu'aucun des méta-modèles proposés ne soit réellement satisfaisant.

Comme nous l'avons déjà évoqué, il nous semble que notre problématique revêt deux aspects fortement interdépendants : l'évaluation et la conception de systèmes P2P. En effet, une évaluation bien menée doit servir de guide à la conception. Elle permet de comprendre quelle technique ou combinaison de techniques donne les meilleures performances, étant donnés une application, un contexte d'utilisation de cette application (comportement utilisateur, propriétés du réseau...) et le type de jeu de données

qui sera manipulé par elle. Elle permet aussi de comprendre, suivant les performances observées sur un système déjà déployé, dans quelle direction orienter les efforts de reconception pour espérer améliorer ses performances. Réciproquement, sur un système bien « conçu », dans lequel les différentes fonctionnalités/composantes et leurs interactions sont clairement identifiées ; il sera plus facile de mener une étude de performances précise - identifie bien les limites du système - complète - tous les paramètres influant les performances sont pris en compte - et modulaire - un maximum de résultats sur des composantes du système sont réutilisables.

Le but à terme est de fournir une trousse à outils au concepteur, un atelier de génie logiciel qui s'inscrive dans la logique suivante :

- Quelles sont les techniques à disposition dans la littérature pour faire ce travail ? Nous voulons donc être capables de lister les techniques P2P décrites par notre état de l'art. Ce n'est pas un problème trivial, dans la mesure où un système P2P utilise généralement une combinaison de techniques P2P.
- Comment adapte-t-on ces techniques ? Que signifie d'utiliser ces techniques pour construire l'application ? Comment combine-t-on des techniques P2P ?
  Cette question se résoud en comprenant comment faire la comparaison technique précise de deux techniques P2P. Nous donnerons des éléments d'architecture des systèmes P2P.
- D'après les évaluations menées en amont, quelle technique donnera les meilleures performances ?
  C'est ici qu'arrive le problème de l'évaluation de performances. Une évaluation des systèmes P2P basée sur l'évaluation des techniques qui ont été combinées pour concevoir ces systèmes.

1.4 Plan du document

Nous avons pris le parti d'aborder le problème par l'aspect évaluation. Le chapitre 3 fait une analyse de l'évaluation de deux systèmes très différents : PlanetP (Cuenca-Acuna et al., 2003) et pSearch (Tang et al., 2003). Il nous permet de mettre en évidence les manques de ces évaluations, symptomatiques de la problématique d'évaluation des systèmes P2P en général. On ne peut pas comparer les performances des deux systèmes. Les auteurs ne donnent pas les mêmes courbes en sortie. Ils n'étudient pas l'impact sur les performances des mêmes paramètres. Seul le mécanisme de recherche est étudié, en négligeant notamment les coûts et la qualité de la topologie nécessaire à son fonctionnement. Ceci rend impossible sa comparaison à PlanetP. Enfin, les performances de pSearch sont comparées à celles d'un système centralisé, tandis que PlanetP est évalué dans une démarche plus classique, « dans l'absolu », par rapport aux résultats fournis par un oracle. Même si l'on regarde système par système, l'influence de certains

paramètres est ignorée, et on n'arrive pas à cerner les conditions limites, qui conservent aux systèmes des performances acceptables.

Nous pensons que ces manques sont justifiés parce qu'il manque aux évaluateurs le soutien d'une réflexion de fond sur l'articulation entre une application et son implémentation en P2P, un méta-modèle des systèmes P2P. Nous posons des éléments de réflexion dans ce sens dans le chapitre 4. Cette réflexion s'articule autour d'un ensemble de définitions, qui permettent de fixer un ensemble de concepts structurants sur le domaine.

Dans le chapitre 5, nous montrons plus précisément comment on peut mener une démarche analytique préalable à l'évaluation. Cette analyse permet une approche plus raisonnée de l'évaluation d'un système, permettant de déjouer les pièges identifiés dans le chapitre 3. Nous montrons sur PlanetP et pSearch comment on peut utiliser les concepts du chapitre 4 dans une sorte de démarche de rétro-conception. On isole ainsi les différentes composantes du système : les étapes du modèle de Recherche d'Information et les mécanismes répartis constituant son implémentation. Nous montrons ensuite comment on peut analyser les inter-dépendances entre ces composantes, pour définir un diagramme d'évaluation ; le diagramme d'évaluation spécifie la démarche d'évaluation d'un système P2P. Les composantes du système sont évaluées indépendamment ; l'étude de leurs inter-dépendances permet de définir de manière précise les paramètres à prendre en compte dans l'évaluation et de recomposer les performances du système global à partir des performances des composantes.

L'analyse menée dans un but d'évaluation nous permet de décomposer les systèmes. Notamment sur les aspects P2P, nous extrayons des solutions P2P, implémentées par des mécanismes P2P (pour les définitions, cf. section 4). Dans le chapitre 6, en nous replaçant à l'échelle de l'ensemble des systèmes de notre état de l'art, nous utilisons notre démarche analytique pour proposer une architecture des systèmes P2P, de forme arborescente, capable d'accueillir l'ensemble des composantes apparaissant dans l'ensemble des systèmes. C'est à dire qu'en décomposant tous les systèmes en suivant la même démarche analytique, des similitudes se font jour qui nous permettent d'abstraire une architecture des systèmes P2P.

L'élément clef de cette architecture est le mécanisme P2P. Nous définissons chaque mécanisme présent dans les systèmes de l'état de l'art, comme la résultante d'une succession de choix de conception. Nous présentons un arbre des choix de conception dont les feuilles sont l'ensemble des mécanismes P2P qu'on peut imaginer à partir de notre état de l'art. Dans la mesure où l'on sait comment ces mécanismes s'assemblent dans notre architecture des systèmes P2P, on a là un recensement exhaustif des technologies P2P proposées dans notre état de l'art, qui les compare techniquement et montre comment elles peuvent être hybridées entre elles. En guise de perspective, nous savons évaluer les composantes des systèmes P2P et nous savons comment ces composantes s'assemblent en un système complet ; nous avons donc tous les outils pour construire un

outil puissant de conception, guidé par les propriétés du jeu de données et autres éléments de contexte d'utilisation d'une application, permettant de choisir la technologie P2P la mieux adaptée à son implémentation.

Chapitre 2

État de l'Art

Dans ce chapitre, nous introduisons les problématiques de la Recherche d'Information (RI) et des systèmes pair-à-pair (P2P). Dans une première section, nous donnons les principes de fonctionnement d'un système de Recherche d'Information, et nous présentons les modèles RI de référence. Dans une seconde section, nous présentons les systèmes pair-à-pair, selon le modèle qu'on peut maintenant qualifier de classique : systèmes pair-à-pair « purs », hiérarchiques et structurés. Une fois ces bases posées, nous pouvons passer à l'état de l'art des systèmes de Recherche d'Information en pair-à-pair (RI-P2P). Nous avons choisi un échantillon de six systèmes/techniques qui permettent de bien cerner les différentes approches envisagées par la communauté. Enfin, dans la dernière section nous introduisons notre problématique, en présentant plusieurs propositions ou pistes d'abstraction des systèmes RI-P2P. Nous verrons ainsi que les outils développés par la communauté sont encore insuffisants pour apporter un réel soutien au concepteur où à l'évaluateur de systèmes RI-P2P en particulier et P2P en général.

2.1 Quelques bases en Recherche d'Information

2.1.1 Le processus de Recherche d'Information

La Recherche d'Information consiste à rechercher - dans notre contexte, automatiquement - dans un corpus documentaire les documents qui contiennent la réponse à un besoin utilisateur. La figure 2.1 ci-dessous présente les principales étapes du processus de Recherche d'Information.

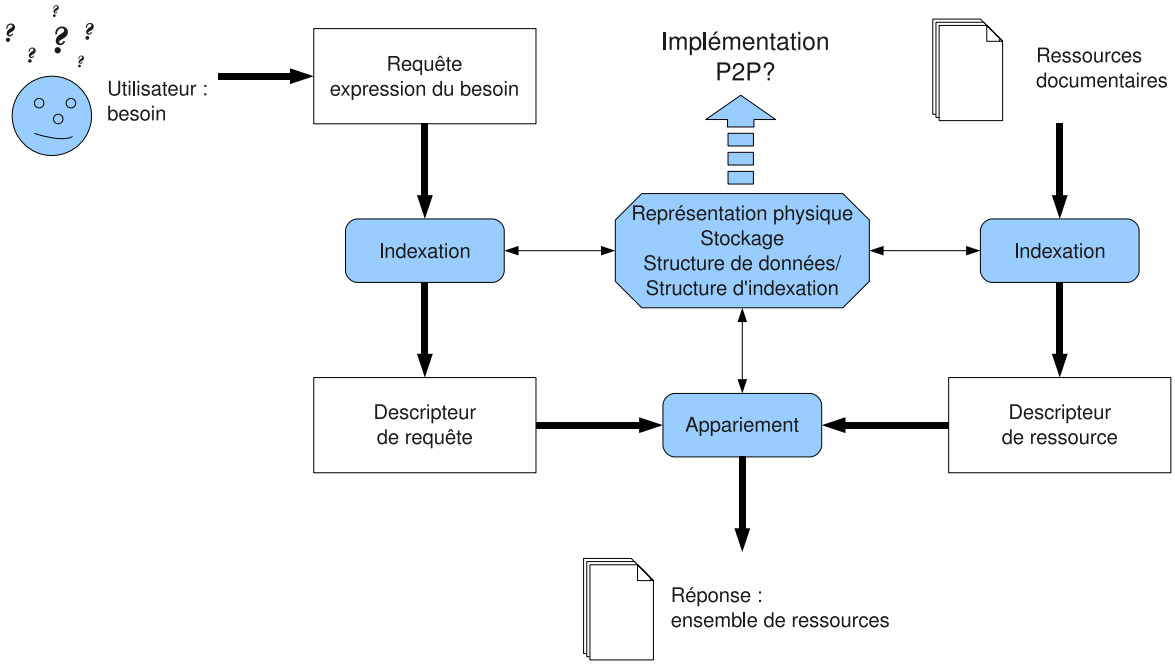


FIG. 2.1 – Le processus de Recherche d'Information.

L'utilisateur décrit son besoin au travers d'une requête. Cette requête peut avoir une syntaxe : le langage de requête. Il en existe de très nombreux ; dans ce travail, nous nous intéressons uniquement aux requêtes par mots clefs. La requête est simplement un ensemble non-ordonné de mots.

L'objectif est de comparer la requête aux documents du corpus : c'est la fonction d'appariement. Le plus souvent, on utilise une fonction calculant la distance d'une requête à une ressource. Le résultat de cette comparaison peut prendre typiquement deux formes : soit on retourne l'ensemble des ressources qui répondent - la notion de pertinence est binaire - soit on trie les ressources du corpus par ordre de pertinence à la requête - pertinence continue. La plupart du temps, on utilise une solution mixte : on retourne un sous-ensemble du corpus, trié par ordre de pertinence.

Pour pouvoir comparer une requête et un document, il faut extraire du document les concepts dont il traite, et de la requête les concepts qu'elle désigne. Tout cela doit être décrit dans un langage adapté au traitement par informatique : c'est l'indexation. En général (mais pas toujours), requête et documents seront décrits dans le même langage.

Au cœur de ce processus, on trouve les problématiques de représentation physique, stockage et indexation (au sens structure de données) des index de ressources et de requêtes. Tout cela conditionne l'implémentation de la fonction d'appariement. Et c'est à ce niveau que les systèmes P2P apportent une solution.

2.1.2 Rappel et précision

Avant de parler des modèles de Recherche d'Information (*comment* on fait de la RI), un mot sur l'évaluation. Classiquement, pour évaluer un système RI, on construit un oracle. C'est-à-dire qu'on réunit un ensemble de personnes, on leur donne un corpus documentaire et un ensemble de requêtes, et on leur demande de choisir « à la main » les documents qui répondent aux requêtes. Ensuite, on peut faire évaluer les requêtes par le système RI, et comparer ses résultats à ceux fournis par l'oracle.

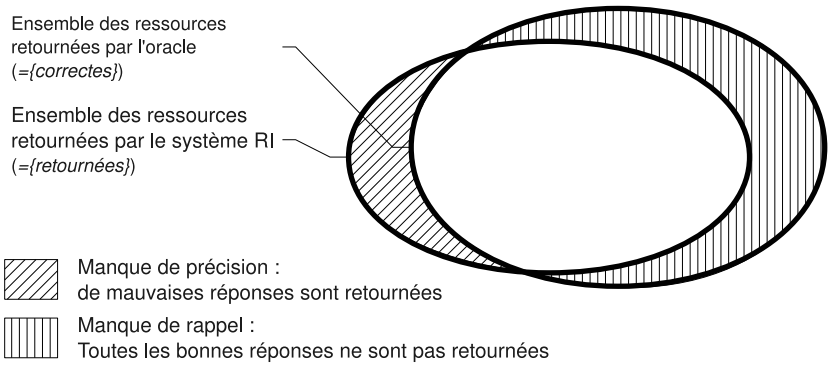


FIG. 2.2 – Qualité d'un résultat en Recherche d'Information : Rappel et Précision.

À partir de là, mesurer la qualité de la réponse retournée par le système est un problème ensembliste : il faut comparer l'ensemble des ressources retournées par le système à l'ensemble des ressources retournées par l'oracle (voir figure 2.2). Classiquement, on mesure le rappel et la précision du système. Le rappel donne le taux de couverture des bonnes réponses, ou la complétude du système :

$$Rappel = \frac{nombre\ de\ bonnes\ reponses\ retournees}{nombre\ de\ bonnes\ reponses}$$

$$Rappel = \frac{|retournees \cap correctes|}{|correctes|}$$

La précision mesure la qualité des réponses retournées, ou correction du système :

$$Precision = \frac{nombre\ de\ bonnes\ reponses\ retournees}{nombre\ de\ reponses\ retournees}$$

$$Precision = \frac{|retournees \cap correctes|}{|retournees|}$$

Jusque là, c'est assez simple, sauf qu'un système RI retourne rarement simplement un ensemble de réponses. En fait, dans la grande majorité des cas, un système RI trie les ressources du corpus par ordre de pertinence à la requête. Pour définir l'ensemble des ressources pertinentes, il faut donc choisir un seuil de pertinence. Avant ce seuil (valeur de pertinence forte), les ressources sont considérées pertinentes, au-delà, elles sont considérées comme non-pertinentes. En abaissant la valeur du seuil, on accepte plus de réponses. On privilégie ainsi le rappel, au détriment de la précision. Lorsqu'on relève le seuil, on a l'effet inverse. La valeur du seuil définit donc un compromis entre rappel et précision, et doit être choisie spécifiquement pour chaque contexte d'application. Pour que le résultat de l'évaluation permette de faire ce choix, on présente usuellement la qualité d'un modèle RI sous la forme d'une courbe, donnant la précision en fonction du rappel. Le rappel dépend directement de la valeur de seuil choisie, car pour un ensemble de réponses donné, on peut toujours trouver une valeur seuil donnant un rappel de 1, un rappel de 2 etc... On y fait correspondre la précision de la première réponse, des 2 premières réponses, etc... (cf. figure 2.3).

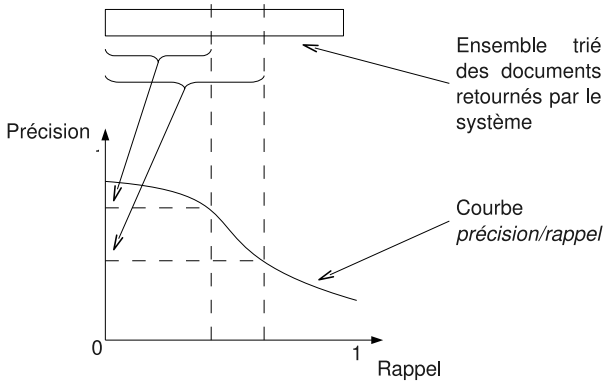


FIG. 2.3 – Construction d'une courbe Precision/Rappel.

2.1.3 Le modèle vectoriel

Le Modèle Vectoriel est sans doute le modèle de Recherche d'Information le plus emblématique. Il propose d'indexer les ressources dans un espace vectoriel : les ressources

sont représentées par des vecteurs. Chaque dimension de l'espace représente un concept. La coordonnée d'une ressource selon une dimension représente la pertinence du concept pour caractériser la ressource. Pour un tour d'horizon du domaine, voir par exemple (Witten et al., 1999) ou (Berry et al., 1999).

Dans la version simple du modèle, chaque concept correspond à un terme du vocabulaire utilisé dans le corpus. Une coordonnée d'un document donne le nombre de fois où le terme apparaît dans le document. Plusieurs améliorations sont cependant possibles ; nous allons citer les principales.

Une technique parmi les plus populaires est LSA (Latent Semantic Analysis). Cette technique propose d'orthogonaliser la base de l'espace vectoriel. Cela permet de réduire la taille de l'espace vectoriel et de mettre en évidence des concepts à partir des termes - on considère que les vecteurs obtenus sont de taille 50 à 350. En effet, lorsqu'on orthogonalise une base, on cherche un ensemble minimal de vecteurs orthogonaux permettant de décrire l'espace. Ces nouveaux vecteurs sont des combinaisons linéaires des vecteurs de l'ancienne base. Ils représentent donc des ensembles de termes pondérés : ce sont des concepts.

LSA est une technique coûteuse, que l'on fait en général précéder de pré-traîtements dont le but est surtout de réduire la taille du vocabulaire d'indexation, et donc le nombre dimensions de l'espace vectoriel. Tout d'abord, on peut utiliser une liste *Stop Words*. Il s'agit d'une liste répertoriant les mots peu ou non porteurs de sens, comme les propositions, conjonctions etc... On enlève ces mots du vocabulaire d'indexation. Ensuite, on peut procéder à une lemmatisation. Schématiquement, cela constiste à remplacer chaque mot par sa racine. Par exemple, vecteur, vectoriel, vectoriserai seront tous associés au terme d'indexation *vect*.

Notez bien qu'il s'agit là des techniques les plus utilisées, mais le processus d'indexation peut être beaucoup plus complexe, suivant les systèmes.

De la même façon que le modèle vectoriel permet d'indexer une ressource documentaire, il permet aussi d'indexer une requête. En fait, il permet de représenter n'importe quel ensemble de mots. Ainsi, un vecteur sémantique peut représenter une portion de texte, un chapitre par exemple, ou un corpus documentaire, comme les documents partagés par un pair.

Le but de cette indexation, c'est de permettre de calculer quels documents (resp. chapitres, pairs...) répondent à une requête donnée. à ce sujet, nous vous proposons de vous reporter aux descriptions des techniques utilisées dans PlanetP et pSearch, données dans les sections 5.1 et 5.3.

2.1.4 Un aperçu du domaine de la Recherche d'Information

Si le modèle vectoriel est le modèle le plus répandu, ça n'est pas le seul modèle RI, loin s'en faut. Nous reproduisons ci-dessous une classification des théories de Recherche d'Information proposée par M. Boughanem :

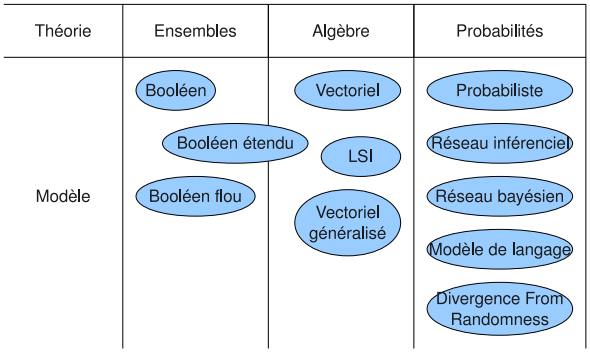


FIG. 2.4 – Les modèles de Recherche d'Information.

Il serait hors sujet de commenter ce tableau en détails ici. Quelques mots cependant sur quelques sujets intéressants :

Google On ne connaît pas les détails du fonctionnement de Google, mais on sait qu'il repose sur la technique dite *Page Rank*. Il s'agit d'inférer la pertinence d'une page hypertexte en comptant le nombre de liens qui pointent vers elle. La pertinence d'un lien hypertexte est elle-même pondérée par la pertinence de la page où il se trouve. Et la pertinence associée à une page est pondérée par la pertinence des liens qui pointent vers elle. Ces relations de pertinence peuvent être gérés à l'aide de *chaînes de Markov*.

Modèles de langage Certaines recherches visent à construire des modèles de langage, permettant par exemple de modéliser une phrase sous forme d'arbre, où on retrouve les propositions, dans lesquelles un verbe peut avoir pour fils son sujet et son complément... On cherche ici à analyser finement le sens des phrases. Cet exemple illustre l'importance des technologies de TALN (Traîtement Automatique du Langage Naturel) dans la Recherche d'Information.

Feedback et visualisation Enfin, sachez que certaines recherches considèrent un processus de Recherche d'Information plus complexe. En effet, l'un des problèmes en Recherche d'Information est que l'utilisateur est rarement expert du domaine sur lequel porte sa requête. Deux pistes sont privilégiées pour contrer cette difficulté. La première

consiste à introduire un cycle de rétro-action entre l'utilisateur et le système, basé sur un retour de l'utilisateur sur la qualité des réponses fournies par le système (feedback). Grâce au feedback, le système peut faire des propositions à l'utilisateur, lui permettant de raffiner sa requête de façon incrémentale. La seconde, c'est l'effort sur la visualisation des données, et/ou d'un domaine de connaissance. Cela permet au système de retourner une vision d'un domaine de connaissance, plutôt qu'une simple liste de résultats. Le but est toujours d'aider l'utilisateur à affiner sa requête.

2.2 Les techniques pair-à-pair de référence

Nous avons maintenant une vision globale du domaine de la Recherche d'Information. Voyons ce qu'il en est des systèmes P2P. Classiquement, on présente les techniques P2P comme pouvant être classifiées en trois catégories : les systèmes P2P « purs », les systèmes hiérarchiques et les systèmes structurés. Après avoir étudié ces trois types de systèmes, nous consacrerons une section à présenter la problématique des réseaux *Small World*.

2.2.1 Gnutella - Systèmes P2P « purs » - Mécanismes de sélection aléatoire

Gnutella (Klingberg and Manfredi, 2002) propose de résoudre les requêtes par inondation partielle du réseau. Dans sa version standard, le pair initiateur de la requête (pair source) la propage à quatre voisins, qui eux-mêmes la propagent à quatre voisins, et ainsi de suite sept fois. 23105 pairs sont ainsi contactés ($\sum_{i=1}^{7}(4^i)$), aux problèmes de pertes de messages et déconnexions près - ce qui suffit à introduire une grande variance. Chacun évalue la requête sur son corpus local et renvoie sa réponse au pair qui lui a transmis la requête. Les réponses sont ainsi retournées au pair source par routage arrière. La fusion des réponses (pondération, tri) est vue comme une opération de plus haut niveau, en principe opérée au niveau du pair source.

Gnutella fonctionne donc sous l'hypothèse que, pour toute requête, le sous-corpus associé à un sous-ensemble aléatoire de pairs donne une réponse satisfaisante. C'est pourquoi on l'appelle parfois système probabiliste. Le sous-ensemble de pairs est sélectionné par propagation aléatoire. Cette propagation forme un sous-arbre de recouvrement du réseau. Nous avons donc affaire à une *classe* de systèmes, dont les éléments diffèrent par l'arbre de propagation construit - largeur, profondeur, stratégie de parcours, stratégie de choix des voisins. Ces différents mécanismes donnent différents rapports

$$\frac{\textit{qualite} \ : \ \textit{qualite probabiliste de l'ensemble de pairs contacte}}{\textit{cout} \ : \ \textit{principalement nombre de messages et temps de reponse}}$$

On les appelle communément « systèmes pair-à-pair purs ». Nous préférons le terme de mécanismes de sélection aléatoire. En effet, le mécanisme de propagation des requêtes en lui-même n'est qu'un élément, un composant d'un système complet. Il faut par exemple router les réponses, construire la topologie, voir partager d'autres information comme les tables d'expansion proposées par (Nakauchi et al., 2004) (section 2.3.6).

(Gkantsidis et al., 2004) étudie les systèmes dits « P2P purs » produisant un arbre de recouvrement de largeur 1. C'est de lui que nous vient l'idée de comprendre Gnutella avant tout comme un moyen de sélectionner un ensemble aléatoire de pairs. La technique *iterative deepening* développée dans (Yang and Garcia-Molina, 2002) permet d'adapter

dynamiquement la taille de l'arbre de recouvrement - et donc le coût de la recherche - en fonction de la qualité des réponses retournées. Nous pensons que c'est un mécanisme de plus haut niveau qui peut être utilisé en combinaison d'autres mécanismes que les mécanismes de sélection aléatoire. Par contre, les deux autres techniques étudiées dans cet article construisent des descripteurs - voir des index - de pair ; elles sortent donc du cadre des mécanismes de sélection aléatoire.

2.2.2 KaZaA, eDonkey - Systèmes Hiérarchiques - Recherche inter-clusters

Les systèmes P2P purs sont réputés pour leurs bonnes qualités de passage à l'échelle. Par contre, ils consomment un grand nombre de messages. Les systèmes *hiérarchiques* compensent ce « défaut » en proposant un modèle semi-centralisé. Un sous-ensemble de pairs est sélectionné pour jouer le rôle de *super-pairs*. Ces super-pairs s'organisent en un réseau P2P, et jouent un rôle de serveur pour un ensemble de pairs « simples ».

Les systèmes *hiérarchiques* présentent donc un ensemble de clusters, administrés de façon centralisée. Le réseau des clusters est lui administré en pair-à-pair - pur ou structuré suivant le système. Avec la classe précédente de systèmes, on avait deux étapes de recherche : rechercher les pairs qui répondent à une requête puis rechercher dans ces pairs les ressources qui répondent aux requêtes. Les système *hiérarchiques* introduisent une étape de recherche supplémentaire : on recherche les clusters répondant à la requête avant de rechercher des pairs sur ces clusters.

Les systèmes hiérarchiques sont les plus connus du grand public, car très utilisés pour le partage de fichiers en P2P : KaZaA, eDonkey, BitTorrent, Gnutella2, ...

2.2.3 Chord, CAN, Pastry - Systèmes structurés - DHT

Dans cette section, nous exposons les principes de fonctionnement des systèmes structurés, puis nous expliquons dans les détails le fonctionnement du système Chord (Stoica et al., 2001) à titre d'exemple.

Principes Cette troisième catégorie de systèmes regroupe les tables de hachage distribuées (DHT : Distributed Hash Tables) et des systèmes dérivés. Les DHT sont des systèmes P2P offrant les fonctionnalités de tables de hachage - fonctions *get()* et *put()*. Ce sont donc des systèmes de stockage distribué.

Tout d'abord, les DHT répartissent l'administration des ressources entre les pairs. Pour cela, à chaque ressource est associée une clef. L'espace des clefs, que nous appellerons espace de nommage, est partitionné. Chaque pair se voit confié l'administration d'une partition. Ensuite, en manipulant les relations de voisinage, les pairs sont

ordonnés pour permettre de retrouver une clef de façon déterministe et rapide, par propagation de proche-en-proche entre les pairs. D'où le terme de système structuré. Cette structure est fortement contrôlée. Au final, lorsqu'une requête sur un nom de fichier est émise, l'application des fonctions de hachage sur le nom de fichier demandé permet de retrouver sa clef, puis par parcours du réseau le pair qui le maintient.

Chord Il existe de nombreuses propositions de DHT. Pour les illustrer, nous allons décrire les bases du système Chord (Stoica et al., 2001). Ce système est souvent cité en exemple, parce que relativement simple. Nous pourrons à partir de là amener les principes du système CAN (Ratnasamy et al., 2001), qui nous seront utiles pour la suite.

Chord utilise comme espace d'adressage un espace métrique de dimension 1 borné de 0 à 2^n - n est codé « en dur » dans le système, il est calculé par rapport aux propriétés du jeu de données.

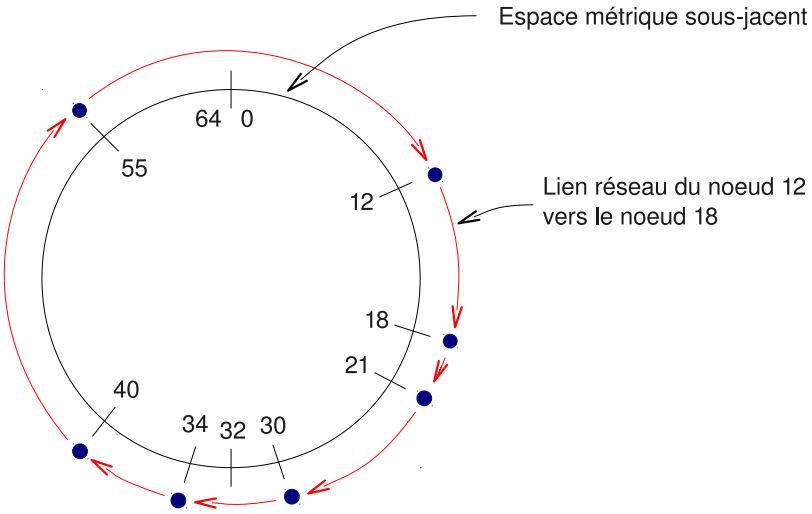


FIG. 2.5 – Un exemple de réseau Chord à 7 nœuds - l'espace métrique sous-jacent est de taille 64.

Le mapping du réseau sur cet espace d'adressage forme un cycle - cf. figure 2.5. Pour déterminer comment son administration est répartie entre les nœuds, on va d'abord attribuer une clé à chaque nœud (son adresse IP, par exemple), qui sera passée à une fonction de hachage, pour obtenir une adresse. Chaque nœud prend en charge l'administration des adresses contenues entre sa propre adresse et l'adresse du nœud qui le précède.

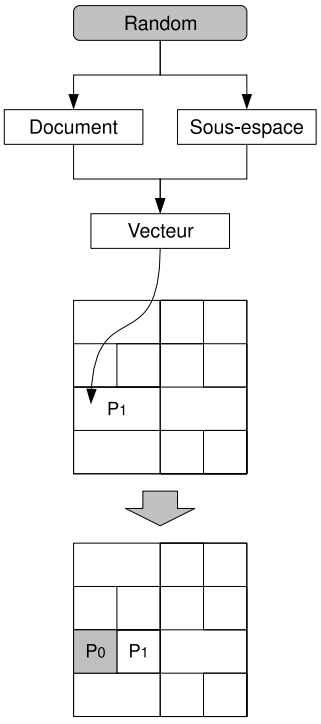
Pour optimiser le parcours de cette structure de réseau, chaque nœud maintient une liaison avec les nœuds administrant les adresses situées à une distance 2^k de lui. Ainsi, le nœud d'adresse 1 peut atteindre les nœuds administrant les adresses 2, 4, 8, 16, 32... Dans la limite des bornes de l'espace d'adressage.

Comme pour les pairs, à chaque ressource est associé un identifiant, typiquement son nom s'il s'agit d'un fichier. L'application d'une fonction de hachage sur cet identifiant donne la clef de la ressource. La ressource est confiée au pair qui administre la portion d'espace d'adressage à laquelle la clef appartient.

CAN Le système CAN (Ratnasamy et al., 2001) repose sur les mêmes principes, mais l'espace d'adressage qu'il utilise est un espace vectoriel à plusieurs dimensions.

CAN propose de construire un réseau logique fortement contrôlé. Sa topologie répond à la double problématique de partitionner l'espace de nommage (l'espace vectoriel) entre les pairs, et d'en permettre un parcours efficace. La structure est définie par le mécanisme d'insertion des pairs dans le réseau.
Le premier pair prend l'administration de l'ensemble de l'espace. Ensuite, lorsqu'un pair P_0 veut s'insérer dans un réseau, il doit d'abord s'attribuer des coordonnées - sa clef, un vecteur de taille m. Celle-ci est obtenue comme pour Chord à l'aide d'une fonction de hachage. Pour cela, il choisit au hasard une ressource et un sous-espace. Ses coordonnées sont la projection du vecteur de la ressource dans le sous-espace. On rappelle que l'espace est partitionné entre les pairs. Les coordonnées désignent donc une de ces partitions, gérée par un pair P_1.
Cette partition est partagée en deux selon une dimension choisie suivant ces règles :
- priorité au « plus grand coté » de la partition
- priorité aux partitions de plus petit rang (conventionnellement, elles correspondent à des concepts de plus grande importance)

L'administration d'une des deux moitiés est confiée à P_0, l'autre reste à P_1.



Les deux pairs doivent ensuite mettre à jour leurs relations de voisinage, de façon qu'elles respectent la logique de l'espace vectoriel. Soit d la dimension de partitionnement. Supposons que P_0 a pris la partition de gauche et P_1 celle de droite. Alors, P_0 récupère les voisins de gauche selon d, de P_1, qui les efface de ses tables de routage. Le voisin de droite de P_0 est P_1, le nouveau voisin de gauche de P_1 est P_0. Tous les voisins de P_1 selon les autres dimensions deviennent aussi voisins de P_0. oo

2.2.4 Les réseaux Small World

Nous allons dans cette section parler des réseaux *Small World*. Il s'agit d'une propriété empirique, observée (entre autres) sur le réseau Internet. Elle a été montrée comme applicable entre autres aux réseaux de partage de fichiers (en pair-à-pair ou non).

Selon la définition de Watts-Strogatz (Watts and Strogatz, 1998), un réseau est dit SmallWorld lorsqu'il satisfait deux propriétés :

- les distances sont courtes en moyenne ;
- le coefficient de clusterisation est important.

En fait, ces deux propriétés sont très liées. Ce que signifie cette définition, c'est que la topologie du réseau se rapproche d'un réseau de nuages de nœuds fortement connexes.

Cette topologie caractérise un réseau où les nouvelles se propagent vite (les distances sont courtes), et en priorité vers un certain sous-ensemble de nœuds (ceux qui appartiennent au même cluster que l'émetteur).

L'article (Iamnitchi et al., 2004) étudie le graphe des fichiers partagés sur les réseaux de partage de fichiers en pair-à-pair. Le graphe des fichiers partagés est obtenu à partir de la description des fichiers détenus par les nœuds d'un réseau P2P. On associe à chaque nœud du réseau un nœud sur le graphe, et on crée un lien entre deux nœuds s'ils détiennent un même fichier. Cela donne un graphe des domaines d'intéret des utilisateurs. Les auteurs montrent que ces graphes sont Small World, et que cette caractéristique trouve son origine à la fois dans la répartition des fichiers - qui suit une loi de Zipf, une loi de répartition hyperbolique (de type $1/x$) généralement observée sur Internet - et dans le comportement des utilisateurs - la façon dont ils échangent des fichiers.

On en déduit que les réseaux de partage de fichiers sont fortement clustérisés. On voit apparaître des groupes d'utilisateurs ayant des goûts similaires, et donc beaucoup de fichiers communs, inter-connectés par un faible nombre d'utilisateurs dont les goûts sont à cheval sur plusieurs clusters.

Cette loi nous intéresse donc par ce qu'elle régit la topologie du réseau Gnutella (cf. section 2.2.1) et le graphe des fichiers partagés des réseaux de partage de fichiers en P2P. Nous garderons cette loi à l'esprit, pour la génération de jeux de données pour nos simulations.

2.2.5 Synthèse

Nous désignerons par *modèle {pur, hiérarchique, structuré}* le modèle descriptif des systèmes P2P que nous venons de décrire, et qui semble aujourd'hui faire largement consensus. Nous avons cependant déjà distillé quelques éléments montrant que ce modèle pouvait être contesté. Nous montrerons dans une prochaine section qu'en pratique il se révèle pauvre en compréhension. Bien que très utile pour nommer les principaux types de systèmes - et nous continuerons à l'utiliser dans cette optique - il comporte des lacunes lorsqu'il s'agit de les classifier, les comparer et les comprendre. Le principal enjeu de cette thèse est de construire un nouveau modèle, sur lequel nous nous appuierons pour comprendre, évaluer et concevoir des systèmes P2P.

2.3 Recherche d'Information en P2P : les systèmes de référence

Nous savons maintenant quels sont les enjeux de la Recherche d'Information et quelles sont les principales techniques P2P. Nous pouvons passer à l'étude des propositions de systèmes de Recherche d'Information en Pair-à-pair (RI-P2P). Nous avons choisi un échantillon de six systèmes, représentatifs des principales techniques RI-P2P : PlanetP (Cuenca-Acuna et al., 2003), pSearch (Tang et al., 2003), *Routing Indices* (Crespo and Garcia-Molina, 2002a), *Semantic overlay networks* (Crespo and Garcia-Molina, 2002b), la technique de construction d'index par observation du comportement de (Handurukande et al., 2004) et la technique d'expansion de requêtes de (Nakauchi et al., 2004).

2.3.1 PlanetP - Index global réparti

PlanetP propose d'adapter à un contexte P2P une technique bien connue en Recherche d'Information centralisée : les filtres de Bloom. Il utilise cette technique pour construire les signatures des pairs. Ensuite, ces signatures sont diffusées sur le réseau par un mécanisme de gossiping ; cette diffusion permet à chaque pair de se constituer une table de routage. Finalement, les filtres de Bloom présents dans les tables de routage sont utilisés pour améliorer le routage des requêtes, comme indiqué dans la figure 2.6.

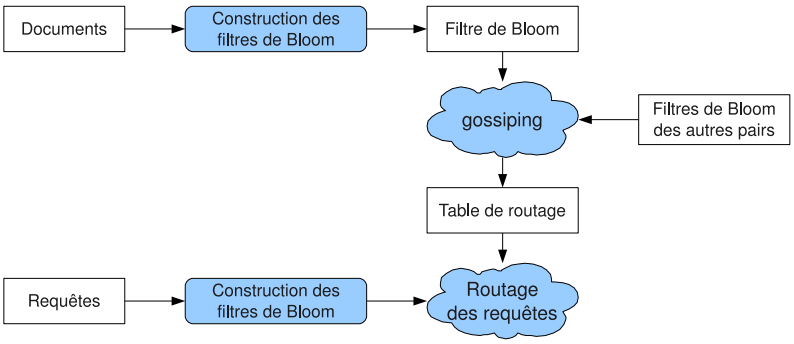


FIG. 2.6 – PlanetP : filtres de Bloom, gossiping et routage.

Construction des filtres de Bloom PlanetP propose donc de construire des signatures des pairs sous forme de filtres de Bloom. Le Filtre de Bloom associé à un pair représente son contenu, c'est-à-dire les concepts clés utilisés dans les documents qu'il partage. Il se présente sous la forme d'un vecteur de bits (de même taille pour tous les pairs), dans lequel des mots sont codés par des séquences de bits à 1. Un filtre de Bloom à zéro signale donc un corpus vide.

Voyons d'abord comment construire le filtre de Bloom représentant un mot (cf. figure 2.7). On initialise le filtre de Bloom à 0. Puis on utilise k fonctions de hachages, qu'on applique successivement au mot. Ces fonctions déterminent k bits du vecteur qui sont mis à 1. Pour construire le filtre de Bloom associé à un document, on fait le OR logique des filtres de Bloom des mots qu'il contient. De même, pour construire le filtre de Bloom associé à un pair, on fait le OR logique des filtres de Bloom des documents de son corpus (cf. figure 2.8).

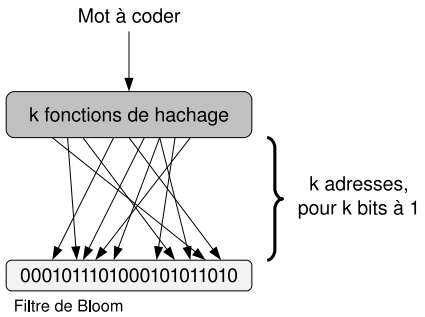


FIG. 2.7 – Construction d'un filtre de Bloom.

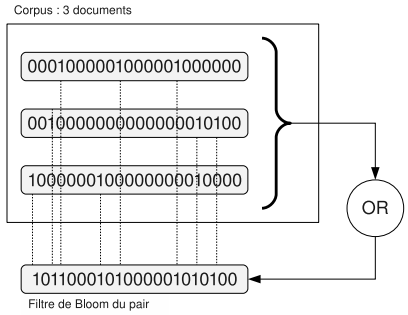


FIG. 2.8 – Construction du filtre de Bloom d'un pair à partir des filtres de Bloom de ses documents.

Notez que dans le cas du filtre de Bloom d'un document, il est possible d'appliquer au préalable une technique d'extraction des mots « importants » de ce document, pour ne fusionner que les filtres de Bloom de ces mots caractéristiques - les auteurs préconisent d'utiliser une liste *stop words*.

On peut également remarquer que les filtres de Bloom peuvent donner des faux positifs, mais jamais de faux négatifs. En effet, si un mot est présent dans un corpus, les bits correspondants seront nécessairement à '1' dans le filtre de Bloom du pair. Un filtre de Bloom signale donc bien tous les mots présents dans le corpus. Par contre, la somme (OU logique) des séquences de bits de plusieurs mots peut faire apparaître la séquence de bits caractéristique d'un mot qui n'appartient pas au corpus. C'est ainsi que sont créés des faux positifs.

Cette technique présente plusieurs atouts. D'abord, l'utilisation des filtres de Bloom nécessite peu d'informations partagées entre les pairs : uniquement la taille des filtres et les fonctions de hachage. On n'a notamment pas besoin de partager le vocabulaire global. Ensuite, en comparaison d'autres types d'index, ce sont des objets de petite taille ; auxquels on peut par contre reprocher un certain manque de précision - faux positifs, perte d'information par rapport aux vecteurs non-compressés (décrits section 2.1.3).

Chaque pair construit et met donc à jour sa signature localement. Puis il la diffuse par gossiping.

Le Gossiping Le gossiping est un mécanisme de propagation épidémique par échange de proche-en-proche. Il permet de diffuser des informations dans un réseau, sur une base fondamentalement aléatoire. Chaque pair diffusant sa signature, il reçoit aussi des signatures provenant d'autres pairs. Il peut ainsi se constituer une table de routage.

Il existe de nombreuses techniques de propagation épidémique. Celle proposée par les auteurs repose sur trois mécanismes :

Push Les rumeurs sont propagées toutes les T_g secondes à un voisin choisi aléatoirement. Un pair arrête de propager une rumeur lorsqu'il a contacté n pairs consécutifs qui l'ont déjà reçue.

Pull Algorithme anti-entropie. Chaque pair demande toutes les T_r secondes à un voisin choisi aléatoirement un résumé de sa table de routage. Cela lui permet de demander à ce voisin les informations qu'il n'a pas.

Partial-Pull Lorsqu'un pair reçoit une requête Pull, il renvoie un résumé de sa table de routage. Le pair originaire de la requête peut ainsi lui demander les filtres de Bloom qui ne sont pas à jour.

Routage : évaluation des requêtes Lorsqu'une nouvelle requête est émise sur un pair, celui-ci construit le Filtre de Bloom correspondant, puis le compare aux signatures de pair présentes dans la table de routage. Cela permet de cibler les pairs les plus susceptibles de répondre à la requête, et donc de réduire le nombre de messages utilisés pour propager les requêtes.

La comparaison se fait à l'aide d'une fonction de distance, prenant en entrée deux filtres de Bloom et retournant un entier. Cette fonction permet d'ordonner les pairs par proximité à la requête. Elle utilise une mesure de type TF*IDF.

TF*IDF (Term Frequency * Inverse Document Frequency) est une technique permettant d'évaluer la pertinence d'un mot pour représenter le contenu d'un pair. On considère que plus le mot apparaît fréquement sur le pair, plus il est caractéristique de son contenu. D'où l'introduction du terme TF : la fréquence d'apparition du terme dans le pair. Le terme IDF permet de relativiser la fréquence d'apparition du terme dans le pair par sa fréquence d'apparition dans le corpus global : un terme n'est pas pertinent parce qu'il apparaît souvent dans le pair, mais parce qu'il apparaît significativement plus souvent dans le pair que dans le corpus global. IDF est l'inverse de la fréquence du terme dans le corpus global.

Pour ne pas avoir à calculer de statistique sur le corpus global, les auteurs proposent que chaque pair calcule l'IDF à partir des filtres de Bloom contenus dans sa table de

routage. C'est l'IPF (Inverse Peer Fequency) ; soit IPF_t l'IPF d'un terme :

$$IPF_t = 1 + log(\frac{nombre\ de\ pairs}{nombre\ de\ pairs\ possedant\ t}))$$

La mesure de pertinence d'un pair pour une requête est donnée par la somme des IPF des termes de la requête apparaîssant dans le corpus du pair, soit :

$$Pertinence(P, Q) = \sum_{t \in Q \cap P} IPF_t$$

Cette mesure donne un rang à chaque pair, permettant de les ordonner par ordre de pertinence à la requête. Soient N le nombre de pairs du système et N_{BF_x} le nombre de pairs possédant le terme décrit par le filtre de Bloom BF_x, le procesus de mesure de pertinence des pairs peut être décrit comme sur la figure 2.9 :

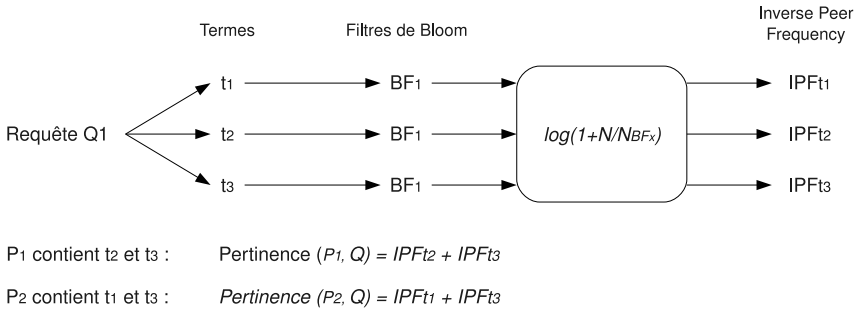


FIG. 2.9 – Propagation d'une requête avec PlanetP.

Notez que la mesure de similarité d'une requête à un document sera calculée localement par les pairs auxquels la requête est routée. Elle est calculée par rapport à l'IPF, soit, si TF_t est la fréquence du terme t dans le document D et $|D|$ la taille du document D (nombre de termes caractéristiques) :

$$Sim(Q, D) = \frac{\sum_{t \in Q} IPF_t \times (1 + log(TF_t))}{\sqrt{|\ D\ |}}$$

Une technique d'optimisation du gossiping Au cours de leurs tests, les concepteurs de PlanetP se sont rendus compte que les pairs à faible connectivité pénalisaient fortement les performances du mécanisme de gossiping. Pour éviter de pénaliser les pairs à bonne connectivité, ils proposent le mécanisme *bandwidth-aware gossiping*. Ce mécanisme inclue la notion de *fast peers* et *slow peers*, munis d'une probabilité de propager l'un vers l'autre. La probabilité qu'un *fast peer* envoie un message de gossiping à un *slow peer* est de 1%. Finalement, les *fast peers* font très largement du gossiping entre eux, de même pour les *slow peers*.

Bilan PlanetP peut donc être caractérisé selon deux dimensions :

- la recherche n'est pas aléatoire : les pairs sont indexés. Les index décrivent le contenu des pairs selon le modèle vectoriel et sont compressés par la technique des filtres de Bloom.
- l'index des pairs est implémenté sous la forme d'un fichier distribué réparti entre les pairs. Sa cohérence (faible) est maintenue par le mécanisme de gossiping.

Le lien entre gossiping et filtres de Bloom est explicite dans l'article : ces derniers, en compressant la taille des index, facilitent la distribution. On adapte donc ici le modèle RI par rapport à une contrainte du modèle P2P.

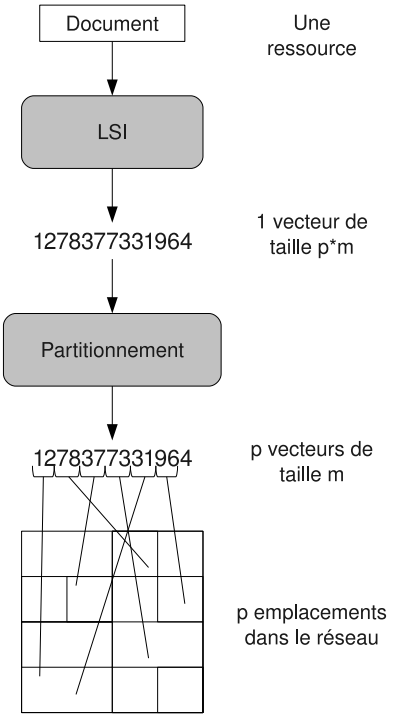
2.3.2 pSearch - Structure d'indexation répartie

Les auteurs de pSearch le définissent comme la rencontre entre le système de Recherche d'Information LSI et le système P2P CAN. Nous avons déjà décrit LSA dans la section 2.1.3, traitant du modèle vectoriel, et nous avons décrit le système CAN dans la section 2.2.3. Le principe, c'est d'utiliser les index construits par le modèle vectoriel comme clefs dans le réseau CAN. Cela pose trois difficultés :

- Il y a une différence de dimensionnalité entre LSI et CAN. En effet, LSI est connu pour construire des vecteurs d'entre 50 et 350 dimensions ; ce qui est trop important pour que le réseau CAN fonctionne bien.
- Les vecteurs LSI n'assurent pas une distribution uniforme des ressources sur les pairs. En effet, dans CAN, les clefs sont construites à l'aide de fonctions de hachage, qui assurent (ou tendent à assurer) une répartition uniforme des clefs dans l'espace des clefs. Mais dans LSI, les vecteurs décrivent des concepts. rien n'assure que la répartition des ressources dans l'espace des concepts soit uniforme.
- *Curse of dimensionnality.* Il s'agit d'un inconvénient connu des modèles multidimensionnels comme le modèle vectoriel. Dans un espace à beaucoup de dimensions, les distances ont tendance à se « tasser ». Il devient alors difficile de discriminer les ressources qui répondent (ou pas) à une requête. Concrètement, la zone de recherche explose, ce qui se traduit dans un cadre P2P par un grand nombre de pairs parcourus, donc un fort coût de recherche.

Réduire la différence de dimensionnalité : les *Rotated Spaces* Pour ramener la taille des index de ressources créés par LSI à une dimensionnalité acceptable pour le réseau CAN, les auteurs proposent le mécanisme des *Rotated Spaces*.

Il s'agit de faire p rotations du vecteur créé par LSI et d'indexer les ressources p fois, suivant les m premières dimensions de chacune de ces rotations. Dit autrement, l'espace vectoriel est divisé en p sous-espaces de taille m. Chaque ressource est indexée p fois, selon les p projections de son vecteur dans chacun des p sous-espaces. Les ressources sont donc maintenues (et peuvent être retrouvées) en p points du réseau.



Les auteurs précisent que LSI est connu pour trier les dimensions, suivant l'importance du concept qu'elles représentent. Ainsi, les dernières dimensions représentent des concepts peu discriminants et peuvent être ignorées, en quel cas $p*m$ sera inférieur à la taille du vecteur initial.

Assurer une distribution uniforme des ressources l'objectif, c'est de répartir la charge sur les pairs, donc que les ressources soient distribuées uniformément sur les pairs. Dans la mesure où LSI impose que les clefs des ressources ne soient pas réparties uniformément dans l'espace d'adressage, une solution est de faire en sorte que l'espace d'adressage ne soit pas partitionné uniformément lui non plus. Les auteurs proposent de jouer sur les clefs associées aux pairs. Pour construire la clef associée à un pair, ils choisissent une ressource au hasard puis une projection au hasard. Ainsi, statistiquement, la répartition des clefs des pairs suit la répartition des clefs des ressources. Donc les portions d'espace contenant plus de ressources sont plus finement partitionnés.

Problème de *curse of dimensionnality* : Les *Sample Sets* La recherche dans pSearch fonctionne en deux étapes. Il faut d'abord trouver le point du réseau correspondant à la requête, puis parcourir le voisinage de ce point, pour récupérer les ressources

voisines sémantiquement de la requête. Le problème de *curse of dimensionnality* apparaît lors de cette deuxième étape. Pour le résoudre, les auteurs mettent-en place pour la seconde étape de recherche un second type d'index de pair, associé à un second algorithme de parcours du réseau : les *Sample Sets*.

Un *Sample Set* est un échantillon des ressources maintenues et des requêtes émises par un pair. Chaque pair maintient un *Sample Set* sur chacun de ses voisins, pour chacun des p sous-espaces. L'ensemble des *Sample Sets* pour un sous-espace forme ainsi un étiquetage du réseau, très similaire dans son fonctionnement - et son implémentation - à la technique *Routing Indices* décrite dans la section suivante.

Dans la suite, nous nous situons dans un des sous-espaces. Le processus de construction des *Sample Sets* est décrit dans la figure 2.10. Chaque pair doit au préalable calculer son centroide : le barycentre de ses ressources et des requêtes « récemment » émises - le terme récemment n'est pas explicité. Sur le pair p_1, le *Sample Set* décrivant le pair p_2 est l'ensemble des k_c index de ressources de p_2 les plus proches du centroide de p_1 et de k_r index de ressources de p_2 choisis aléatoirement.

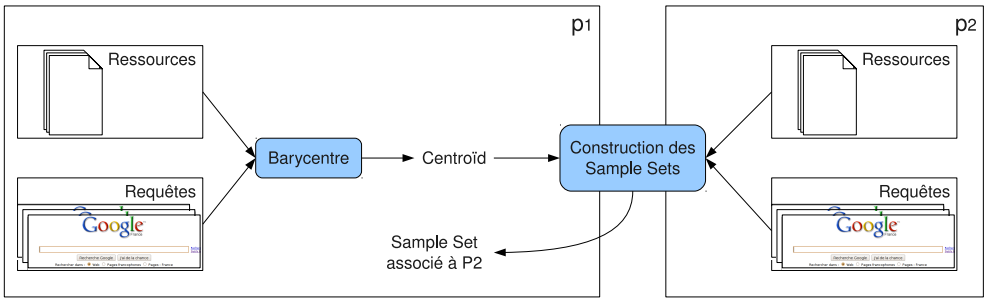


FIG. 2.10 – Construction des *Sample Sets*.

Résolution des requêtes La résolution des requêtes fonctionne en deux étapes. La première consiste à retrouver le point sémantique correspondant à la requête et s'appuie sur le premier type d'index : les vecteurs sémantiques. La seconde consiste à parcourir une boule sémantique autour de ce point - les ressources au contenu voisin de celui recherché par la requête - et s'appuie sur le second type d'index : les *sample sets*.

La première étape de recherche suit le fonctionnement d'une DHT CAN. Les requêtes sont propagées récursivement au voisin dont le vecteur sémantique est le plus proche de celui de la requête, jusqu'à arriver sur le pair qui maintient la portion d'espace à laquelle appartient la requête. Pour chaque requête, p sous-requêtes sont générées dans chaque dimension, qui désignent chacune un pair que nous appellerons P_Q^i, où Q désigne la requête et i la dimension.

La seconde phase de recherche utilise une mesure de similarité associée aux *sample sets*, qui retourne la distance de la requête à l'élément du *sample set* qui lui est le plus proche :

$$sim2(Q, P_Q, P) = max_{\ V \in sampleSet(P_Q,P)}\ cos(V, Q)$$

où P_Q est le cluster courant de Q et P est un voisin de P_Q. Les clusters sont sélectionnés dans l'ordre par groupe de b, jusqu'à ce que les T derniers clusters sélectionnés ne rapportent pas de meilleure réponse. b et T sont calculés suivant le système suivant :

$$b = min(d, T/2)$$

$$T = max(5, F - 5) * 0.8^w$$

$$w = min_{P \in N} distance(P_Q^i, P)$$

où N est la file des clusters sélectionnés, P_Q^i le cluster obtenu par la première phase de recherche, dans le *rotated space* i, d et F des constantes (à choisir empiriquement).

La file N est initialisée suivant les règles suivantes :

1. Les pairs sélectionnés pendant la première phase de recherche ($\{P_Q^i\}_i$), sont tous sélectionnés.
2. Les voisins de P_Q^0, le pair correspondant à la sous-requête du premier sous-espace, sont tous sélectionnés.

Tous ces pairs sont interrogés en parallèle.

Mécanismes de construction et maintenance *pSearch* hérite des mécanismes de construction et maintenance du réseau CAN : *join* et *leave* gèrent respectivement les arrivées et départs de pairs, *takeover* est une procédure rapide de gestion d'un défaut de voisin, *maintenance* fonctionne en tâche de fond, pour détecter et réparer les défauts dans la topologie créés par *takeover*.

Les auteurs n'en parlent pas, mais il est nécessaire de mettre les *sample sets* à jour lorsque le contenu d'un pair change. En principe, il n'est pas nécessaire de faire des mises-à-jour systématiques, une cohérence faible doit suffir. Mais cela reste à évaluer.

Bilan pSearch peut donc être caractérisé selon quatre dimensions :

Première étape de recherche :

- la recherche n'est pas aléatoire : les pairs sont indexés. Les index décrivent le contenu des pairs selon l'implémentation LSI du modèle vectoriel.
- les pairs sont indexés dans une structure de recherche. Cette structure est implémentée sur le réseau P2P. Sa cohérence est maintenue de façon forte.

Seconde étape de recherche :

Cette seconde indexation est prévue pour fonctionner conjointement à la première, mais peut être définie indépendamment. C'est-à-dire qu'en principe, elle fonctionne bien uniquement si elle est associée au réseau structuré. Mais concrêtement, elle peut être implémentée toute seule, ou au-dessus de toute autre topologie réseau.

- la recherche n'est pas aléatoire : les pairs sont indexés. Les index décrivent le contenu des pairs selon la technique des *Sample Sets*.
- L'étiquetage du réseau est construit indépendamment pour chaque lien (connexion), par un algorithme distribué entre les deux pairs concernés.

2.3.3 *Routing Indices* - Réseau étiqueté

Principe Le mécanisme *Routing Indices* est conçu pour être implémenté au-dessus d'un réseau de type Gnutella. Le principe est de mettre en place des index - les RI, *Routing Indices* - permettant de guider le routage des requêtes vers les portions de réseau qui possèdent le plus de réponses. Chaque pair maintient donc un RI associé à chacun de ses voisins. Le RI associé à un pair représente l'ensemble des documents dans le sous-réseau accessible par lui, triés par *topics* (sujets ou concepts). Les RI ressemblent donc fortement à des vecteurs sémantiques : chaque dimension représente un *topic*, la coordonnée dans une dimension représente le nombre de documents dans ce *topic*. Un RI donne en plus le nombre total de documents dans le sous-réseau, tous *topics* confondus.

Propagation des requêtes Sur un pair P donné, pour router une requête, on calcule la pertinence de la requête à chacun des voisins V_i de P, puis on choisit le voisin qui obtient le plus grand score. Les auteurs proposent d'utiliser un routage en profondeur, avec arrêt de la propagation si la requête est satisfaite - la notion de satisfaction n'est pas plus précisée.

La mesure de pertinence utilisée est la suivante, pour un voisin V_i de P, soit *topics* l'ensemble des « topics » :

$$pertinence(V_i) = nb\_docs\_accessibles\_par\_V_i \times \prod_{topics} \left(\frac{RI_P(V_i)[topic]}{nb\_docs\_accessibles\_par\_V_i}\right)$$

où $RI_P(V_i)[topic]$ est la valeur connue par p de l'index de routage de V_i pour le concept *topic*.

Si on reprend l'analogie avec le modèle vectoriel, il s'agit du produit des TF (Term Frequency), normalisés (division par le nombre de documents) ; multiplié par le nombre de documents. La pertinence d'un sous-réseau dépend donc de la proportion de documents pertinents, et du nombre total de documents.

Calcul des RI Soient P un pair, $\mathcal{V}$ l'ensemble de ses voisins, $V_i \in \mathcal{V}$ un de ses voisins et $RI(P)$ le RI représentant le contenu de P. Le RI associé à P sur V_i représente l'ensemble des documents accessibles par P, ce qui correspond à l'ensemble des documents sur P, plus l'ensemble des documents accessibles par ses voisins. Donc le RI associé à P sur V_i est la somme de $RI(P)$ et des RI de tous ses voisins sauf V_i :

$$RI_{V_i}(P) = RI(P) + \sum_{V_j \in (\mathcal{V} \setminus V_i)} RI_P(V_j)$$

où $RI_{V_i}(P)$ (lire « RI de P sur V_i ») décrit le sous-réseau accessible par P, dans la table de routage de V_i ; et $RI(P)$ décrit les ressources de P. C'est P qui va calculer ses RI pour chacun de ses voisins, puis les leur transmettre. Chaque pair du réseau calcule ainsi son descripteur pour chacun de ses voisins.

Mise-en-place et maintenance des RI Lorsqu'un pair arrive sur le réseau, il calcule tout d'abord le descripteur de ses ressources (RI_P). En parallèle, ses voisins lui transmettent leurs descripteurs ($RI_P(V_i)$). À partir de $RI(P)$ et $\{RI_P(V_i) | V_i \in \mathcal{V}\}$, P peut calculer ses descripteurs pour chacun de ses voisins - les $RI_{V_i}(P)$. Chaque voisin doit ensuite mettre-à-jour ses descripteurs sur ses voisins, puisque de nouvelles ressources sont accessibles par lui. Et ainsi de suite, tous les RI du réseau sont mis-à-jour récursivement.

De même lorsque le contenu d'un pair est modifié, il met-à-jour son descripteur, puis met-à-jour ses descripteurs sur ses voisins, qui eux-mêmes mettent-à-jour leurs descripteurs sur leurs voisins et ainsi de suite récursivement.

Le problème avec ce mécanisme, c'est qu'il ne fonctionne pas s'il y a des cycles. Nous y revenons plus loin ; mais d'abord, nous étudions deux variantes proposées par les auteurs.

Hop-count RIs Cette solution consiste à distinguer dans le RI associé à un pair les documents accessibles à 1 saut, 2 sauts, etc... On a donc k RI, donnant les documents accessibles en k sauts. Nous les noterons $RI[0], ..., RI[k]$; donc $RI_P(V)[topic][s]$ est le nombre de documents parlant de *topic* accessibles en s sauts par V, depuis P.

Une nouvelle mesure de pertinence est associée aux *hop-count RIs*. Le principe est de moduler la pertinence à s sauts par le nombre de messages nécessaires pour interroger ces pairs. On calcule donc un taux de pertinence par unité de message ; calculé en moyenne sur chaque nombre de sauts jusqu'à l'horizon. La pertinence d'un voisin V_i est donnée par la formule :

$$pertinenceHCRI(V_i) = \sum_{s=0..k} \frac{pertinence(V_i[*][s])}{(degre\_moyen - 1)^s}$$

où $pertinence(V_i)[*][s]$ correspond à la mesure de pertinence des RI classiques, calculée sur le RI au s^{eme} saut.

$(degre\_moyen - 1)^s$ se veut être une approximation du nombre de messages nécessaire pour interroger les voisins situés à j sauts : « Under these assumptions, it takes F^h messages to find all documents at hop h. ». Mais c'est une erreur. $(degre\_moyen - 1)^s$ donne le nombre de voisins à s sauts. En réalité, il faut $\sum_{i=1..s}(degre\_moyen-1)^i$ messages : $(degre\_moyen-1)$ messages pour atteindre les voisins à 1 saut, $(degre\_moyen-1) + (degre\_moyen - 1)^2$ messages pour atteindre les voisins à 2 sauts, etc...

Exponential RIs La technique *Exponential RIs* (ERI) propose de prendre en compte la distance des pairs dans le calcul des RI, en pondérant la pertinence d'un pair pour un *topic* par la distance de ce pair. Ainsi, pour calculer $RI_{V_i}(P)$, au lieu de faire une somme simple des RI des autres voisins, on fait :

$$\frac{\sum_{V_j \in (\mathcal{V} \setminus V_i)} \{RI_P(V_j)\}}{degre\_moyen - 1} + RI(P)$$

Ainsi, P étant plus près de V_i que les V_j, son contenu entre plus fortement en compte dans le calcul de $RI_{V_i}(P)$. À s sauts, $RI(P)$ aura été divisé s fois par $degre\_moyen-1$.

Gestion des cycles Les auteurs passent rapidement sur cet aspect et renvoient à l'état-de-l'art. Ils précisent cependant que trois approches sont envisageables :

- Ne rien faire, pour HCRI et ERI. En effet, HCRI fonctionne dans la limite d'un horizon, et ne permet donc pas de boucle infinie. En ce qui concerne ERI, dans une boucle, la valeur d'une mise-à-jour étant divisée par $degre\_moyen - 1$ à chaque propagation, elle deviendra rapidement très petite et sera ignorée. Cependant, ces solutions dégradent la qualité des RI et entraînent un surcoût de maintien.
- Agir sur la topologie sous-jacente pour qu'elle ne comporte pas de cycle.
- Éviter les cycles dans le mécanisme de mise-à-jour des RI.

Bilan *Routing Indices* peut donc être caractérisé selon deux dimensions :

- La recherche n'est pas aléatoire : les pairs sont indexés. Les index décrivent le contenu disponible sur le sous-réseau accessible par un pair, selon la technique des *topics*, apparentée au modèle vectoriel.
- Ces index sont calculés coopérativement par un algorithme distribué, et constituent un étiquetage du réseau.

2.3.4 Semantic Overlay Networks - Réseau clustérisé

(Crespo and Garcia-Molina, 2002b) propose de clustériser les ressources, pour gérer chaque cluster sur un réseau logique indépendant : un *SON* (Semantic Overlay Net-

work). Lorsqu'une requête est émise, il faut donc choisir le *SON* sur lequel elle sera résolue. Les auteurs appuient leur étude sur l'exemple du partage de fichiers musicaux, clustérisés par style. Le système fonctionne en trois étapes :

1. Un premier mécanisme décide de la classification utilisée, par analyse du jeu de données (répartition des ressources sur les pairs) ; et permet aux pairs d'en partager la connaissance.
2. En fonction de cette classification, les pairs peuvent classifier leurs ressources puis adhérer aux *SONs* correspondants.
3. Les requêtes sont classifiées localement puis résolues dans le *SON* correspondant.

Les auteurs choisissent la classification en amont et la codent en dur dans le système. Ils remarquent qu'un mécanisme de type Gnutella peut être utilisé pour résoudre les requêtes sur les *SON* et y répartir les pairs, mais ne spécifient pas d'implémentation.

La hiérarchie de classification Les auteurs proposent d'utiliser une classification de type hiérarchique, choisie de façon statique, par analyse du jeu de données. On peut considérer qu'elle est un paramètre du système. Une bonne classification permet de minimiser le nombre de pairs dans chaque *SON* et le nombre de *SONs* auxquels appartient chaque pair, tout en préservant le rappel.

Les auteurs étudient le nombre de pairs par cluster et le nombre de clusters par pair, pour plusieurs classifications. Ils en déduisent que la classification par styles est plus adaptée - elle aussi plus souple à utiliser, car on peut diviser les SON en utilisant les sous-styles. On peut remarquer que les auteurs ne cherchent pas à déterminer les rapports $\frac{nombre\ de\ pairs}{cluster}$ et $\frac{nombre\ de\ clusters}{pair}$ moyens optimaux.

En comparant une classification automatique à une classification manuelle, les auteurs trouvent 25% d'erreur. Celles-ci sont dues à :

- des fichiers dont le nom est mal formaté (normalement, le nom est de la forme <auteur>_<titre>.mp3)
- des fichiers non-musicaux
- des fautes d'orthographe dans les titres

On peut remarquer que deux des sources d'erreur citées ici on un effet neutre. En effet, si la mauvaise classification provient du nom de fichier mal formaté ou d'une faute d'orthographe, cette même erreur ne permettra pas de trouver que le document répond à la requête, même dans le cas d'un système non-clustérisé.

Malgré ce taux d'erreur important dans la classification des ressources, les pairs sont bien clustérisés, avec seulement 4% d'erreur. Cela s'explique par la qualité de la clustérisation naturelle des ressources sur les pairs : lorsqu'une ressource est mal classifiée, d'autres ressources bien classifiées attestent du domaine d'intérêt du pair.

Coté requêtes, pour le jeu de données utilisé, avec une classification manuelle, 8% des requêtes sont classifiées à la racine de la hiérarchie (à évaluer sur l'ensemble des SONs), 78% le sont au niveau style et 14% au niveau sous-style.

Remarque Il aurait été possible - et sans doute intéressant - d'utiliser une classification en plusieurs dimensions ; par exemple style, année, genre (engagée, d'amour...).

Topologie et résolution des requêtes La topologie est construite comme suit : à chaque élément de la hiérarchie de classification est associé un *SON*. Un pair peut appartenir à plusieurs *SON* selon les règles suivantes. Soit un pair P, et une valeur seuil s, paramètre du système :

- Si plus de $s\%$ des ressources de P appartiennent à un *SON*, alors P intègre ce *SON*.
- S'il existe un ensemble de cluster frères $\{Cl_1...Cl_k\}$ que P n'a pas intégrés selon la première règle ; alors P intègre le père de ces clusters si et seulement si la somme des ressources de $\{\{Cl_1\} \cup \{Cl_2\} \cup ...\{Cl_k\}\}$ dépasse $s\%$ des ressources de P.
- Tous les pairs intègrent le *SON* racine (correspondant à la racine de la hiérarchie de classification).

Les requêtes peuvent être classifiées automatiquement ou bien l'utilisateur peut choisir à la main sur quel *SON* elle est lancée. Une requête est émise sur un *SON*. Si trop peu de réponses sont reçues, elle est renvoyée sur le père de ce *SON*, et ainsi de suite jusqu'à ce que suffisamment de réponses soient reçues.

Bilan Le système *SON* peut donc être caractérisé selon deux dimensions :

- La recherche n'est pas aléatoire : les pairs sont indexés. Les index décrivent le contenu des pairs. Un index est un ensemble de concepts : ceux auquels le pair est associé.
- On sait que les pairs sont clustérisés. Mais les modèles de répartition inter et intra-clusters ne sont pas spécifiés ; les mécanismes de répartition de ces modèles non plus.

2.3.5 Indexation à partir du comportement

Jusqu'ici, les systèmes que nous avons étudiés construisent les index de pair à partir des informations de contenu : les ressources maintenues et les requêtes émises par ces pairs. Mais il existe une autre source d'information pour indexer les pairs : le comportement. Ici, il s'agit d'observer quel pair répond aux requêtes de quel autre pair, et d'utiliser cette information pour structurer le réseau logique (on est donc bien en présence d'un système structuré).

On peut citer deux équipes ayant travaillé sur le sujet. Toutes utilisent Gnutella comme système de base. (Sripanidkulchai et al., 2003) est à notre connaissance le premier à proposer de maintenir une file LRU des pairs *utiles*, c'est-à-dire que chaque fois qu'un pair retourne une réponse à une requête, il est ajouté ou déplacé en première place de la file. Si la file est pleine, le dernier pair est supprimé.

(Handurukande et al., 2004) compare LRU avec l'heuristique *history*. Selon cette heuristique, la position d'un pair dans la file égale le nombre de réponses qu'il a fourni sur une fenêtre temporelle - la taille de la fenêtre est un paramètre du système. Les auteurs montrent sur un jeu de données issu d'une capture de traces que cette heuristique permet de gagner jusqu'à 10 points de précision si peu de pairs sont contactés. À partir d'une recherche parcourant une centaine de pairs, cette technique n'apporte plus de réelle plus-value.

Ce dernier article étudie aussi l'impact de la présence de pairs « généreux ». Il est connu que dans les réseaux P2P, un petit nombre de pairs possède des réplicats d'une grande majorité des ressources. Il est possible que le mécanisme que nous étudions, au lieu de structurer les réseaux en rapprochant les pairs selon leurs domaines d'intérêt relie simplement tous les pairs à ce petit nombre de pairs « généreux ». Pour étudier cette question, les auteurs refont les expériences en supprimant de la file les $x\%$ plus généreux pairs ($5 <= x <= 15$). Lorsque la taille de la file est très petite, cette manipulation a un impact très limité sur les performances de la recherche. Cela confirme donc la pertinence de l'approche : la technique est efficace pour clustériser les ressources. Elle montre aussi que les pairs « généreux »sont utiles pour atteindre de forts niveaux de rappel. Les auteurs essaient aussi de propager les requêtes sur 2 sauts. Il en ressort que propager sur deux sauts n'améliore pas les performances de manière significative, donc le réseau est bien structuré.

La possibilité de maintenir plusieurs listes pour différents domaines d'intérêt est peu étudiée. Les auteurs annoncent simplement qu'ils arrivent à gagner 10% de précision en différenciant les fichier audio des fichiers vidéo.

Bilan Ce système peut donc être caractérisé ainsi :

Mécanisme de recherche « sémantique » :

- la recherche n'est pas aléatoire : les pairs sont indexés. Les index décrivent une relation entre le domaine d'expertise d'un pair et le domaine d'intérêt d'un autre.
- les pairs sont indexés dans un fichier réparti, nourri par observation du mécanisme de recherche sous-jacent. Sa cohérence est donc faible.

Mécanisme sous-jacent :

Ce mécanisme ne met pas en place d'index de pair ; la recherche est aléatoire.

2.3.6 Expansion de requêtes dans un réseau de type Gnutella

Nous n'allons pas parler ici d'un système à proprement parler, mais d'un mécanisme qui peut être mis-en-place au-dessus d'un autre système P2P. C'est donc déjà en tant que tel un pas vers l'abstraction des systèmes P2P, que nous abordons dans la suite du document. (Nakauchi et al., 2004) propose un mécanisme pour maintenir des tables d'expansion - *KRDB*, pour *Key Word Data Base* - au-dessus du système Gnutella, par exemple.

Expanser une requête consiste à y ajouter des termes en relation avec les termes initialement entrés par l'utilisateur. Selon les auteurs, ceci est utile dans deux cas : ajouter des synonymes - ajouter *auto* à une requête contenant le mot *voiture* - ou rajouter un terme se situant à un autre niveau de granularité - ajouter *pomme* à une requête contenant le mot *fruit* ; le second terme a un sens plus large que le premier. Les auteurs précisent bien qu'ils étudient un mécanisme simple. Par exemple, pour une requête contenant le mot *disque*, il peut être utile d'ajouter le mot *freinage* ou *musique*. Choisir entre ces deux possibilités n'est pas un problème trivial et n'est pas abordé ici.

Une $KRDB$ est une matrice carrée telle que $KRDB[i][j]$ donne la pertinence de rajouter le mot j à une requête contenant le mot i. Chaque pair maintient une $KRDB$, qui lui permet d'expanser les requêtes qu'il reçoit. Les requêtes sont donc uniquement expansées coté receveur, ce qui évite que le nombre de termes dans la requête explose au cours de la propagation. Les $KRDB$ sont gérées selon cinq règles/mécanismes :

1. *Initialisation* Les $KRDB$ sont initialisées par un mécanisme de recherche de co-occurrences dans le corpus local de chaque pair. Si une co-occurrence est extraite entre les mots i et j, $KRDB[i][j]$ est initialisé à la valeur KR_{init}.
2. *Transitivité* Si $KRDB[i][j] > KR_{seuil}$ et $KRDB[j][k] > KR_{seuil}$, alors $KRDB[i][k]$ est initialisé à la valeur KR_{init}.
3. *Suppression* Si $KRDB[i][j] < KR_{min}$, alors $KRDB[i][j]$ est supprimé.
4. *Feedback* Les $KRDB$ sont mises-à-jour par un mécanisme de Feedback, décrit ci-après.
5. *Synchronisation* Les $KRDB$ sont partagées par un mécanisme de Synchronisation, décrit ci-après.

Le mécanisme d'expansion de requêtes permet d'augmenter le rappel. On prend donc le risque de polluer l'utilisateur avec un grand nombre de résultats peu précis. Pour pallier ce problème, les auteurs proposent de pondérer le rang des réponses par les valeurs des $KRDB[i][j]$ utilisées. Ils proposent une mesure de similarité telle que

$$rang_{reponse} = \sum_{i \in requete, j \in reponse} KRDB[i][j]$$

Notez que dans le cadre du modèle vectoriel, il pourrait simplement s'agir que la coordonnée de la requête dans la dimension du mot b soit égale à $KRDB[a][b]$, où a est le terme de la requête qui a été expansé vers b.

Feedback Le principe de ce premier mécanisme est d'utiliser un retour utilisateur (*relevance feedback*) pour améliorer la précision des $KRDB$. Les auteurs ne précisent pas comment il est obtenu, mais ils supposent qu'on dispose d'un flux d'informations du type « la ressource R a bien répondu à la requête Q ».

À partir de là, on complexifie un peu la structure des $KRDB$. On met-en-place deux informations supplémentaires :

- $UsedCnt[i][j]$ donne le nombre de fois que $KRDB[i][j]$ a été utilisé,
- $HelpfulCnt[i][j]$ donne le nombre de feedbacks positifs reçus sur une réponse obtenue en utilisant $KRDB[i][j]$.

Dès lors, $KRDB[i][j]$ est donné par la formule suivante :

$$KRDB[i][j] = \frac{HelpfulCnt[i][j]}{UsedCnt[i][j]}$$

Cette information de feedback est créée sur le pair initiateur de la requête, et utilisée sur le pair répondant. Il faut donc rajouter un message, dans lequel le pair initiateur envoie au répondant l'information de feedback, après que les réponses ont été reçues.

Synchronisation Ce mécanisme a deux finalités :

- Proposer aux pairs de nouvelles entrées dans leurs $KRDB$
- Partager les informations de feedback. Ceci est surtout utile pour initialiser une nouvelle entrée.

Il se décompose en deux étapes :

Avec qui partager sa *KRDB* ? Chaque pair doit maintenir une liste des N pairs « les plus proches ». Il synchronise périodiquement sa $KRDB$ avec eux. La proximité entre deux pairs égale le nombre d'entrées en commun dans leurs $KRDB$. Les auteurs précisent qu'une mesure plus précise pourrait être utilisée, qui prenne en compte la valeur des entrées communes. Mais dans leur contexte - le partage d'objets multimédia - la mesure (plus simple) qu'ils utilisent est suffisante pour discriminer les pairs. Une mesure plus fine est nécessaire pour des jeux de donées où les ressources sont moins nettement clustérisées naturellement dans les pairs.

Pour trouver les pairs avec qui partager, les auteurs proposent - sans spécifier - plusieurs options :

- broadcast (Gnutella)
- multiple random walks
- transporter les $KRDB$ avec les requêtes

Nous pourrions ajouter à cette liste :

- utiliser un mécanisme de gossiping
- indexer les $KRDB$ dans une DHT (ou autre structure d'indexation distribuée)

Comment synchroniser deux *KRDB* ? On complexifie encore un peu la structure de données, en différenciant les entrées extraites du corpus local (PK, Primary Keyword) des autres (SR, secondary keyword). Seules les entrées de type PK sont synchronisées. Lors d'une synchronisation, les deux $KRDB$ prennent la valeur pour laquelle $UsedCnt$ est le plus grand.

Bilan Si on récapitule les mécanismes P2P utilisés ici, on a :

- Un mécanisme de maintien de cohérence des informations de feedback. Le feedback est produit sur le pair émetteur et est utile au pair répondant.
- Un mécanisme de synchronisation, pour lequel il faut sélectionner les pairs avec qui on se synchronise :
  - la recherche n'est pas aléatoire : les pairs sont indexés. Les index sont des relations binaires entre les pairs ; ils décrivent le nombre d'entrées en commun dans leurs KRDB.
  - les pairs sont indexés dans un fichier réparti. Ce fichier est nourri par un mécanisme de recherche aléatoire de correspondants.
- Mécanisme sous-jacent :
  Ce mécanisme ne met pas en place d'index de pair ; la recherche est aléatoire - par exemple, Gnutella.

2.4 Tableau comparatif des systèmes de notre état de l'art

Nous présentons ci-dessous un tableau comparatif des systèmes décrits dans notre état de l'art. Par anticipation sur le travail exposé dans le reste du document, nous exhibons deux aspects des systèmes : indexation et technologie P2P. Coté indexation, nous présentons ce que les index décrivent - leur sémantique - puis le langage utilisé pour exprimer cette sémantique. Côté technologie P2P, nous distinguons un modèle de répartition et des mécanismes de répartition qui implémentent ce modèle.

On peut tirer quelques enseignements de ce tableau. Tout d'abord, on observe une grande variété de techniques. On remarque que certains systèmes occupent plusieurs colonnes, et que d'autres n'en occupant qu'une seule sont présentés de manière partielle. Il existe donc de grandes différences de complexité entre les systèmes.

Ce tableau met-en-évidence qu'un système P2P naît d'une combinaison de techniques. Même si la séparation des problèmes d'indexation et de technologie P2P permet de mieux cerner l'architecture des systèmes, on voit bien que le panachage des différentes techniques mises-en-évidence (échanger des cases) n'est pas un problème trivial. Il faut pour cela mieux comprendre les interactions entre les cases d'une même colonne - différents aspects d'un même système.

Enfin, ce tableau laisse entrevoir que pour bien évaluer des systèmes P2P, il faudra les découper, pour permettre de comparer leurs performances à un même niveau fonctionnel (par exemple, évaluer colonne par colonne). Symétriquement, un certain nombre de cases comportent la mension « non-spécifié ». Comment alors comparer un *morceau de colonne* à une colonne complète ? Nous verrons qu'on peut comparer les cases d'une même ligne ; mais encore une fois, il faut pour cela bien cerner comment elles interagissent avec leurs voisines verticales.

	Gnutella	Hiérarchique		CAN
		Recherche inter-clusters	Recherche intra-clusters	
Les index décrivent...	Pas d'indexation : Sélection aléatoire	Non spécifié non centralisé	Le corpus local	Le corpus local (après re-clustérisation)
Langage d'indexation			Non spécifié	Espace des clefs
Modèle de répartition	Propagation aléatoire	Non spécifié	Centralisé	Structure d'indexation distribuée
Mécanisme de répartition	Topologie aléatoire + Propagation aléatoire	Non spécifié	Élection d'un super-pair	Modèle *{Join, Leave Takeover, Maintenance}* Propagation guidée, Cohérence forte

	PlanetP	pSearch		*Routing Indices*
		1ère phase de recherche	2nd phase de recherche	
Les index décrivent...	Le corpus local	Le corpus local (après re-clustérisation)	Pour chaque voisin, des ressources et requêtes passées *utiles*	Le corpus d'un sous-réseau
Langage d'indexation	Modèle vectoriel + Filtres de Bloom	Modèle vectoriel		Tableaux de *topics* (apparenté au modèle vectoriel)
Modèle de répartition	Fichier d 'index distribué	Structure d'indexation distribuée	Centralisé	Réseau étiqueté
Mécanisme de répartition	Gossiping (Propagation aléatoire, cohérence faible)	Mécanismes CAN (propagation guidée, cohérence forte)	Algorithme distribué entre 2 pairs	Calcul distribué

	SON	Indexation à partir du comportement	*KRDB Synchronisation*
Les index décrivent...	Le corpus local	Relation domaine d'intérêt/domaine d'expertise	Le nombre d'entrées en commun dans les KRDB
Langage d'indexation	Ensemble de concepts de la hiérarchie de classification	Classement en 2 types de pairs (proches ou non)	Un entier
Modèle de répartition	Non spécifié	Mixte Fichier d'index distribué, Propagation aléatoire	Propagation aux plus proches Cohérence faible
Mécanisme de répartition		Propagation aléatoire + Tri local	Non spécifié (ex : propagation avec requêtes)

FIG. 2.11 – Tableau comparatif des systèmes de notre état de l'art.

2.5 Éléments de méta-modèle des systèmes P2P

Nous ne sommes pas les premiers à soulever le problème de l'évaluation des systèmes P2P. Nous en tenons pour preuve les nombreuses études qui ont été menées pour proposer un standard d'évaluation. (Schmitz and Löser, 2006) propose une synthèse des travaux sur cette question. Il fait un tour d'horizon des principales techniques P2P, il propose les formats d'entrée et de sortie d'un simulateur idéal et donne un état de l'art des jeux de données. Cet article est caractéristique d'une approche applicative des systèmes P2P ; le système y est vu comme une boîte noire qui remplit une fonctionnalité. Dans cette approche, le jeu de données et la simulation ont une place prépondérante : on simule le système sur un jeu de données.

Nous verrons cette approche dans une première section. Nous étudierons ensuite le modèle des systèmes DHT proposé par (Ratajczak and Hellerstein, 2003). Ce modèle est une approche mathématique de la topologie des systèmes structurés. Mais s'il est efficace pour abstraire les techniques P2P, il n'aborde pas la question de leur articulation avec les problématiques applicatives.

Il existe un troisième type d'approche, qui consiste à centrer l'étude sur les techniques mises en œuvre. Dans ce cas, le problème est approché comme un ensemble de techniques qu'on peut combiner et adapter pour remplir une (ou plusieurs) fonctionnalités. À notre connaissance, on ne trouve pas de réelle étude sur le sujet, mais on voit régulièrement des idées transparaître dans des articles, à l'occasion par exemple de la présentation d'un système où d'une étude. Nous avons choisi de présenter en fin de section une étude de la vision du domaine qui transparaît dans (Yang and Garcia-Molina, 2002) et (Beaumont et al., 2007).

2.5.1 Approche en couches

(Aberer et al., 2004) propose un modèle en cinq couches. Nous synthétisons cette approche en donnant les fonctionnalités offertes par chaque couche et en montrant comment les auteurs conçoivent leur implémentation.

Couche 1 : Transport Layer Envoyer un message.

C'est la couche haute du modèle Internet habituel. Elle implémente les protocoles TCP/IP et UDP/IP.

Couche 2 : Structured Overlay Network Retrouver l'adresse physique d'un pair/ d'une ressource à partir de son identifiant.

Cette fonctionnalié est assurée en indexant les pairs dans une DHT (cf. section 2.2.3). Les auteurs précisent que cette fonctionnalité doit être accessible en contactant n'importe lequel des pairs qui forment la DHT : « The service can be requested from each peer of the group in a uniform way ».

Couche 3 : Document and Content Management Placer les documents sur les pairs, répartir la charge.

Par rapport à une DHT classique, les auteurs proposent d'externaliser la fonctionnalité de placement des ressources sur les pairs dans une couche de plus haut niveau. Cette couche gère l'attibution des clefs aux ressources/pairs ; a priori ces clefs correspondent aux index calculés par la couche 4. Cela signifie aussi que cette troisième couche gère l'attribution d'une portion de l'espace d'adressage à chaque pair. Enfin, cela permet de faire de la répartition de charge.

Couche 4 : Retrieval Model Résoudre les requêtes ; interface Recherche d'Information.

Cette couche implémente le modèle de Recherche d'Information. Elle indexe les requêtes et ressources, elle effectue l'appariement requêtes/ressources.

On remarque que les auteurs ne donnent pas à la couche 4 la possibilité de calculer les index de pair. Pourtant, tous les systèmes RI de notre état de l'art attribuent des index « sémantiques » aux pairs. On peut cependant leur concéder qu'ils précisent bien que leur modèle peut être augmenté et doit être l'objet d'une procédure de standardisation.

Il nous semble aussi important de remarquer que l'implémentation de la couche 2 n'est pas une DHT, mais seulement le réseau logique d'une DHT (*overlay*). La transformation la plus importante est que l'on retire la fonctionnalité d'attribution des clefs aux ressources et pairs. En conséquence, sur la couche 2 on maintient un réseau logique sans en connaître les spécifications. Surtout, on ne maîtrise pas le langage des index, donc on ne maîtrise ni la dimensionnalité ni la topologie de l'espace des index. Nous pensons que la faisabilité de la séparation en couches à ce niveau mériterait d'être étudiée plus avant, peut-être une approche mathématique telle celle étudiée dans la prochaine section aiderait en ce sens.

Plus embêtant, tous les systèmes de Recherche d'Information de notre état de l'art mettent cette architecture en défaut (excepté les DHT). Cette architecture est clairement conçue pour intégrer des systèmes structurés, ce qui exclut de fait tous les systèmes sauf pSearch ; mais celui-ci ne « rentre pas » non plus. Regardons simplement le mécanisme des *sample sets*. Leur construction et utilisation sont fortement dépendants de la topologie logique (topologie de l'overlay). Les *sample sets* forment un système d'indexation parallèle aux index utilisés comme clefs sur la DHT. Leur calcul est complètement une problématique RI (couche 4), qui nécessite d'accéder aux tables de routage (couche 2), et d'effectuer des échanges entre voisins (couche 1). Les *sample sets* font voler en éclats la séparation en couches proposée. Ceci est l'exemple le plus évident, mais globalement, pSearch ne cadre pas avec cette architecture. (Li et al., 2004), système dérivé de pSearch, n'y correspond pas non plus.

À l'origine de ce manque de généralité du modèle, nous avons repéré deux éléments. Le premier, c'est que cette architecture a clairement été destinée à modéliser des sys-

tèmes structurés (utilisant des réseaux logiques dérivés de DHT). Le second, c'est que les auteurs voient le réseau P2P comme un espace de stockage utilisé par l'application. Or, ce n'est pas toujours le cas. Parfois, le réseau P2P est le lieu de la résolution des requêtes. Typiquement dans pSearch, propager une requête, c'est la résoudre. Le réseau se comporte comme une structure de données ; propager une requête, c'est parcourir cette structure de données pour résoudre un problème. On n'est plus en présence d'une problématique d'accès/stockage de données, mais d'un algorithme distribué.

Avant de finir cette section, citons (Lua et al., 2005) qui propose une approche en cinq couches. Le modêle est un peu plus complexe. Il inclut les notions de sécurité et fiablité ; il permet de distinguer/implémenter plusieurs types de services et ne se réduit pas à l'application *Recherche d'Information*. Cependant, il ne comble pas les deux lacunes que nos reprochons à son prédécesseur : il est clairement orienté systèmes structurés et le réseau P2P se limite à un espace de stockage.

2.5.2 Un modèle des systèmes DHT

2.5.2.1 Description

(Ratajczak and Hellerstein, 2003) est certainement le modèle des systèmes DHT le plus abouti. Il ne se place pas au même niveau que les modèles étudiés précédemment. Son objet n'est pas de comprendre le lien entre l'application et le réseau P2P, mais de proposer un méta-modèle mathématique des structures de réseau logique. Pour cela, les auteurs proposent de modéliser des topologies réseau à l'aide de graphes de Cayley (théorie des groupes). Le dual de ce modèle mathématique, c'est l'« emulation scheme », technique de mise-en-place de cette structure sur le réseau.

Les graphes de Cayley sont sommet-transitifs (*vertex-symmetric*), c'est-à-dire que n'importe quelle permutation des nœuds est homomorphe au graphe originel. Visuellement, si on trace le graphe du réseau à partir de la donnée des relations de voisinage (tables de routage), on peut échanger les noms des sommets comme on veut (sur le même dessin), le graphe obtenu sera toujours conforme aux tables de routage. Plus pragmatiquement, d'un point-de-vue P2P, cela permet que l'algorithme de routage ait le même comportement, quel que soit son point de départ. Donc tout pair peut être pris comme point de départ d'une recherche, cette recherche aura le même comportement, même coût, même qualité - on parle qualité et coût moyens, minimaux ou maximaux.

Pour compléter cette brève description de la théorie des graphes de Cayley, ils reposent sur un couple (G, S), où G est un groupe et S un ensemble d'éléments de G. Grossièrement, G donne l'ensemble des sommets et le sous-groupe de G généré par S donne les arêtes. Pour un ensemble de sommets G, on peut construire plusieurs topologies, en choisissant plusieurs ensembles S.

Le modèle mathématique utilisant les graphes de Cayley permet d'étudier l'efficacité d'une structure de réseau : répartition de charge, efficacité de la recherche ; par exemple parce qu'il permet de mesurer le diamètre du réseau. Le problème, c'est que dans un système P2P ces structures sont difficiles à mettre en œuvre. On a par exemple des contraintes sur le nombre de nœuds dans la structure ; du type « le nombre de nœuds doit être une puissance de 2 ». Il faut aussi gérer la dynamicité du réseau : des pairs joignent et quittent le réseau en permanence. Cela impose que la construction d'un réseau P2P repose sur des fonctions *join* et *leave* (au minimum), parce que ces réseaux se construisent et évoluent par entrées et sorties de pairs du réseau. Il faut faire la correspondance entre cette structure de l'algorithme de construction et le modèle mathématique que nous venons de voir : c'est le rôle de l'« emulation scheme ».

Les auteurs recensent trois types d'émulation schemes. Grossièrement, le premier consiste à considérer que les pairs sont répartis dans un espace (ex. espace vectoriel, comme pour Chord et CAN). Les liens entre les pairs correspondent aux relations de voisinage dans cet espace. Équilibrer la structure, c'est équilibrer la répartition des pairs dans l'espace. Pour le second, les pairs sont les feuilles d'un arbre (typiquement, l'arbre des partitionnements de l'espace sous-jacent, comme pour pSearch). Équilibrer la structure, c'est équilibrer l'arbre. Le troisième est basé sur une estimation de la taille du réseau. Chaque pair possède sa propre estimation n de la taille du réseau et met en place trois tables de routage correspondant aux hypothèses que le réseau est de taille n, $n-1$ et $n+1$. On aura donc au moins trois réseaux imparfaits qui vont co-exister pour couvrir l'ensemble des pairs. Les requêtes vont être routées au mieux, à cheval sur ces trois réseaux.

2.5.2.2 Remarques

Pour être vraiment complet, il faudrait ajouter une troisième dimension (peut-être une troisième couche) en plus du modèle mathématique et de l'émulation scheme. En effet, il manque les aspects programmation répartie. Par exemple, les pairs (Stoica et al., 2001) recherchent régulièrement leur prédécesseur, pour vérifier que la structure ne contient pas de « trou ». Lorsqu'une telle irrégularité est détectée, un algorithme de réparation est appliqué. (Tang et al., 2003) fonctionne différemment, avec un algorithme de réparation rapide de la structure en cas de défaut de pair (*takeover*) - qui ne répare jamais la structure de façon optimale - et un algorithme de rééquilibrage qui tourne en tâche de fond et répare les défauts créés par *takeover*. Il est certain cependant que l'efficacité d'un choix d'implémentation est très lié au type d'« émulation scheme »implémenté.

Cette proposition va clairement dans le sens d'une étude séparée de la structure mathématique (aspect applicatif) et des mécanismes de gestion répartie de cette structure (aspect réseau). L'inconvénient, c'est qu'elle convient très bien pour étudier les systèmes DHT - dans le sens solution de stockage d'informations - mais n'est pas générique

à tous les types d'application. En Recherche d'Information, par exemple, on ne connaît pas a priori le vocabulaire employé par les utilisateurs. Le vocabulaire est celui de la collection, de l'ensemble de documents, et sera connu à l'exécution, et non lors de la conception du système. Du coup, la structure de données sous-jacente n'est pas connue a priori : elle est mise-en-place dynamiquement ; et le modèle de (Ratajczak and Hellerstein, 2003) n'est pas applicable.

2.5.3 Approche centrée mécanismes : Quelques éléments tirés de la littérature

Nous voulons ici donner quelques éléments intéressants d'abstraction - voir de métamodèle - des systèmes P2P trouvés dans des articles dont le but n'est pas directement la réflexion sur ce sujet. Ce qui nous semble original ici par rapport aux travaux précédents, c'est que plutôt que de se concentrer sur les fonctionnalités, ce qui conduit à un découpage en couches fonctionnelles ; les auteurs se concentrent sur les mécanismes mis en place et s'appuient sur leur analyse pour essayer de structurer le domaine des systèmes P2P. Nous pensons que cette approche est plus efficace que la première.

2.5.3.1 (Yang and Garcia-Molina, 2002)

Cet article étudie trois techniques pour optimiser la propagation des requêtes dans les réseaux P2P « purs » :

- *Iterative Deepening* : si suite à une première recherche les résultats sont insatisfaisants, « pousser » la requête, pour élargir la recherche à une profondeur plus importante. On peut ainsi pousser la requête récursivement plusieurs fois. On peut avoir des heuristiques complexes, par exemple pousser les requêtes deux sauts par deux sauts, ou suivant une fonction logarithmique...
- *Directed BFS* : sélectionner les voisins vers qui propager les requêtes. Cela suppose qu'on maintienne des informations sur les voisins. Les auteurs proposent plusieurs types d'informations :
  - nombre de résultats retournés
  - coût des résultats retournés (ex. nombre moyen de sauts pour les trouver)
  - taille de la file de requêtes - requêtes qu'on lui a envoyé auxquelles il n'a pas encore renvoyé de réponse ; c'est un indicateur de charge.

  Ces informations sur les pairs peuvent être calculées :
  - dans l'absolu, par exemple *nombre de résultats retournés depuis qu'il est notre voisin*
  - par requête, par exemple *nombre moyen de résultats retournés par requête*

  - sur une fenêtre temporelle, par exemple *nombre de résultats retournés durant les m dernières minutes*

- *Local Indices* : réplication. Le principe est que chaque pair maintienne des index des ressources de ses voisins à s sauts. Vu autrement, les descripteurs de ressources sont répliqués sur une boule de taille s. Cela permet d'abaisser les coûts de recherche parce qu'on interroge moins de voisins pour obtenir le même nombre de réponses ; au prix d'une augmentation des coûts de stockage.

Au travers de cette étude, les auteurs donnent plusieurs éléments d'abstraction des systèmes P2P. Tout d'abord, ils partagent les systèmes en deux classes, suivant le niveau de qualité de service fourni. Grossièrement, ils classent d'un coté les systèmes P2P « purs »(Gnutella, Freenet), de l'autre les systèmes structurés (DHT). Selon eux, l'utilisation de la première classe de sysèmes est justifiée par un faible niveau d'exigence de qualité de service : « persistence and availability not guaranted or necessary ». Nous pensons qu'il est possible d'aller plus loin dans ce sens, en considérant que l'objectif de l'évaluation d'un système est de le positionner dans un espace des critères de qualité de service.

Ensuite, leur étude même repose sur le principe qu'on peut incorporer ces trois techniques à des systèmes existants. C'est un effort d'abstraction vers une meilleure modularité des systèmes P2P. Selon nous, cependant, lorsqu'on applique la technique *Directed BFS* on sort du cadre des systèmes P2P « purs », car on met-en-place des index de pairs, ce qui implique que la recherche n'est plus aléatoire. Néanmoins, on n'est clairement pas en présence d'un système structuré. Il y a un travail d'approfondissement à mener dans ce sens, pour mieux comprendre quelles sont les différentes classes de systèmes, et par quelles modifications on passe d'une classe à l'autre.

Enfin, les auteurs listent les paramètres qui vont être mesurés - les mêmes pour tous les systèmes. Les résultats d'évaluation obtenus permettent donc de comparer les systèmes entre eux. C'est un effort significatif de standardisation du processus d'évaluation. Rien n'indique cependant qu'il sera compatible avec les choix opérés par d'autres chercheurs du domaine.

Nous conclurons en disant qu'on a ici trois pistes intéressantes d'abstraction et de compréhension des systèmes P2P, qui sont clairement des pas vers la construction d'un métamodèle et d'un processus d'évaluation standardisé. Nous ferons cependant un reproche symétrique à celui fait aux deux dernières sections : cette fois, les éléments de modèle ne concernent que les systèmes non-structurés ; rien n'est prévu pour intégrer les systèmes DHT. Pourtant, une recherche itérative, par exemple, est envisageable dans un tel système.

2.5.3.2 (Beaumont et al., 2007)

Dans cet article, les auteurs développent une vision originale des systèmes P2P, que nous détaillons plus loin. Cette vision conduit à l'idée de maintenir un système structuré par une approche de type Monte-Carlo, par opposition aux approches traditionnelles où le réseau logique est maintenu de façon forte par des algorithmes déterministes. Ici, un mécanisme de gossiping est utilisé comme source aléatoire de pairs. Chaque pair sélectionne localement ses voisins à partir de cette source, pour construire son voisinage. L'essentiel de l'article est consacré à étudier la pertinence de cette approche, en exposant une proposition de système fonctionnant sur ces principes, puis en évaluant ses performances. Ce qui nous intéresse avant tout ici, c'est la vision du domaine développée par les auteurs.

En amont de leur démarche, les auteurs proposent d'abstraire les différents types d'application par le type de requête manipulé. Ils différencient trois catégories de langages de requêtes :

1. « exact search is used to access data objects identified by a unique identifier ;
2. attribute-based search enables to access data using a set of attribute, value pairs ;
3. in range queries, the attribute values are specified for a given range »

Le principe est que le langage de requêtes conditionne la topologie du réseau logique : « We argue that, in order to improve upon the efficiency of expressive queries, the structure of the peer to peer overlay should reflect the application's one. Peers are then application objects and get connected to neighbours (i.e. sharing similar characteristics from the application point of view) ».

SI l'on suit leur démarche, on en vient à définir une vision applicative du système, clairement distincte de son implémentation P2P. Cela sous-entend l'existence d'une structure au niveau applicatif - conceptuellement, du moins - qui est implémentée sur le réseau P2P. Nous reviendrons sur cette idée dans la suite du document. Nous retenons également l'idée que les pairs sont des objets applicatifs (« application objects »). Cependant, nous ne parlerons pas de *pair* mais de *cluster*. En effet, nous pensons que ce qui caractérise un pair au niveau applicatif, ce sont les requêtes qu'il pose et les ressources qu'il publie. Considérer un pair au niveau applicatif comme un *groupe* de ressources et de requêtes est donc à notre avis un niveau d'abstraction utile et suffisant.

Ainsi, le système P2P peut être une implémentation plus ou moins fidèle de l'application, dessinant un compromis entre les contraintes applicatives et les contraintes liées au caractère réparti de l'application : « The exact logical structure is not as crucial, provided that its estimation enables correct routing for all requests ».

Si on résume cette approche, la conception d'un système P2P (structuré) serait un processus en trois étapes :

- concevoir la structure de données utile d'un point de vue applicatif
- concevoir la structure du réseau logique (overlay)
- choisir les mécanismes P2P qui l'implémentent

La structure du réseau logique est un compromis entre les contraintes apportées par l'application et par les mécanismes P2P utilisés. En l'occurence, les auteurs construisent une « estimation » de la structure applicative.

Nous pensons qu'il est possible d'adapter cette approche pour l'élargir à d'autres classes de solutions que les systèmes structurés. Nous verrons dans un prochain chapitre comment nous analysons PlanetP (qui n'est pas un système structuré) avec une approche similaire.

2.6 Synthèse

Au terme de cette étude bibliographique, nous pouvons tirer un certain nombre de conclusion sur ses apports à notre problématique.

L'étude des principaux types de systèmes P2P nous a montré qu'il s'agit de systèmes complexes. Comparer leurs architectures n'est pas un problème trivial. Sans cela, on se trouve vite démunis lorsqu'on veut faire du (re-)design. Par exemple, on n'a pas de vision claire de la façon dont on adapte une technique d'un système sur un autre ; on ne sait pas anticiper l'impact d'un changement dans les mécanismes d'indexation ou les technologies P2P utilisées.

Selon nous l'évaluation des systèmes P2P doit être orientée design. C'est-à-dire que le but d'une étude de performances est selon nous de savoir quelles techniques marchent, dans quel contexte, et de donner des clefs pour comprendre dans quelle direction orienter le (re-)design d'un système, suivant les connaissances qu'on a du contexte dans lequel il est utilisé - type de jeu de données, comportement des utilisateurs... Là encore, la mauvaise compréhension de l'architecture des systèmes P2P est un frein considérable - nou y revenons dans le chapitre suivant.

Nous avons donc prolongé notre état de l'art par une étude des méta-modèles et autres éléments de compréhension proposés par la littérature. Nous y distinguons deux types d'approches. La première se concentre sur les fonctionnalités et aboutit à un découpage en couches ; la seconde se concentre sur les mécanismes. Nous pensons que cette seconde, bien que plus complexe à mettre en œuvre, est préférable à l'approche en couches, et ce pour trois raisons principales :

- Tout d'abord, les systèmes P2P sont des systèmes complexes, composés de nombreux éléments ou mécanismes. L'approche en couches évalue le système dans sa globalité et ne permet pas de regarder en détails l'apport de chaque composante.
- Ensuite, un même mécanisme peut servir à implémenter différentes applications/ fonctionnalités. Suivant l'approche en couches, pour évaluer une technique P2P sur plusieurs applications, il faut, pour chacune, construire ou capturer un nouveau jeu de données, implémenter le système et le simuler. Les possibilités de réutilisation des résultats sont donc réduites.
  À titre d'exemple, un jeu de données tiré d'un système de partage de fichiers actuel est inadapté à l'évaluation d'un système de Recherche d'Information en P2P. En effet, en Recherche d'Information, une problématique essentielle est de définir le besoin de l'utilisateur, généralement à partir d'une requête. En partage de fichiers, lorsqu'un utilisateur pose une requête *« Bob Marley »*, son besoin est clair. De fait, avec une étude suivant le modèle en couches, les connaissances sur

les mécanismes P2P acquises par l'étude des systèmes de partage de fichiers sont difficilement réutilisables dans un cadre Recherche d'Information.
- Remarquons également que la qualité d'un système exerce une rétro-action sur le jeu de données. Par exemple, aujourd'hui sur les systèmes de partage de fichiers en P2P, on ne trouve presque pas de musique classique. Si demain un système permet de partager efficacement ce type de fichiers rares, les utilisateurs pourront les échanger entre eux, ce qui mécaniquement fera augmenter le nombre de ces fichiers dans le système. La qualité du système modifie donc les propriétés du jeu de données. Cela implique un décalage entre les performances obtenues lors de l'évaluation d'un système sur une capture de données, et ses performances une fois déployé.
- Notons enfin que le travail de compréhension nécessaire pour une analyse centrée mécanismes est utile pour aider à la conception des systèmes P2P.

Plus précisément, notre parti est de considérer qu'un système P2P est une implémentation répartie d'une application. Pour implémenter une application dans un réseau P2P, il faut :
- partager des données
- répartir des calculs

Dans les chapitres 5.1 et 5.3, nous montrons sur deux exemples comment des systèmes RI-P2P peuvent être abstraits comme des spécifications applicatives, pour lesquelles ont été faits différents choix d'implémentation, que nous détaillons jusqu'au niveau mécanisme de partage de données et algorithme distribué. Ce travail nous permet de proposer une procédure originale d'évaluation des systèmes P2P, ainsi que d'identifier précisément un ensemble de techniques de répartition en P2P. Nous détaillons ces techniques dans le chapitre 6.

Finalement, notre approche est très proche de la vision éveloppée dans (Beaumont et al., 2007). En implémentant un réseau structuré (structure de type *Voronoi tesselation*) en utilisant un mécanisme de gossiping, les auteurs font en quelques sortes un mélange entre PlanetP et un système structuré. L'idée qui permet ce « mélange », c'est que le mécanisme P2P de Gossiping peut-être utilisé pour maintenir un fichier distribué des pairs, ou pour maintenir la structure d'un réseau logique (overlay). C'est une première étape de découplage entre le mécanisme P2P et ce à quoi il est utilisé. Dans cette thèse, nous allons plus loin, en analysant ce que sont les mécanismes P2P et comment ils peuvent être utilisés pour implémenter une application dans un cadre P2P. Bien sûr, le choix du mécanisme utilisé implique aussi une contrainte sur le type de qualité de service qui pourra être rendu. Les différents niveaux de qualité de service atteignables par différents modèles de répartition doivent être mis en rapport avec les différents niveaux de qualité de service exigibles par différents types d'applications.

Dans le chapitre suivant, nous faisons une étude poussée de l'évaluation des systèmes PlanetP et pSearch. Cela nous permet de bien cerner les limites des procédures d'éval-

uation actuelles. Ensuite, le chapitre 4 pose la vision originale du domaine que nous avons développée. Nous montrons dans les deux derniers chapitres comment nos utilisons cette vision, pour évaluer des systèmes P2P (section 5) et pour leur conception (section 6).

Chapitre 3

Évaluation de systèmes P2P : le cas de PlanetP et pSearch

Nous reprenons dans ce chapitre l'exemple des systèmes PlanetP et pSearch. Nous avons choisi l'exemple de ces systèmes parce qu'ils sont très différents. En effet, par rapport à l'ensemble des systèmes de notre état de l'art, PlanetP fonctionne sur la base d'un principe relativement simple, tandis que pSearch est l'un des systèmes les plus complexes. Par ailleurs, ils sont décrits précisément et leurs concepteurs en font une évaluation poussée, fournissant ainsi la matière nécessaire pour l'étude que nous en faisons. Nous étudions dans ce chapitre la façon dont leurs concepteurs ont mené leur évaluation, comme préalable à notre propre processus d'évaluation, présenté dans les chapitres 4 et 5.

Nous étudions donc l'évaluation des systèmes PlanetP et pSearch menée par leurs concepteurs. Nous analysons les moyens mis en œuvre pour l'évaluation, les grandeurs mesurées et les courbes qui en sont tirées. Nous verrons qu'il est difficile de tirer des conclusions de ces évaluations. Cette étude met en évidence à partir de deux exemples que :

1. Il n'existe pas de manière standard d'évaluer des systèmes P2P. Il n'y a pas consensus sur les moyens de l'évaluation (simulation, analyse, implémentation centralisée...), les paramètres dont il faut mesurer l'influence, le type de jeu de données à utiliser, ni même les courbes à produire en sortie.
2. Les résultats obtenus sont difficilement interprétables. Les résultats d'évaluation doivent permettre de mieux comprendre le système, son comportement et l'influence de ses différentes composantes. Cela n'est pas du tout évident ici. De plus, on ne peut pas comparer les performances des deux systèmes. Nous verrons notamment que les courbes restituées au terme des deux évaluations ne sont pas les mêmes.

3. Enfin, nous poserons la question de la réutilisation des résultats. Les évaluations proposées ne réutilisent pas de résultats précédents et les résultats obtenus ne sont pas prévus pour être réutilisés. Les résultats doivent aussi être ré-injectables dans un processus de conception : guider la conception d'un système par la connaissance des performances des techniques P2P en bibliothèque.

3.1 Étude critique des résultats d'évaluation de PlanetP

Cette section étudie les résultats de l'évaluation du système de Recherche d'Information en pair-à-pair (RI-P2P) PlanetP, menée dans (Cuenca-Acuna et al., 2003). Nous n'étudions donc ni l'article dans son ensemble, ni le système dans son ensemble. Nous étudions essentiellement les données fournies dans la section *4 Performance*. Nous souhaitons faire le point sur la connaissance du système PlanetP que nous apporte cette évaluation ; nous souhaitons en saisir la portée et en situer les limites.

L'évaluation est divisée en trois parties : la partie *4.1 Search Efficacy* évalue la qualité des résultats retournés par le système, la partie *4.2 Storage cost* évalue les coûts de stockage engendrés par le système et la partie *Gossiping Performance* évalue la qualité du mécanisme de gossiping.

3.1.1 *Search Efficacy* : Description des expériences

Le protocole expérimental La démarche des auteurs est de comparer les résultats retournés par PlanetP aux résultats retournés par une implémentation centralisée du modèle LSI. Pour cela, ils utilisent des jeux de données fournissant chacun un jeu de ressources documentaires, un jeu de requêtes, et un oracle qui définit quelles ressources répondent à quelles requêtes. Toutes les requêtes sont lancées sur une implémentation centralisée de LSI et sur une implémentation de PlanetP sur simulateur. On compare ensuite la qualité des résultats délivrés par les deux systèmes. PlanetP et LSI sont donc évalués en parallèle par rapport à un même oracle. Le simulateur est validé par comparaison avec un prototype : 200 pairs déployés sur 8 PCs.

Le jeu de données Les auteurs disposent de cinq jeux de données : CACAM, MED, CRAN, CISI et AP89. Les quatre premiers ont été collectés et utilisés par (Buckley, 1985), le cinquième provient de la collection TREC. Il s'agit de jeux de données construits pour évaluer des systèmes RI classiques, qui ne précisent donc pas la répartition des données sur les pairs. Pour pallier ce problème, les auteurs proposent de construire des distributions artificielles. La première est une distribution uniforme des ressources sur les pairs - en nombre de documents par pair. La seconde est une distribution en loi de Weibull, elle est prévue et calibrée pour être compatible avec les mesures effectuées par (Saroiu et al., 2002) et avec des mesures effectuées par eux mêmes sur une communauté P2P d'étudiants.

Les paramètres de qualité mesurés Pour chaque requête, pour chaque jeu de données, pour chacun des deux systèmes PlanetP et LSI, les auteurs calculent le Rappel (R) et la Précision (P) des résultats obtenus. R mesure la capacité du système à retourner toutes les bonnes réponses, P mesure la capacité du système à ne pas retourner

de mauvaise réponse. La notion de bonne réponse est définie par l'oracle et fait partie du jeu de données. Les auteurs en tirent quatre courbes :

- *figure 1 (a) : Rappel moyen sur l'ensemble des requêtes, en fonction de k*
- *figure 1 (b) : Précision moyenne sur l'ensemble des requêtes, en fonction de k*
- *figure 1 (c) : Rappel moyen en fonction de l'échelle (nombre de pairs)*
- *figure 2 : nombre moyen de pairs contactés en fonction de k*

k est un paramètre de PlanetP, il désigne le nombre de réponses attendues par l'utilisateur et conditionne l'arrêt de la recherche : « Stop contacting peers when the documents identified by p consecutive peers fail to contribute to the top k ranked documents ».

3.1.2 *Search Efficacy* : Analyse critique des résultats

Le jeu de données Lors de l'évaluation, les performances mesurées se révèlent plutôt indépendantes du type de distribution quantitative - répartition des ressources sur les pairs sans considérations sémantiques. On comprend donc bien pourquoi les auteurs n'ont pas jugé utile de définir une plage de distributions ni plus précise, ni plus large - plus ou moins Weibull, plus ou moins zipfienne... Par contre, on peut regretter que les auteurs n'étudient pas l'impact de la répartition sémantique des ressources ; à savoir si les pairs ont des domaines d'intérêt plus ou moins ciblés. Intuitivement, les propriétés de répartition sémantique des ressources ont nécessairement un impact sur la précision des filtres de Bloom, dans la mesure où si un pair a un domaine d'intérêt ciblé, cela diminue le rapport $\frac{nombre\ de\ termes}{nombre\ de\ documents}$ sur ce pair. Il y même un réel enjeu derrière cette problématique : l'approche PlanetP fonctionne si les domaines d'expertise des pairs sont suffisamment ciblés, permettant ainsi de discriminer les pairs susceptibles de bien répondre pour chaque requête. À la limite, le jeu de données utilisé n'est pas pertinent pour évaluer ce système.

Critères de qualité En RI on a l'habitude de construire une courbe montrant la précision en fonction du rappel. Cette courbe permet d'étudier efficacement la qualité d'un ensemble trié de réponses - cf. section 2.1.2. Ici, les auteurs ne présentent pas une telle courbe, et on ne peut pas la calculer à partir des mesures présentées. En fait, les auteurs ont omis de mesurer les éléments nécessaires pour construire les courbes *précision/rappel*.

Ensuite, les auteurs présentent uniquement des résultats moyens. Ceux-ci ne permettent pas de se rendre compte des précision et rappel au pire cas. Ils ne permettent pas non plus de se rendre compte de la variabilité des résultats, de savoir si le système permet d'obtenir des précision et rappel proches des valeurs moyennes dans la plupart des cas ; ou si les précision et rappel divergent beaucoup d'une requête sur l'autre. Pour cela, deux techniques connues sont de présenter des barres d'erreur ou de présenter la valeur minimum et l'écart type.

Nous remarquons également que les auteurs n'étudient pas le problème du temps de réponse. C'est pourtant un marqueur important de la qualité de service rendue à l'utilisateur.

Nous souhaitons également dire un mot sur l'idée d'indexer les pairs suivant 30% des termes. À l'évidence, cette « astuce » permet d'économiser beaucoup de ressources - la taille des filtres de Bloom est divisée par trois - et a un impact limité sur la qualité des résultats - on perd 5 points de précision. Cependant, on peut envisager son évaluation sous un angle différent. Il s'agit en fait d'une technique de Recherche d'Information, qui permet de modifier le rapport *qualité/coût* de la résolution des requêtes sur l'ensemble des pairs. Bien d'autres ajustements de ce type sont possibles ou seront inventés. Il serait intéressant de découpler l'évaluation de ces techniques de l'évaluation de PlanetP. Pour cela, il faut mesurer l'impact d'une modification du rapport *qualité/coût* du modèle RI de résolution des pairs sur l'efficacité de PlanetP.

Nous finissons avec deux remarques sur les conclusions avancées par les auteurs quand à l'étude des résultats d'évaluation :

- *« PlanetP tracks the performance of the centralized implementation closely »*. PlanetP, c'est deux mécanismes principaux : la compression de l'index des pairs par la technique des filtres de Bloom et leur maintenance par gossiping. D'après les auteurs, on est en moyenne 11% moins efficace que le système RI centralisé LSI. Si on veut essayer de réduire cet écart, on a besoin de connaître plus précisément dans quelle mesure chacun des deux mécanismes en jeu contribue à cette perte de performances.
- *« PlanetP's performance is independent of how the shared documents are distributed »*. C'est un raccourci un peu rapide, parce que pour obtenir ces performances similaires on génère des coûts très différents - cf. figure 2 de l'article, non reproduite ici, qui présente le nombre de pairs contactés, donc le nombre de messages émis, pour plusieurs types de distributions des données. En réalité, à coût constant, on a de très grosses différences de performances suivant le type de distribution. PlanetP est plus efficace si les ressources sont distribuées de façon Weibull que si elles sont distribuées uniformément. Il y a pourtant une phrase des auteurs qui va dans le sens de notre remarque : *« PlanetP's adaptive stopping heuristic is critical to its performance »*. On peut aussi remarquer qu'analyser deux types de distributions est un peu court pour tirer une telle conclusion.

3.1.3 *Storage cost*

Pour étudier les coûts de stockage, les auteurs calculent la taille de l'index des pairs. Ils proposent d'étudier un scénario pire cas. Ils utilisent la collection TREC, un corpus documentaire au fort ratio *nombre de termes uniques/taille du corpus* ; et ils étudient une distribution uniforme, qui - selon eux et en toute logique - maximise le nombre de termes par pair.

Les auteurs mesurent le nombre moyen de termes par pair et la taille de l'index global (index des pairs), en faisant varier :

- *l'échelle :* 1000 ou 5000 pairs
- *le taux de réplication des données :* 1 ou 3 fois
- *le taux d'indexation :* sur 100% ou 30% des termes

Prendre uniquement deux valeurs pour chaque paramètre d'évaluation permet difficilement de tirer des conclusions. Cependant, la taille de l'index semble grandir moins que linéairement avec le nombre de pairs ou avec le taux de réplication ; et il apparaît qu'indexer selon les 30% de termes les plus caractéristiques peut influer fortement à la baisse la taille de l'index. Dans tous les cas, la taille de l'index reste inférieure à 70MO.

Outre le nombre de valeurs pour chaque paramètre, nous ferons la remarque que le modèle de distribution des ressources semble un peu simple. Cela rejoint une remarque faite au paragraphe précédent : quid de l'impact d'une distribution « sémantique » des ressources sur la taille de l'index global ? Les auteurs étudient la taille de l'index si les ressources sont répliquées 1 ou 3 fois. Or dans un réseau P2P, certaines ressources sont extrêmement populaires (donc fortement répliquées), d'autres sont très rares. Pour inclure cette donnée, il faut introduire dans le modèle de ressources la notion de distribution du taux de réplication des ressources.

3.1.4 Le gossiping

Première série de tests : propagation d'une nouvelle Les auteurs commencent par mesurer le temps et la bande-passante moyenne consommée pour propager un filtre de Bloom de 1000 termes en fonction de l'échelle. Nous interprétons cette étude comme une validation du mécanisme *partial anti-entropy*. En effet, pourquoi veut-on étudier le mécanisme de gossiping ? Parce qu'il impacte la qualité des résultats et le coût pour les obtenir. Comment est-ce qu'il impacte la qualité des résultats ? C'est lui qui maintient la cohérence des index globaux. Donc in fine, on a besoin d'étudier le mécanisme de gossiping pour connaître les rapports qualité/coût de la cohérence des index globaux qu'il permet d'obtenir. Or, les auteurs commencent par étudier le temps de propagation d'une nouvelle sur le réseau. Nous pensons que l'intérêt de ce test, c'est surtout qu'il est plus facile - en termes de programmation et de temps de calcul - que le calcul de la cohérence des tables de routage, et qu'il permet de valider la supériorité du mécanisme *partial anti-entropy* : « Our algorithm significantly outperforms ones that use only push anti-entropy for both propagation time and network volume ».

On s'attendait beaucoup à la deuxième conclusion tirée par les auteurs : « (3) We can easily trade off propagation time against gossiping bandwidth by increasing or decresing the gossiping interval ». Cependant, nous ne sommes pas totalement convaincus que le test proposé valide vraiment cette proposition. En effet, lorsque plusieurs informations sont propagées en parallèle, augmenter la fréquence du gossiping peut créer des

congestions et donc ralentir ou du moins plafonner la vitesse de propagation. Le test proposé ne permet pas de se rendre compte de ce phénomène. Il nous semble bien plus pertinent d'étudier cette proposition dans les séries de tests suivantes. Dans la mesure où c'est ce qui est fait, il semble étrange d'avoir voulu tirer une telle conclusion à ce stade.

En fait, ce type de conclusion peut prendre du sens dans une dimension qui n'est pas exploitée ici. On a vu qu'il est plus facile d'étudier la propagation d'un seul filtre de Bloom que la cohérence de l'index global - a minima, la simulation est moins longue. Il serait donc très intéressant d'étudier s'il existe une relation entre les performances de propagation unitaires et les performances globales d'un mécanisme de gossiping, pour voir si on peut déduire les secondes de l'étude des premières. Cela justifierait qu'on commence à tirer des conclusions dès ce stade pour les comparer aux conclusions des tests suivants. Mais cette approche n'est pas explicite ici.

Seconde série de tests : résistance à la montée en charge On en revient ici à l'étude de la cohérence de l'index global que nous évoquions à la section précédente. Les auteurs mesurent le temps nécessaire pour rétablir la cohérence lorsqu'un groupe de pairs rejoint le réseau (taille des groupes : de 50 à 250 par pas de 40). Ils en tirent la conclusion suivante : « even in this worst case scenario, [...] if peers have DSL or higher connectivity, then PlanetP does quite well ». Ils proposent une modification pour s'adapter au cas des réseaux contenant des pairs à faible connexion. Les tests suivants seront faits en intégrant cette situation.

Si on réfléchit sur cette situation d'arrivée massive de pairs sur le réseau, il y a en fait deux actions qui se produisent en parallèle et qui mériteraient d'être étudiées séparément. La question « à quelle vitesse un pair entrant arrive-t-il dans le même état que les autres ? » se décompose en :

- « à quelle vitesse un pair entrant est-il connu de tous les autres - et peut-il contribuer aux résultats ? »
- « à quelle vitesse un pair entrant voit-il tous les autres - et peut-il router les requêtes correctement ? »

On est en droit de se demander dans quelle mesure le temps de convergence est dû à l'un ou l'autre des deux phénomènes. Plus précisément, du point de vue de l'utilisateur, c'est très différent de dire que les nouveaux pairs mettent du temps à télécharger l'index global que de dire que leurs filtres de Bloom mettent du temps à être diffusés à l'ensemble du réseau.

Troisième série de tests : régime permanent La dernière série de tests consiste à évaluer la qualité du mécanisme de gossiping en régime permanent. Pour cela, les auteurs simulent deux types de comportement :

1. Un réseau de 1000 pairs, que 100 pairs rejoignent selon une loi de Poisson, en moyenne toutes les 90 secondes.
2. Un réseau de 1000 pairs, 40% des pairs sont stables. Les autres 60% ont un comportement en loi de Poisson ; ils restent connectés en moyenne 60 mn, puis se déconnectent pendant une durée moyenne de 140 mn. Ce schéma est basé sur les mesures de (Saroiu et al., 2002).

Pour ces deux scénarii, les auteurs présentent le pourcentage de pairs ayant eu connaissance de la nouvelle par rapport au temps, en moyenne sur tous les filtres de Bloom propagés. Ils comparent tout d'abord les graphes obtenus avec et sans mécanisme *partial anti-entropy*. Ensuite, ils comparent les temps de propagation pour les pairs « rapides »et les pairs « lents ». Cela leur permet de valider la technique *bandwidth aware gossiping*, qui vise à assurer de bonnes performances aux pairs bien connectés en présence de pairs à faible connexion. Puis ils montrent la bande-passante consommée sur l'ensemble du temps de simulation.

Ils tirent de ces expérimentations les conclusions suivantes :
- on retrouve une nette amélioration lorsqu'on utilise le mécanisme *partial anti-entropy*
- le temps de convergence moyen se situe autour de 400 secondes
- « our bandwidth aware gossiping algorithm allows fast nodes to propagate events as in the LAN case without harming the speed of propagation to slow nodes. »
- La consommation en bande-passante se situe entre 10 Kb/s et 100 Kb/s.

Analyse Première remarque, cette série de test valide la plus-value qualitative du mécanisme *Partial anti-entropy* bien plus sûrement que la première. Elle confirme les résultats. Il est cependant dommage que ce gain de qualité ne soit pas mis en rapport avec les coûts engendrés. Il est en effet probable que le mécanisme *partial anti-entropy* engendre des messages supplémentaires, et on n'a aucun élément de mesure de cette sur-consommation.

La seconde remarque concerne la validation du mécanisme *bandwidth aware gossiping*. Nous pensons que les résultats présentés ne permettent absolument pas de tirer la moindre conclusion. En effet, il est logique que les filtres de Bloom des pairs « lents » se propagent plus lentement que ceux des pairs « rapides », même dans le cadre du mécanisme de gossiping standard. Pour valider la supériorité du mécanisme *bandwidth aware*, il faut donc nécessairement comparer les résultats avec ceux obtenus avec le mécanisme standard. Ce n'est pas fait ici.

Remarques générales sur l'étude du mécanisme de gossiping Nous ferons deux remarques générales sur l'étude du mécanisme de gossiping. Premièrement, il existe de nombreux types de mécanismes de propagation épidémique (par exemple (Jelasity

et al., 2003)). Ces mécanismes ont été beaucoup étudiés. Ne peut-on pas réutiliser les résultats ? Quid des performances de PlanetP avec un autre mécanisme de maintien de cohérence des tables de routage ? Doit-on reconstruire le système entier pour répondre à cette question ? Nous pensons qu'il est possible de construire une implémentation de PlanetP qui prenne en paramètre les performances du mécanisme de gossiping utilisé, permettant ainsi de découpler l'étude du mécanisme de gossiping de l'étude de PlanetP dans son ensemble.

La seconde remarque, c'est que la limite d'échelle de PlanetP n'est pas précisément étudiée. Les auteurs nous disent que PlanetP fonctionne « of across [...] communities of several thousand, perhaps up to ten thousand peers ». On s'attend à voir un graphique qui montre clairement les performances de PlanetP autour de la dizaine de milliers de pairs. On aimerait chercher la limite : PlanetP, ça marche jusqu'à quand ? Au lieu de cela, les auteurs nous donnent leur avis, leur intuition ; mais celle-ci n'est corroborée par aucun résultat expérimental. On peut aussi se demander à quel point cette limite est intrinsèque à la technologie PlanetP ou dépend de facteurs externes (connexions, volatilité des pairs, ...).

3.1.5 Remarques générales quand à la démarche d'évaluation

Les techniques mises en œuvre par PlanetP peuvent être appliquées à d'autres modèle RI

En comparant PlanetP à une implémentation centralisée de LSI, on donne en fait implicitement une définition de PlanetP : on sous-entend que PlanetP est une implémentation P2P du modèle LSI. Il nous semble que cette définition est très réductrice. Au minimum, les techniques proposées par PlanetP peuvent être appliquées à n'importe quelle implémentation du modèle vectoriel, puisque la technique des filtres de Bloom est compatible avec n'importe quelle implémentation du modèle vectoriel. Mais si on considère le principe d'un index réparti maintenu par gossiping, celui-ci peut s'appliquer à d'autres types de modèles RI que le modèle vectoriel ; il peut même être utilisé pour d'autres applications que la Recherche d'Information. Le problème ensuite, c'est de poser une définition formelle de PlanetP, non pas en tant que modèle RI-P2P, mais en tant que *technique de répartition en P2P*, d'une application ou d'un service. La Recherche d'Information et LSI en particulier apparaissent alors comme des applications particulières qui peuvent être implémentées en P2P par la technique PlanetP.

Plus concrètement, en première approche, plutôt que comparer les résultats fournis par PlanetP et LSI à un troisième oracle (les correspondances requêtes/réponses fournies par les jeux de données) ; il nous semblerait plus intéressant d'évaluer PlanetP en prenant comme oracle les résultats fournis par le modèle LSI. La différence semble minime, mais nous verrons que par exemple nous ne présenterons plus les mêmes courbes comme résultats d'évaluation. C'est un changement d'esprit de l'évaluation.

L'intuition, c'est qu'en étudiant comment PlanetP altère les performances de LSI, on se donne une idée de comment PlanetP altère les performances du modèle vectoriel en général. La question corollaire, c'est de savoir si la façon dont PlanetP dégrade les performances de LSI est un bon indicateur de la façon dont il dégrade les performances de n'importe quelle autre implémentation du modèle vectoriel, de n'importe quel modèle RI, de n'importe quelle application, ...

Il est à noter qu'on sent bien chez les auteurs eux-mêmes la volonté de comprendre plus finement dans quelle mesure PlanetP est fidèle à LSI ou dégrade ses performances : « when comparing the documents returned by PlanetP and CENT at low recall levels, we found an average intersection of 70% . [...] This gives us the confidence that our adaptations did not change the essential ideas behind TF*IDF's ranking. ». Nous verrons comment une comparaison plus fine des résultats calculés par LSI et PlanetP, qui est ici esquissée d'une façon un peu intuitive, peut être mise en place de façon plus systématique et rigoureuse, sur la base de principes très simples.

On ne peut pas évaluer le coût indépendamment de la qualité des résultats ; parce qu'ils dépendent l'un de l'autre ; parce qu'il y a un compromis à trouver entre l'un et l'autre.

L'évaluation est structurée en trois parties. Les auteurs évaluent successivement la qualité des résultats de la Recherche d'information (*4.1 Search Efficacy*), les coûts de stockage engendrés par le système (*4.2 Storage Cost*) et la performance du mécanisme de gossiping (*4.3 Gossiping Performance*). Nous pouvons déjà faire la remarque que ces trois éléments ne sont pas indépendants. Nous proposons d'éclairer cette affirmation par une citation de la section *3.3 Implementing the Global Index : « Memory-constrained Peers can also independently trade-off accuracy for storage by combining several filters into one »* - *« Les pairs à capacité mémoire réduite peuvent indépendamment trouver un compromis entre la précision [de la Recherche d'Information] et les coûts de stockage en combinant plusieurs filtres [de Bloom] en un »*. Si on admet qu'il est possible de trouver un compromis entre la qualité de la recherche d'information et les coûts de stockage ; alors il faut nécessairement présenter des courbes permettant d'étudier l'évolution des performances en fonction des coûts. C'est indispensable à la recherche d'un compromis.

Remarquez que si l'on combine ainsi les filtres de Bloom de plusieurs ressources, cela diminue le nombre de données à partager par le mécanisme de gossiping ; donc on économise des messages ; les messages ainsi économisés peuvent être réinvestis pour améliorer l'efficacité du mécanisme en augmentant la fréquence des échanges. Donc le compromis est au moins tripartite ; il inclut aussi la performance du gossiping. Seulement ceux-ci ne constituent que des exemples d'inter-dépendances. Il est possible que d'autres inter-dépendances existent. Là aussi, nous verrons qu'en posant quelques principes simples on peut détecter et traiter ce type de dépendances de façon plus structurée et systématique.

3.2 Étude critique des résultats d'évaluation de pSearch

3.2.1 Dans l'introduction

Dès l'introduction, les auteurs résument les performances de pSearch en disant que sur un réseau de 128000 pairs, contenant 528543 documents ; une recherche pSearch parcourt 19 pairs, qui échangent 95,5 KB de données ; et les 15 meilleurs résultats retournés par pSearch et LSI ont une intersection de 91.7%. On peut déjà remarquer que ce résultat est exposé de façon structurée ; les auteurs présentent le jeu de données, puis les critères de coût, puis les critères de qualité, nous les décryptons ci-après. Par contre, la description du jeu de données est très légère ; il paraît notamment étonnant de définir des performance indépendamment du type de distribution des données sur les pairs. Au niveau des coûts, il manque des éléments sur les coûts de stockage (tables de routage...) et sur les coûts d'administration du réseau. Notez à ce sujet que pour PlanetP, le coût de résolution d'une requête est très faible, puisque l'essentiel du travail réside dans le maintien d'un fichier réparti des pairs. Donc si on compare PlanetP et pSearch uniquement sur les coûts de résolution d'une requête, PlanetP sera nécessairement (et artificiellement) bien moins coûteux que pSearch. Pour finir, nous trouvons très intéressante l'idée de mesurer la qualité de pSearch en comparant les résultats qu'il retourne à ceux retournés par LSI. Cette approche abonde dans le sens d'un reproche majeur que nous faisons à l'évaluation de PlanetP, en mesurant les *pertes de performances* dues à pSearch, plutôt que ses performances dans l'absolu. Cependant, il nous semble qu'une simple mesure de rappel de pSearch par rapport à LSI est insuffisante. A minima, ce rappel doit être mis en perspective avec la précision obtenue ; nous verrons dans une prochaine partie que le problème des critères de qualité est en fait un peu plus complexe, parce qu'ils doivent permettre de comparer l'ordre des réponses retournées par LSI et pSearch.

Pour synthétiser, à l'analyse de l'introduction, la démarche d'évaluation des auteurs semble plus structurée que pour PlanetP, avec notamment une distinction claire (même si implicite) entre jeu de données, critères de coût et critères de qualité. Cependant, on remarque un manque d'exhaustivité dans la description de ces éléments. On peut aussi remarquer que les auteurs ne parlent pas du tout des paramètres, qui permettent d'ajuster le comportement du système - mais cela se justifie par la nécessaire concision d'une introduction.

3.2.2 Protocole expérimental

Les auteurs construisent un simulateur de pSearch, à partir du système SMART *relié* « linked » à un simulateur du système CAN. Ils valident leur implémentation de LSI sur SMART en testant la précision obtenue sur trois corpus de l'état de l'art.

Comme jeu de données, ils utilisent les jeux de données de TREC-7 et TREC-8.

Ensuite, les auteurs présentent une série de résultats d'expérimentation, dans l'essentiel sous la forme d'une question « que se passe-t-il lorsque..? », suivie de résultats d'expérimentation (courbe(s)), puis d'une analyse de(s) la courbe(s) pour répondre à la question. Dans la grande majorité des expérimentations, ils mesurent le nombre de pairs contactés (note : bonne indication du nombre de messages) et l'« accuracy ». L'accuracy est définie par : soient A l'ensemble des réponses retournées par LSI et B l'ensemble des réponses retournées par pSearch, $accuracy = \frac{|A \cap B|}{|A|}$; c'est donc le rappel de pSearch par rapport à LSI (on utilise l'ensemble des réponses retournées par LSI comme oracle).

Si on essaie de restructurer cette démarche, en fait les auteurs font varier les paramètres système et les caractéristiques du jeu de données. Pour chaque configuration, ils mesurent la qualité des résultats obtenus et les coûts générés pour obtenir ces résultats. Cela permet d'étudier l'évolution du rapport qualité/coût en fonction des paramètres du jeu de données et des paramètres système.

En effet, l'*accuracy* est clairement désignée comme un indicateur de la qualité des résultats, même si nous avons vu qu'elle est un indicateur partiel. Concernant les coûts, la démarche est moins explicite. Cependant, si on anticipe un peu la réflexion que nous mènerons dans la partie suivante, on a trois sortes de coûts : on consomme de la bande-passante, des ressources de calcul (CPU + RAM) et de l'espace mémoire. Il est clair que la puissance de calcul mobilisée ainsi que la bande-passante générée par une requête sont proportionnelles au nombre de pairs interrogés. Les auteurs y font une référence plus ou moins explicite à plusieurs reprises, notamment section *7.2.1 : « The search cost in this figure [...] »* - la figure ne présente pas le coût de la recherche, mais le nombre de pairs atteints - puis dans *7.5 Analysis of System Resource Usage*, quand les auteurs utilisent le nombre de pairs atteints dans la formule de calcul de la bande-passante. Par contre, les auteurs n'étudient pas la consommation d'espace mémoire.

Comme pour tous les systèmes RI-P2P, dans pSearch on peut choisir son compromis entre le coût de la recherche et la qualité des résultats retournés : on peut obtenir de meilleurs résultats au prix d'une plus forte dépense de ressources ; ou bien on peut économiser des ressources, aux dépens de la qualité des résultats. On retrouve généralement pour cela un ensemble de paramètres qui contrôlent la terminaison de la recherche. Grossièrement, plus on termine tôt, plus on économise des ressources au détriment de la complétude des résultats. Dans pSearch, c'est un seul paramètre, le *Quit bound* F, introduit dans la section *6.2 Description of the Search Algorithm*, qui contrôle la terminaison de la recherche.
La grande majorité des diagrammes présente donc le *Quit bound* en abscisses. En ordonnées, on présente une courbe pour l'*accuracy* et des histogrammes pour le nombre de pairs contactés. On visualise donc bien pour chaque valeur du *Quit bound* le compromis qualité/coût correspondant.

Nous n'avons pas encore reconstruit l'intégralité de la démarche d'évaluation. Ce qui manque, c'est qu'on étudie comment évoluent ces compromis qualité/coût en fonction des paramètres du jeu de données et des paramétres système.
Nous verrons qu'on peut se donner des critères pour distinguer l'ensemble des paramètres contrôlant la terminaison de la recherche des autres paramètres système - qui définissent notamment les spécifications applicatives (paramètres du modèle RI) et la structure de données. Nous verrons alors que d'autres critères que le *Quit bound* peuvent être utilisés.
Coté jeu de données, nous comprenons le terme au sens large ; c'est-à-dire qu'est considéré comme paramètre du jeu de données tout élément extérieur au système lui-même (tout ce qui n'est pas un paramètre système). On retrouve donc les ressources, les requêtes et leur modèle de distribution sur le réseau, mais aussi des éléments tels qu'un modèle de la topologie du réseau (physique ou logique, suivant l'objet évalué), des éléments de modèle du corpus, qui impactent notamment la taille des vecteurs LSI etc...
Pour étudier l'impact de ces paramètres sur le compromis qualité/coût, les auteurs présentent sur un même graphique plusieurs courbes/histogrammes représentant les rapports qualité/coût pour différentes valeurs d'un même paramètre.

3.2.3 Les expérimentations

3.2.3.1 Section 7.2.1

Cette section fait une première analyse du rapport qualité/coût, avec un ensemble de paramètres et un jeu de données fixés. L'idée générale est d'étudier l'évolution du rapport qualité/coût avec l'augmentation du nombre de pair et du nombre de réponses. Pour cela, les auteurs procèdent à trois simulations :

Figure 10 On fait varier la taille du réseau seule (échelle). Notez que la taille des *Sample Sets* est adaptée - divisée par deux à chaque fois que le nombre de pairs est quadruplé. En augmentant la taille du réseau sans toucher au nombre de ressources, on étudie en fait surtout l'impact d'une diminution de la concentration de ressources sur le réseau. D'où la nécessaire adaptation du mécanisme *Sample Sets*.

Figure 11 Pour cette deuxième expérimentation, les auteurs font varier l'échelle (nombre de pairs), mais à proportion de ressources constante, cette fois ; c'est-à-dire que le nombre de ressources est augmenté proportionnellement au nombre de pairs.

Figures 12(a) et 12 (b) Ici, on étudie les variations du rapport *qualité/coût* en fonction du nombre de réponses à rechercher. Sur un réseau de 10000 pairs, on lance des requêtes possédant $15.2^0 ... 15.2^6$ réponses (selon LSI), et on mesure le coût de la recherche (en nombre de pairs) et le rappel obtenu.

Les deux premières expérimentations montrent que la qualité des résultats décroît en $\frac{1}{x}$ avec l'échelle. La troisième semble montrer que le taux de rappel est constant

	description	default value
n	number of nodes in the system	10,000
l	dimensionality of LSI and CAN	300
p	number of rotated semantic spaces	4
m	rotated dimensions	use Equation 4
s	size of the sample set $S^i[Z, P]$	50
F	quit bound in Equation 8	24
k	number of returned documents for a query	15
g	number of warm-up queries (size of $Q^i[Z]$)	0
d	the concurrent-search factor in Equation 10	1

Table 1: Parameters varied in experiments.

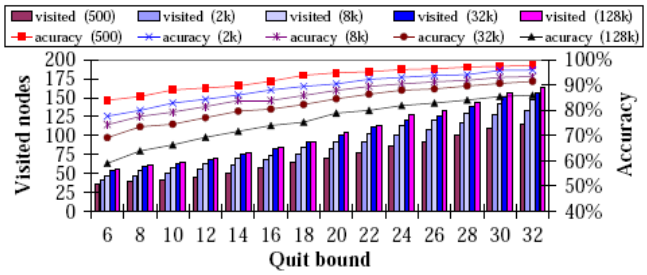


Figure 10: The effect of varying system size.

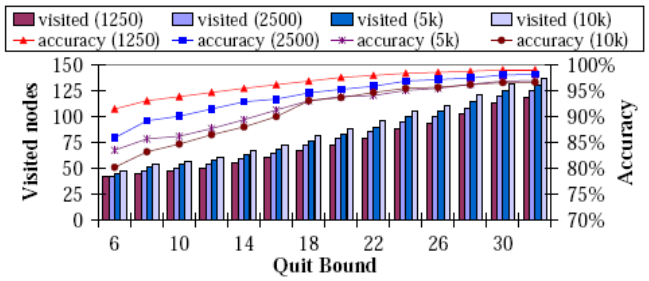


Figure 11: The effect of simultaneously varying system and corpus size.

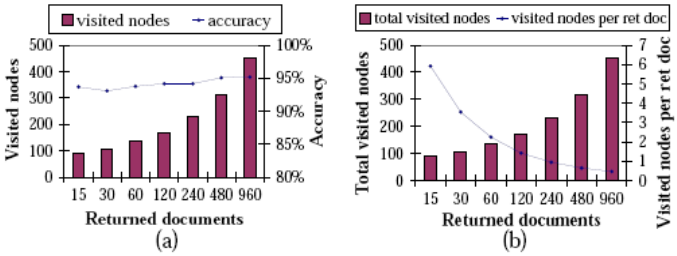


Figure 12: The effect of varying the number of returned documents for a 10k-node system. (a) Visited nodes and accuracy. (b) Visited nodes per returned document.

FIG. 3.1 – **Figures 10 à 12** de l'article (Tang et al., 2003)

quand on fait varier le nombre de réponses à rechercher, ce qui s'oppose aux résultats précédents. Cependant, il faut remarquer que plutôt que de mettre le *Quit bound* en abscisse et de tracer 7 courbes pour les 7 types de requêtes, le nombre de réponses recherchées est passé en abscisse, et le *Quit bound* est ignoré. On ne sait donc pas si le taux de rappel constant doit être attribué à un comportement naturel du système ou à la volonté des auteurs de mesurer les coûts à qualité constante.

Notez que très certainement, le rappel relatif par rapport à LSI diminue, mais le rappel dans l'absolu - nombre de réponses retournées - augmente. C'est-à-dire que le nombre de réponses retrouvées augmente moins vite que le nombre de réponses disponibles ; mais il augmente quand-même. Donc le rapport qualité/coût d'une recherche top-k - le nombre de réponses recherchées est fixé - devrait s'améliorer avec l'augmentation conjointe du nombre de ressources et de l'échelle. On n'a pas d'élément pour vérifier cette hypothèse.

Concernant les coûts maintenant, les trois expérimentations montrent que le nombre de pairs interrogés croît logarithmiquement par rapport à l'échelle, au nombre de ressources ou au nombre de réponses recherchées. Les auteurs suggèrent que le nombre de pairs interrogés dépend du degré des pairs. Mais le degré des pairs ne varie que si le réseau est trop petit pour partitionner toutes les dimensions. À grande échelle, une autre interprétation est nécessaire.

Nous proposons une autre explication : les réponses à une requête sont sémantiquement proches d'elle, donc elles se retrouvent physiquement proches du pair qui maintient la requête sur le réseau. Donc lorsqu'on augmente le nombre de réponses à une requête dans le système, on va surtout charger plus les pairs qui environnent cette requête, et dans une moindre mesure, de nouvelles réponses vont être publiées sur des pairs qui ne possédaient pas de réponses avant. D'où l'augmentation limitée du nombre de pairs parcouru : on obtient plus de réponses parce que les mêmes pairs retournent plus de réponses. Cette interprétation est confortée par la figure 12(b) qui montre que le nombre de pairs visités par réponse retournée diminue (en 1/x) à mesure que le nombre de réponses recherchées augmente.

Cette interprétation, ainsi que les propriétés de l'algorithme de recherche qui en découlent (coût de la recherche), peuvent être étudiées en indexant les ressources dans une implémentation centralisée de la structure de données CAN - typiquement avec objets « cluster » contenant des ressources et reliés entre eux par des pointeurs. Cela permet de simplifier considérablement la complexité de l'évaluation, et de s'abstraire des problématiques « système P2P » pour se concentrer sur l'étude des propriétés de la structure de données.

Dans les 3 sections suivantes, les auteurs étudient l'avantage d'utiliser les requêtes passées dans les *Sample Sets*, d'échanger des *Sample Sets* sur deux sauts, ou de mettre-en-place un mécanisme de réplication des index de ressources sur ses voisins.

3.2.3.2 Section 7.2.2

Dans cette section, les auteurs évaluent l'impact de la prise en compte des requêtes (passées) dans les *Sample Sets* (heuristique nommée *query*, par opposition à l'heuristique *content*). Dans un premier temps, ils tracent sur un même graphique les courbes pour les configurations avec et sans requêtes dans les *Sample Sets*. Ce graphique montre un rapport qualité/coût observé moyen pour chaque valeur du *Quit bound*. Sur cette expérimentation, les auteurs ont souhaité étudier plus en détails les niveaux de qualité obtenus pour chaque requête. Ils montrent un diagramme indiquant le nombre de requêtes ayant obtenu un rappel de 50%, 60%, ... 100%.

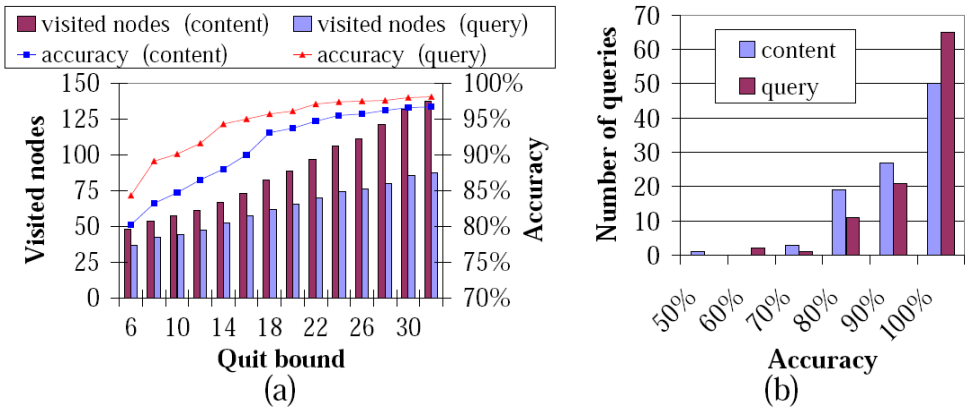


FIG. 3.2 – **Figure 13** de l'article (Tang et al., 2003) - *Comparing content-directed search heuristics fo a 10k-node system. (a) Accuracy and visited nodes. (b) Accuracy histogram.*

On remarque que l'utilisation des requêtes permet systématiquement de diminuer le nombre de pairs contactés et d'améliorer le taux de rappel. À ce propos, (Handurukande et al., 2004) et (Sripanidkulchai et al., 2003) ont étudié l'utilisation des requêtes passées dans les réseaux non-structurés. Mais il n'en font pas tout-à-fait la même utilisation. Ces articles étudient l'heuristique « un pair qui a répondu à mes requêtes dans le passé répondra à mes requêtes dans le futur ». Dans pSearch, on considère que les requêtes émises par un pair sont un bon indicateur de son domaine d'expertise.

3.2.3.3 Section 7.2.3

Ici, les auteurs évaluent l'apport d'un mécanisme de réplication. Ils étudient deux types de réplication : avec *repl*, chaque pair réplique les index de ses voisins (chaque index est répliqué sur tous les voisins de son pair d'origine) ; avec *repl-content*, en plus de répliquer les index, chaque pair réplique les *Sample Sets* de ses voisins.

Vu autrement, *repl* rend chaque pair capable de répondre à la place de ses voisins ; *repl-content* ajoute une modification de la structure du réseau, telle que les voisins des voisins d'un pair deviennent ses voisins.

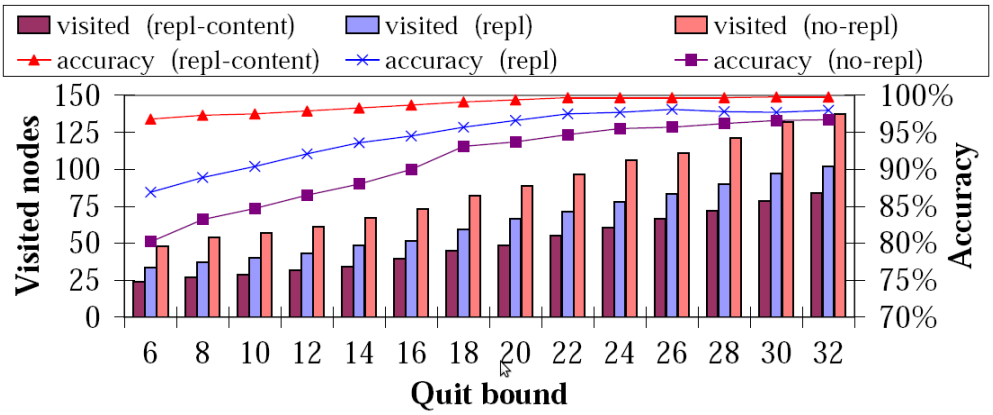


FIG. 3.3 – **Figure 14** reprise de l'article (Tang et al., 2003) - *The effect of replication on a 10k-node system.*

On a une nette amélioration de la qualité des résultats et une nette diminution du coût de la recherche, en passant de la configuration classique à *repl* puis de *repl* à *repl-content*. Cependant, cette amélioration s'obtient au prix d'une grosse dépense en place mémoire, qui n'est ni évoquée ni quantifiée ici.

repl : On visite les mêmes pairs, mais chacun maintient plus d'index de ressources ; donc on couvre plus de ressources, d'où un meilleur rappel. Par contre, logiquement on pourrait baisser les coûts en modifiant l'algorithme de recherche. En effet, ici certaines ressources sont évaluées trois fois : par le voisin par lequel la requête arrive, puis par le pair courant, possesseur initial de la ressource, puis par le voisin auquel la requête est propagée.

repl-content : On augmente le voisinage des pairs. D'où un meilleur rappel/précision sur le choix des pairs, qui se retrouve dans une meilleure qualité des ressources

retournées. Cet algorithme résout partiellement le problème de l'évaluation multiple des ressources créé par la réplication.

3.2.3.4 Section 7.2.4

Cette fois, on passe de 10000 à 128000 pairs, et on compare *content*, *repl*, *query*, *repl-content* et *repl-query*. On confirme la transitivité des résultats précédents : *repl-query* surpasse en qualité et coût toutes les autres configurations.

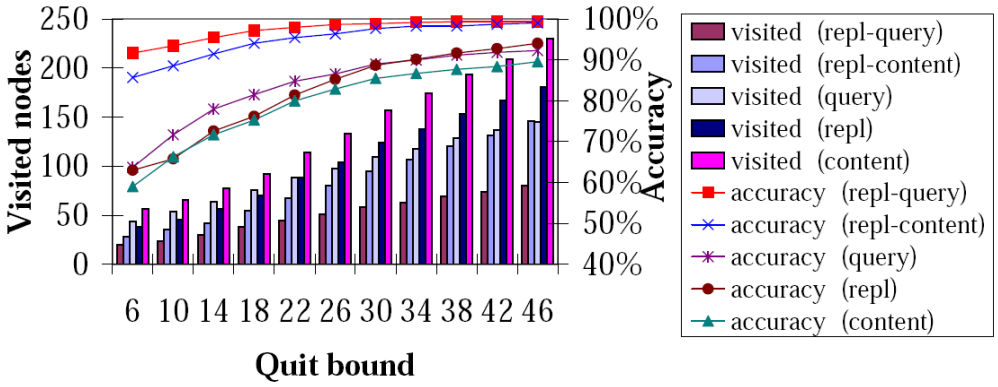


FIG. 3.4 – **Figure 15** reprise de l'article (Tang et al., 2003) - *Performance on a 128k-node system.*

On remarque que dans cette série de tests, les critères sont ajoutés progressivement, pour complexifier progressivement les courbes affichées ; c'est sans doute une idée à retenir.

3.2.3.5 Section 7.2.4 : Analysis of Result Source Distribution

De nouveau, on quitte temporairement le schéma d'expérimentation général, cette fois pour analyser l'origine des requêtes.

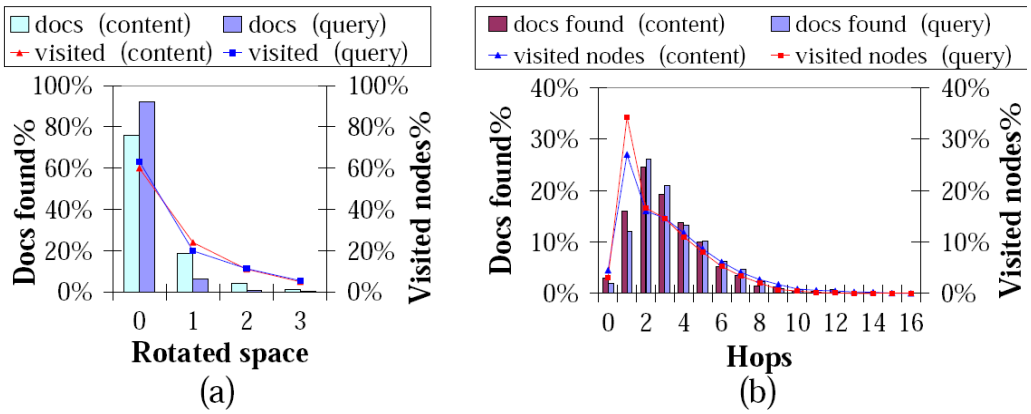


FIG. 3.5 – **Figure 16** reprise de l'article (Tang et al., 2003) - *Distribution of documents for a 10k-node system. (a) Rotated spaces. (b) Hop counts.*

Figure 16(a) La première courbe montre la répartition des réponses par *rotated space*. La répartition des réponses correspond à la répartition des termes en général, montrée sur la *figure 7, section 4*. Elle semble cependant moins concave ; c'est peut être dû au fait que le corpus utilisé est plus petit (voir le début de la section 7.1 pour les détails) ; et que de toutes façons, le nombre de réponses est très inférieur au nombre de ressources ; cela peut expliquer que la courbe soit moins « racée ».

Figure 16(b) La seconde courbe montre le nombre de ressources retrouvées à chaque saut, pour les heuristiques *content* et *query*. De manière générale, les deux courbes ont une très longue queue, l'essentiel des ressources est retrouvé à une distance 4 ou 5 du premier pair. Ensuite, l'heuristique *query* a une plus longue queue, ce qui traduit que sa supériorité par rapport à *content* est due à sa capacité à retrouver les réponses situées loin du premier pair (remarque des auteurs).

3.2.3.6 Section 7.4 : Sensitivity to system parameters

Dans toute cette section, les auteurs étudient les performances de *pLSI* (et non pSearch).

Figure 17(a) Ils commencent par étudier les performances pour différentes configurations du système SMART. Il s'agit toujours de performances relatives ; on mesure donc la fidélité des résultats retournés par pLSI à ceux retournés par LSI. Dit autrement, on étudie l'évolution du rapport qualité/coût lorsqu'on fait varier les paramètres du

modèle de Recherche d'Information. Concernant le nombre de pairs parcourus, les différences entre configurations sont très stables par rapport au *Quit bound*, et divergent dans une fourchette de 15%. Du point de vue de la qualité des résultats, l'*accuracy* obtenue pour les différentes configurations tient dans une fourchette d'environ 7% - ce qui est faible - qui tend à se resserrer à mesure que le *Quit bound* augmente.

Figure 17(b) Cette deuxième courbe étudie les variations du rapport qualité/coût en fonction de la longueur des requêtes. Nous interprétons la longueur des requêtes comme un paramètre du jeu de données. Il semble que la qualité des résultats soit encore moins sensible à ce paramètre qu'au précédent.

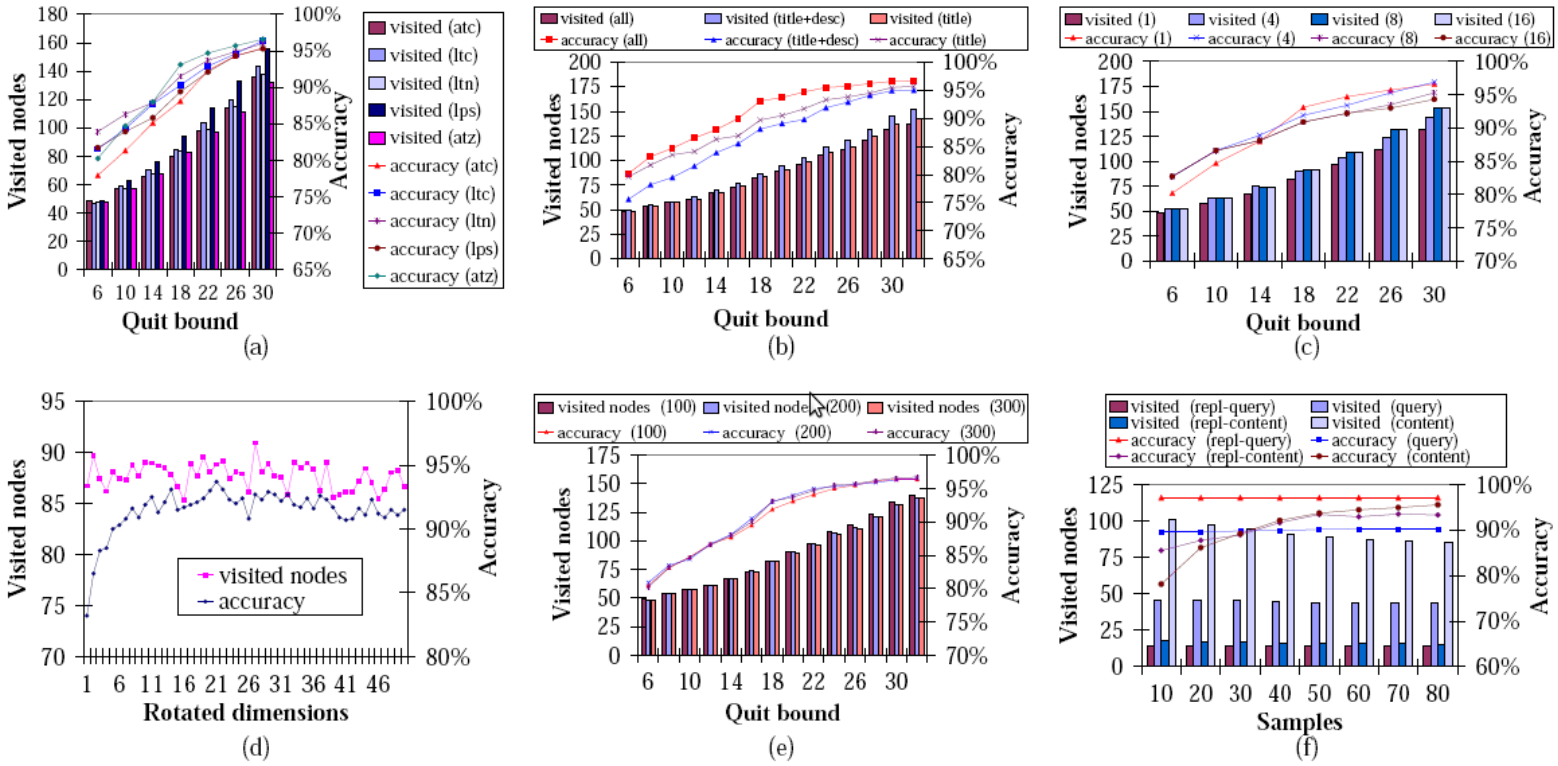


FIG. 3.6 – **Figure 17** reprise de l'article (Tang et al., 2003) - *Sensitivity to system parameters for a 10k-node system. (a) Term weighting schemes. (b) Query length. (c) Parallel search. (d) Rotated dimensions. (e) Dimensionality of the semantic space. (f) Sample size.*

Figure 17(c) Ensuite, les auteurs étudient les variations du rapport qualité/coût en fonction du *concurrent search factor*, qui contrôle le nombre de pairs interrogés en parallèle. Encore une fois, on a de très faibles variations du nombre de pairs visités et de la qualité des résultats.

Cependant, ce problème aurait pû être étudié différemment. En effet, l'heuristique de parallélisation optimale peut se lire sur une courbe présentant l'évolution de l'*accuracy* par rapport aux pairs interrogés, dans laquelle les pairs sont triés par ordre décroissant de pertinence. Si on remarque par exemple qu'il faut systématiquement parcourir au moins les 15 premiers pairs pour atteindre 80% de rappel, alors interroger les 15 premiers pairs en parallèle, puis interroger les suivants 1 par 1 ; permettra de diminuer les temps d'exécution sans augmenter les coûts de recherche. De manière plus générale, une courbe *rappel/pairs* permet de *calculer* le rapport *temps de réponse/coût* correspondant à une heuristique de parallélisation donnée.

Figure 17(d) Puis les auteurs étudient l'impact de m, la taille des p projections selon lesquelles chaque index est découpé (p est fixé à 4 *rotated spaces*). De façon surprenante, cette courbe ne suit pas le schéma habituel, les auteurs omettant le paramètre *Quit bound*. En conséquence, m est passé en abscisses ; et on suppose que l'expérimentation utilise la valeur par défaut *Quit bound* $= 24$. Cela rend les résultats exposés moins précis ; il est vrai cependant que les auteurs expérimentent 50 valeurs de m et qu'il aurait été difficile de présenter 50 courbes sur le même graphique.

Le résultat de cette expérimentation est qu'à partir d'environ $m = 10$, le rapport qualité/coût du système ne semble plus dépendre de m. Les auteurs conjecturent que $(m = 10) \times (p = 4) = 40$ permet d'intégrer les 40 dimensions les plus significatives (en référence au *graphique* $7(a)$ présenté à la *section* 4). Cette assertion aurait pu être vérifiée - ou du moins renforcée - par une étude du rapport qualité/coût de plusieurs couples (p, m) tels que $p \times m = d$, où d prend des valeurs proches de 40 (e.g $d \in [1; 50]$).

Figure 17(e) Dans cette avant-dernière étude, les auteurs mesurent l'impact de la dimensionnalité des index LSI sur les performances de pLSI. Visiblement, celle-ci n'a aucun impact. On ne sait pas comment p et m sont fixés en fonction de l. Il serait étonnant qu'ils soient fixes. Les auteurs ne précisent pas non plus comment ils font varier l - troncature du corpus, des index...

Figure 17(f) Enfin, les auteurs étudient l'impact de la taille des *Sample Sets* sur les performances de pLSI, pour les heuristiques *query*, *content*, *repl-query* et *repl-content*. Ici, on a deux dimensions pour les paramètres système étudiés : on fait varier la taille des *Sample Sets* et leur type (*query* etc...). Pour pouvoir présenter ce résultat

sur un graphique en deux dimensions, les auteurs omettent de faire varier le *Quit bound* et le remplacent sur l'axe des abscisses par la taille des *Sample Sets*.

Les heuristiques *query* et *repl-query* sont complètement insensibles aux variations de taille des *Sample Sets*. Les performances des heuristiques *content* et *repl-content* se stabilisent à partir de *Sample Sets* de taille 50 environ.

Il est important de remarquer que les auteurs étudient pLSI et non pSearch. Il y a fort à parier que les différences de qualité et de coût soient plus marquées avec pSearch. En effet, on peut considérer que pLSI est une implémentation en P2P du modèle RI LSI - qui peut donc être vu comme une spécification. La perte de qualité dans les résultats de pLSI par rapport à LSI est donc imputable aux contraintes de l'environnement P2P : on introduit la notion de pair, avec une évaluation en deux phases : trouver les pairs répondant à la requête puis trouver sur ces pairs les ressources répondant à la requête. Bien sûr, une autre partie des pertes est due aux problèmes de latence et de perte de messages.
Les modifications apportées par pSearch par rapport à pLSI se situent sur un autre plan, et sont complémentaires. Cette fois, on transforme le modèle RI, en introduisant les *rotated spaces* et l'heuristique de recherche basée sur les *Sample sets*. Donc en théorie, on conserve les éléments impliquant les pertes de performance de pLSI et on en ajoute de nouveaux.

3.2.3.7 Section 7.5 : Ressources consommées

En lieu de l'étude des ressources consommées, les auteurs étudient uniquement la bande-passante consommée par le mécanisme de recherche (propagation d'une requête). La bande-passante consommée par l'administration du réseau (une structure CAN !) est complètement passée sous silence. Idem pour la bande-passante consommée pour la mise-en-place des *Sample Sets*.

La bande-passante n'est pas le seul type de ressources consommé. Le réseau structuré et les *Sample Sets* consomment beaucoup d'espace mémoire sur chaque pair. Cet aspect est totalement occulté. Enfin, pour être complet, il faudrait ajouter à cela une étude de la charge de calcul impliquée sur les pairs par le système.

On notera que l'étude de la bande-passante consommée par les requêtes est précise, et paramétrée par le nombre de pairs parcourus ; paramètre mesuré dans les expérimentations. Cependant, il y a ici une erreur : le nombre de pairs interrogés n'est qu'une approximation du nombre de messages, du fait des pertes de messages et des cycles (2 messages pour un même pair).

3.2.4 Remarques générales

Tant qu'à faire que d'évaluer pSearch par rapport à LSI, on peut aussi utiliser les index calculés par LSI comme jeu de données. C'est-à-dire que plutôt que de manipuler des documents, le simulateur peut manipuler des index des documents. Cela permet de simplifier le simulateur, en n'y intégrant pas le système SMART, cela fait gagner du temps de simulation, en capitalisant l'indexation pour les différents paramétrages du système pSearch. Notez que le jeu de données ainsi produit (jeu de vecteurs LSI) pourrait ensuite être utilisé pour évaluer PlanetP.

Notez que la description de la section *7.1 Experimental Setup* semble indiquer que c'est ce qui a été fait ; donc que les index de ressources ont bien été calculés statiquement, en amont de la simulation ; mais cela est contradictoire avec les explications données dans l'introduction.

Si on combine toutes les petites différences de 5% d'*accuracy*, la qualité de la meilleure configuration peut très vite diverger très significativement de la moyenne. Il n'est pas évident du tout dans cette étude de se rendre compte des qualités qui peuvent être atteintes en utilisant la configuration optimale.

D'une approche pragmatique, le problème de l'évaluation peut se poser ainsi : soient les paramètres du jeu de données, soit un coût qu'on accepte de payer, quel est le jeu de paramètres du système qui nous permet d'obtenir la meilleure qualité de résultats ? On a du mal à répondre à ce type de question en sortie de cette étude.

On remarque un manque de structuration et d'exhaustivité dans l'étude de l'impact des paramètres système sur le rapport qualité/coût du système. pSearch est composé de 6 éléments :

- le modèle RI : LSI ;
- la technique d'indexation des pairs, avec une résolution en deux étapes : retrouver les pairs répondant à la requête puis retrouver sur ces pairs les réponses ;
- la technique des *rotated spaces*, qui « découpe »les index de requête pour adapter leur taille à ce que peut supporter le réseau structuré ;
- la structure d'indexation, munie d'un algorithme de résolution des requêtes ;
- l'heuristique de parcours du voisinage qui utilise les *Sample Sets* ;
- l'implémentation répartie des mécanismes d'indexation, de la structure d'indexation et des mécanismes de parcours.

Cette analyse peut servir de support pour structurer l'évaluation. En effet, ces six mécanismes consomment des ressources. Ils impliquent chacun des pertes de performances, amenées par des biais différents. La qualité des résultats de chaque mécanisme tient

lieu de jeu de données pour le suivant. Par exemple, la taille des index LSI tient lieu de paramètre de jeu de données du point de vue de la technique d'indexation des pairs.

3.2.5 Une remarque importante

Dans cette section, nous commentons une remarque faite à la section*3.1 The Basic algorithm* : « in theory, pLSI can achieve the same precision as LSI ». Nous sommes complètement d'accord avec cette remarque. Finalement, pLSI n'est qu'une implémentation du modèle LSI, qui n'est en rien modifié. Donc en fait pLSI introduit une problématique de coût, ce qui est tout le problème de définir une « bonne » implémentation d'un modèle ; mais dans la mesure où il est fidèle aux spécifications, la qualité des résultats est la même que pour LSI.

Par contre, il nous semble que cette remarque très pertinente est sous-utilisée. D'après ce que nous venons de dire, si on voulait évaluer pLSI, on n'aurait pas à évaluer la qualité des résultats, mais seulement les coûts générés. Et cela représente potentiellement un gain de temps énorme. En fait, notre approche est un peu simpliste, parce que la nature distribuée de l'implémentation introduit des imprécisions dans l'exécution de l'algorithme (dues aux pertes de messages, aux latences etc...). En conséquence, il faudrait quand-même d'évaluer l'impact de ces imprécisions sur la qualité des résultats. Mais on ouvre tout de même la voie vers une simplification importante de l'effort d'évaluation - au prix cependant d'un effort de réflexion et d'analyse du fonctionnement de pSearch. Notez aussi que cette approche permet d'obtenir des résultats d'évaluation mieux réutilisables, dans la mesure où le coût d'une implémentation répartie de LSI donne une bonne idée du coût d'une implémentation répartie d'un autre modèle vectoriel. De même, il y a sans doute des corrélations à faire entre « l'impact de ces imprécisions sur la qualité des résultats » de différentes implémentations du modèle vectoriel.

Cependant, pLSI n'est pas pSearch. Mais cette remarque sur pLSI ouvre deux pistes pour l'évaluation de pSearch. La première piste serait de considérer pLSI comme une étape intermédiaire dans le processus de construction de pSearch à partir de LSI. Plutôt que de comparer les résultats de pSearch à ceux de LSI, on peut se demander si comparer la qualité des résultats de pSearch à ceux de pLSI ne permettrait pas de simplifier le processus d'implémentation. La seconde piste, c'est de modifier les spécifications ; donc de trouver le modèle voisin de LSI dont pSearch est l'implémentation en P2P. On aurait ainsi les mêmes relations entre ce modèle RI et pSearch qu'entre LSI et pLSI.

3.3 Synthèse

Cette section comprend deux parties. Dans la première, nous faisons le point sur les manques identifiés dans ces deux évaluations de systèmes P2P. Dans la seconde partie, nous posons les bases d'une démarche d'évaluation idéale.

3.3.1 Les manques identifiés sur ces deux évaluations

D'après les remarques formulées dans les sections précédentes, la façon dont sont opérées les évaluations de PlanetP et pSearch pose quatre types de difficultés :

- on ne peut pas comparer les systèmes ;
- l'évaluation n'est pas complète : l'inluence de certains paramètres, certains aspects ou composantes des systèmes ne sont pas étudiés
- on a peu d'éléments pour comprendre comment chaque composante d'un système participe aux performances globales ;
- les conditions limites, conservant au système des performances acceptables, sont mal définies ;
- enfin, ces études posent des problèmes de réutilisation de résultats : elles réutilisent peu de résultats d'évaluations précédentes, et les résultats qu'elles apportent seront difficiles à réutiliser.

Dans la suite, nous synthétisons comment les remarques des sections précédentes étayent ces idées.

3.3.1.1 Comparer les systèmes

PlanetP est évalué dans l'absolu (oracle construit manuellement), tandis que pSearch est évalué par rapport aux résultats fournis par une implémentation centraliée. Sur ce premier aspect, les courbes obtenues ne sont donc déjà pas comparables.

De plus, les critères de qualité mesurés sont différents. Pour PlanetP, pour chaque requête, pour chaque jeu de données, les auteurs mesurent la précision et le rappel, à partir desquels on suppose qu'il doit être possible de tirer une courbe précision/rappel. Pour pSearch, les auteurs ne mesurent que le rappel - l'accuracy correspond au rappel de pSearch par rapport aux résultats fournis par LSI. Les critères de qualité mesurés ne correspondent donc pas bien non plus. Notez aussi qu'a priori, dans la mesure où ce sont tous deux des systèmes de Recherche d'Information, on s'attendrait à trouver directement des courbes précision/rappel - traditionnelles en RI. Le choix de ne pas présenter ces courbes n'est pas justifié par les auteurs.

Aucun des deux articles ne mesure les temps de réponse. On peut estimer que dans l'étude de pSearch le *Quit bound* et le nombre de sauts en sont des indicateurs, mais les auteurs ne posent pas clairement la relation entre ces indicateurs et les temps de

réponse. Globalement, il n'est pas possible de comparer les temps de réponse des deux systèmes.

Enfin, dans l'évaluation des deux systèmes, on a difficilement une mise en relation claire de la qualité des résultats produits par le système par rapport aux coûts générés. On ne peut pas comparer les qualités obtenues à même coût ou les coûts générés à qualité constante.

3.3.1.2 Granularité de l'évaluation

Dans PlanetP, on compare les résultats retournés par PlanetP aux résultats retournés par un système centralisé. Cependant, on a plusieurs modifications qui mènent du système centralisé à planetP : la compression par filtres de Bloom, puis leur partage par la technique de gossiping. On ne peut pas ici distinguer l'impact de ces deux modifications sur les performances du système. C'est pourtant une information importante si on veut tenter d'améliorer le système.

Toujours dans PlanetP, les auteurs étudient l'impact d'une indexation suivant les 30% des termes les plus significatifs. En fait, cette technique modifie le rapport qualité/-coût du modèle RI de recherche inter-cluster. Finalement, la possibilité d'utiliser cette technique est un paramètre du modèle de Recherche d'Information tout autant que la possibilité de compresser les index par la technique des filtres de Bloom. La qualité du système devrait être paramétrable par l'impact de ce type de modification du modèle RI.

Dans pSearch, on sait que la littérature présente plusieurs approches à la gestion d'un réseau logique. Il serait intéressant d'avoir une idée de la façon dont une approche donnant un résultat plus ou moins optimal influerait sur les performances du système. Par exemple, un système peut maintenir une structure dégradée, à moindre coût. Quelle serait la qualité d'un système RI construit sur ces bases ?

Nous avons évoqué l'ambiguité du problème de mesurer les performances du mécanisme de gossiping lorsqu'un nouveau pair rejoint le réseau. D'un coté, le nouveau pair doit télécharger une copie de l'index global. De l'autre, le filtre de Bloom du nouveau pair doit être propagé aux « anciens pairs ». Dans ce cas, le mécanisme de gossiping est utilisé dans deux buts différents. D'un coté, on observe la propagation d'une nouvelle prise individuellement ; de l'autre, on observe la construction des tables de routage. Ses performances pour remplir ces deux fonctionnalités doivent être mesurées ; elles impliquent des expérimentations différentes, avec des mesures de qualité différentes.

Les auteurs de pSearch n'évaluent que le mécanisme de recherche (propagation des requêtes). Or, le mécanisme de recherche de pSearch repose sur un réseau logique particulier, dont la construction et la maintenance engendrent des coûts. Il faut prendre en compte ces coûts pour comparer pSearch et PlanetP.

3.3.1.3 Cas limites : dans quelles conditions utiliser ou pas ces techniques

Dans pSearch, les ressources sont re-distribuées sur les pairs. Or, on pourrait imaginer ne pas mettre en place ce mécanisme, et indexer les pairs dans la structure CAN sur la base d'index de contenu. De même dans PlanetP, si les ressources sont trop uniformément distribuées sur les pairs, les filtres de Bloom seront peu discriminants. On pourrait envisager dans ce cas de mettre en place dans PlanetP un mécanisme de redistribution des ressources sur les pairs. Nous pensons que la question de la nécessité d'un mécanisme de redistribution des ressources est indépendante de la solution P2P adoptée (type pSearch ou type PlanetP). Il serait intéressant de caractériser à quel moment cette option devient nécessaire.

Dès le début du document, les auteurs de PlanetP positionnent leur système dans des échelles de quelques dizaines de milliers de pairs maximum. Du coup, ils négligent de mesurer les performances du système à « grande » échelle : on ne connaît pas les limites de PlanetP en termes d'échelle.

3.3.1.4 Réutilisation des résultats

Autant le gossiping que la technique des filtres de Bloom et le modèle vectoriel ont été intensivement étudiés. On a beaucoup de résultats sur leurs performances. Pourtant, dans une étude comme dans l'autre, aucun résultat extérieur n'a été réutilisé, réintroduit dans les mesures de performances.

Dans l'étude de pSearch, les auteurs étudient le mécanisme de recherche, indépendamment des mécanismes de maintien de la structure d'indexation (structure de l'overlay). Cependant, ils ne prennent pas en compte que cette structure peut être dégradée plus ou moins fortement, suivant les performances des mécanismes de maintien. Si de nouveaux mécanismes sont inventés et évalués, on ne pourra donc pas ré-injecter leurs résultats d'évaluation dans cette étude ; on ne pourra donc pas connaître les performances de pSearch fonctionnant sur la base de ces nouveaux mécanismes.

3.3.2 La démarche d'évaluation idéale : principes fondateurs

Pour surmonter les problèmes rencontrés par les évaluateurs de PlanetP et pSearch, nous proposons de nous appuyer sur une démarche originale d'analyse de l'architecture des systèmes P2P. Cette vision des systèmes P2P permet de « découper » les systèmes, pour mettre en évidence d'un coté une application à « P2P-iser » et de l'autre un ensemble de techniques P2P. Une difficulté de cette approche est aussi de bien comprendre les interactions entre les composants mis en évidence. Mais une fois qu'on maîtrise bien les composants d'un système et leurs interactions, on est en mesure de faire une évaluation composant par composant.

- Cette approche permet de mieux mesurer l'impact de chaque composant du système sur les performances globales.
- L'objet de l'évaluation est plus petit, plus précis. Cela facilite la compréhension et l'exhaustivité sur les paramètres et critères de qualité à prendre en compte dans l'évaluation.
- On a aussi une meilleure maîtrise des exigences à avoir sur le jeu de données - quelles propriétés du jeu de données on veut évaluer.
- On peut choisir indépendamment la technique d'évaluation la plus adaptée pour chaque composant - simulation, analyse, implémentation centralisée...
- Dans cette vision, concevoir un système P2P, c'est assembler des composants. Notre démarche d'évaluation composant par composant donne donc des résultats directements ré-injectables dans un processus de design ou re-design.
- Si un même composant se retrouve dans plusieurs systèmes différents, son évaluation peut être capitalisée d'un système sur l'autre.
- Au final, on ne va plus parler de comparer des systèmes, mais on va comparer des morceaux de systèmes différents situés au même niveau fonctionnel - grosso-modo, situés sur une même ligne du tableau de la section 2.11.

3.3.3 Suite du document

Dans le chapitre suivant, nous présentons de manière plus complète notre démarche d'analyse des sytèmes P2P. Nous concrétisons cette vision originale du domaine dans deux directions, l'évaluation des systèmes P2P, exposée dans le chapitre 5 et la conception de systèmes P2P, exposée dans le chapitre 6.

Chapitre 4

Notre démarche

Dans cette section, nous introduisons notre vision des systèmes P2P, notamment par le biais d'un ensemble de définitions. Nous montrons comment nous utilisons cette vision pour mieux évaluer les systèmes, puis pour aider à leur conception. Nous finissons par une description détaillée des composants, un outil graphique que nous avons mis au point pour décrire et manipuler la structure des systèmes P2P.

L'idée, c'est qu'un système P2P est l'implémentation en P2P d'une application, c'est-à-dire d'un ensemble de fonctionnalités. La clef d'une meilleure compréhension des systèmes P2P serait donc de mieux comprendre l'articulation entre application et implémentation P2P. En effet, une application est généralement modifiée au cours du processus d'implémentation en P2P. Symétriquement, les techniques P2P doivent être adaptées à l'application implémentée.

Nous fournissons des outils pour extraire précisément les spécifications applicatives implémentées par un système donné des composants formant son implémentation. Cela permet une évaluation plus efficace. D'un autre coté, nous fournissons des outils pour mettre à disposition du concepteur une bibliothèque de composants utilisables pour construire son système. En évaluant les performances de chaque composant, et en analysant la façon dont ils peuvent être assemblés en un système, on obtient un outil puissant pour guider les choix de conception.

4.1 Un système P2P : l'implémentation en P2P d'une application

Nous commençons donc par un ensemble de quatre définitions, pour poser les bases de notre vision des systèmes P2P. D'un coté, on décrit les fonctionnalités implémentées à l'aide des notions d'*application* et de *service*. De l'autre, ces fonctionnalités sont implémentées par des *solutions P2P* et des *mécanismes P2P*.

Définition 3.1 : Application Nous désignons par *application* l'ensemble des fonctionnalités implémentées (ou implémentables) par un système P2P (un système P2P est une implémentation en P2P d'une application).

Entre autres exemples d'applications, on trouve le partage de fichiers, la Recherche d'Information, les bases de données et à peu près tout ce qu'on peut imaginer d'implémenter par un système réparti.

La notion de service Que ce soit d'un point de vue évaluation ou conception, la notion d'application est peu utilisable parce qu'insuffisamment précise. C'est pourquoi nous découpons les applications en services.

Dans les grandes lignes, d'un point de vue évaluation, on fait un découpage fonctionnel : un service assure le partage d'un type de ressources et un seul, en résolvant un type de requêtes. C'est ainsi que nous distinguions dès l'introduction le service d'accès du service d'indexation, qui composent à eux deux l'application de Recherche d'Information. D'un point de vue conception, nous appuyons notre découpage sur la notion fondamentale de solution P2P, définie ci-dessous. Nous découperons donc l'application en services correspondant chacun à une et une seule solution P2P. Ce second découpage est cohérent avec le premier. C'est-à-dire qu'un service d'un point de vue évaluation se décompose en plusieurs services d'un point de vue conception. On obtient donc une hiérarchie de services, comme sur la figure 4.1.

En conséquence, dans la suite du document, la notion de système P2P pourra se référer à un service ou à un ensemble de services (ex : une application), suivant le contexte.

Un service est implémenté en P2P par une solution P2P. Pour définir ce qu'est une solution P2P, nous avons besoin de la notion de mécanisme P2P.

Définition 3.3 : Mécanisme P2P Un mécanisme P2P prend en charge un type d'objets partagés entre les pairs. Typiquement, les pairs partagent des requêtes, des réponses, des descripteurs de pair (partagés pour construire la topologie), peut-être des

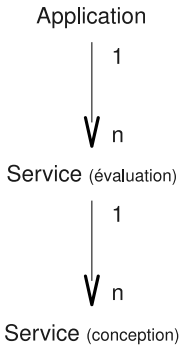


FIG. 4.1 – Le découpage d'une application en services forme une arborescence

informations de *feedback* ou des informations statistiques globales... À chaque fois, un mécanisme P2P définit comment ces objets sont partagés.

Un mécanisme P2P peut remplir deux types de fonctionnalités : soit il maintient la cohérence d'un objet entre plusieurs pairs, soit il permet que cet objet soit construit - calculé - de façon coopérative par un algorithme distribué. À ce sujet, on peut comprendre la propagation d'une requête sur un réseau comme un mécanisme de maintien de cohérence : il s'agit de maintenir la connaissance de la requête sur tous et uniquement les pairs qui sauront y répondre.

Définition 3.4 : Solution P2P Une solution P2P est une option d'implémentation d'un service. Nous avons recensé quatre types de solutions P2P :

- *Structure d'indexation répartie* : L'index des pairs est implémenté de façon répartie sur le réseau. Chaque pair est un nœud de l'index ; se déplacer dans l'index consiste à se déplacer de proche en proche entre les pairs - et implique donc des échanges de messages. Des mécanismes P2P assurent la cohérence de la structure.
  On trouve dans cette catégorie les systèmes dits structurés, c'est aussi la solution adoptée pour la première étape de recherche dans pSearch (Tang et al., 2003).
- *Fichier d'index distribué* : Chaque pair maintient une copie de l'index des pairs. Des mécanismes P2P assurent la cohérence de ce fichier sur tous les pairs. On peut limiter volontairement la cohérence du fichier - par exemple parce qu'on sait que les ressources sont très répliquées et qu'un pourcentage seulement des pairs assure une qualité de service suffisante.
  Cette solution est adoptée dans PlanetP (Cuenca-Acuna et al., 2003).
- *Réseau étiqueté* : Des index sont mis-en-place, qui permettent une résolution par parcours du réseau. Cependant, la structure du réseau est indépendante de ces index. Les index sont donc un étiquetage construit au-dessus de la structure, utilisé pour guider la mise en cohérence des requêtes.

On retrouve cette solution dans *Routing Indices* (Crespo and Garcia-Molina, 2002a) et pour la seconde phase de recherche dans pSearch (Tang et al., 2003).

– *Résolution probabiliste* : Aucun index n'est mis en place. Dans ce cas, on ne peut que choisir aléatoirement le sous-ensemble des pairs sur lequel les requêtes sont mises en cohérence.
Couramment, on propage pour cela les requêtes de proche-en-proche aléatoirement, construisant ainsi un sous-arbre de recouvrement aléatoire du réseau. C'est la solution adoptée dans les systèmes tels Gnutella (Klingberg and Manfredi, 2002) et Freenet (FreenetWebsite,).
Le mécanisme de gossiping peut aussi être utilisé pour implémenter une résolution probabiliste.

Pour une même solution P2P plusieurs implémentations sont possibles, en utilisant différents ensembles de mécanismes P2P. Ainsi, dans pSearch, la structure d'indexation répartie est maintenue par les quatre mécanismes *join, leave, takeover* et *maintenance*, où *join* et *leave* sont les procédures de connexion et déconnexion de pair, *takeover* permet de réparer rapidement mais imparfaitement la structure lorsqu'un défaut de pair est détecté, pendant que *maintenance* tourne en tâche de fond et répare les imperfections crées par *takeover*.

Les auteurs de Chord (Stoica et al., 2001) proposent un autre schéma, plus simple mais où la cohérence de l'index réparti est moins forte. On a toujours une méthode *join*, mais plus de procédure de déconnexion. Le couple *(takeover, maintenance)* est remplacé par une procédure de recherche du prédécesseur qui permet à la fois de détecter les départs de pair et de réparer la structure.

Une même application fait donc appel à plusieurs solutions P2P, pour implémenter ses différents services (cf. figure 4.2). Le type d'une solution P2P se détermine en analysant le type des mécanismes P2P qui l'implémentent, nous y reviendrons en détails le moment venu.

4.2 Application de notre vision à l'évaluation des systèmes P2P

Nous pensons que si l'on arrive mal à évaluer les systèmes P2P, c'est parce qu'on a du mal à comprendre leur architecture. Et pour cause, dans un système P2P, les aspects applicatif et système distribué sont très imbriqués. Ils doivent de plus presque systématiquement être modifiés, adaptés l'un à l'autre. Ainsi, très simplement, lorsqu'on « propage des requêtes par rapport à des index », on a dans une même phrase le terme « propage », qui répond à une problématique systèmes répartis ; et le terme « index », dont la nature et la construction sont des problématiques applicatives. On peut aussi

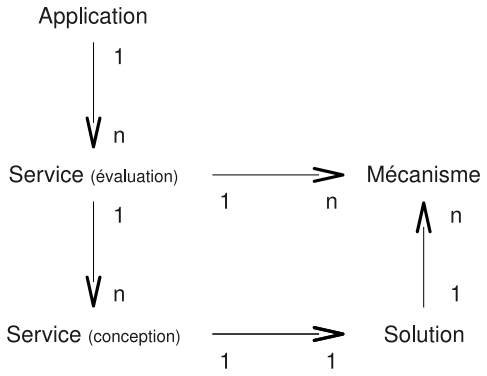


FIG. 4.2 – Une solution P2P est implémentée par un ensemble de mécanismes P2P. Pour les besoins de l'évaluation, nous considérons les mécanismes P2P ; pour les besoins de la conception, nous considérons les solutions P2P.

citer l'exemple du système pSearch (section 5.3), né de la rencontre du système CAN et du modèle vectoriel. On n'y retrouve pas le système CAN, mais seulement une partie - modifiée - de celui-ci. De même, avec le réseau structuré, la gestion des pairs, les *Sampls Sets*, il est difficile de retrouver quelle implémentation du modèle vectoriel - quelle fonctionnalité précise - est remplie par pSearch, indépendemment des biais introduits par le caractère P2P de son implémentation.

Notre idée, c'est que pour faire une évaluation plus efficace d'un système, le principal frein est d'être capable de démêler correctement les fonctionnalités de leur implémentation. Nous proposons donc de faire une sorte de rétro-conception du système évalué, pour ramener les systèmes à un référentiel applicatif commun. Il est impossible de comparer des systèmes P2P si l'on n'est pas capable de les situer précisément l'un par rapport à l'autre, à un même niveau fonctionnel. Notre démarche d'analyse permet cela, en :

Mettre en évidence les service implémentés

: On a ainsi une définition précise des fonctionnalités implémentées par le système. Cela permet d'évaluer la qualité fonctionnelle des applications/services, indépendamment de la qualité de leur implémentation. En évaluant une implémentation centralisée de ces applications/services, on peut notamment donner une borne supérieure de la qualité atteignable, quelle que soit la qualité des mécanismes P2P utilisés.

Mettre en évidence les mécanismes utilisés pour implémenter ces services

: On obtient une liste de mécanismes P2P bas niveau. Cela maximise les chances que ces mécanismes soient utilisés dans d'autres systèmes, et donc de capitaliser l'effort d'évaluation.

Cela permet d'évaluer la qualité de l'implémentation indépendamment de l'application/service implémenté.

Pour faire ce travail de « rétro-conception », nous utilisons un outil que nous appelons *composant*. Nous les définissons ci-après et en faisons une étude plus poussée dans la section 4.4.3.

Définition 3.5 : Composant Un composant est un outil graphique permettant de représenter une brique fonctionnelle d'un système P2P et ses inter-dépendances avec les autres briques fonctionnelles du système. Les composants interagissent en s'échangeant des objets. Les performances d'un composant déterminent la qualité des objets qu'il produit. Cet objet est ensuite consommé par un second composant : les performances du premier composant influent donc sur les performances du second (cf. section 4.4.3).

Nous découpons les systèmes P2P en composants de deux types :

Un composant applicatif représente le programme déployé sur un pair pour assurer un *service*. Une partie de son jeu de données et/ou de ses résultats correspond au jeu de données et/ou au résultat du système global. Dans cette thèse, nous n'étudions que des exemples de systèmes ne contenant qu'un seul composant applicatif, qui décrit le modèle de Recherche d'Information mis en place.

Un composant P2P encapsule un mécanisme P2P. Un mécanisme P2P partage ou calcule un objet avec une certaine erreur, du fait de son fonctionnement distribué. Il introduit donc un biais dans la qualité d'un type d'objets manipulé par les composants applicatifs.

Pour définir le comportement d'un composant applicatif, on a besoin d'introduire la notion de cluster.

Définition 3.6 : Cluster Nous appelons cluster l'image du pair, considéré d'un point de vue applicatif. En effet, pour construire un système P2P, nous avons besoin de définir au niveau applicatif comment les pairs sont pris-en-charge. De ce point de vue applicatif, un pair n'est pas un objet réseau, c'est un groupe de ressources et de requêtes. D'où l'utilisation du terme cluster.

Cette définition cohabite avec la définition traditionnelle « cluster de pairs ». Le sens à attribuer au terme dépend du contexte dans lequel il est utilisé. Pour rappel, si des clusters de pairs sont utilisés, il faut choisir une solution P2P pour la résolution inter-clusters et une solution P2P pour la résolution intra-clusters.

Définition 3.6 : Diagramme d'évaluation Le graphe des composants d'un système est appelé diagramme d'évaluation. Ce graphe spécifie les interfaces d'entrée et de sortie, pour l'évaluation des briques fonctionnelles correspondant aux composants : spécification du jeu de donnés et des configurations en entrée, critères de coût et de qualité à mesurer en sortie. Ce diagramme permet aussi de recomposer les performances d'un système où d'une partie d'un système (e.g une solution P2P) à partir des performances de ses composants.

4.3 Application de notre vision à la conception des systèmes P2P

Bien que l'évaluateur et le concepteur de systèmes P2P puissent être une même personne, il abordent le domaine sous un angle différent. En effet, l'évaluateur considère un système, voir un morceau de système, dont il veut connaître les performances. Le concepteur, lui, part d'une application à implémenter en P2P et se demande quels sont les outils à sa disposition et comment il peut opérer un choix parmi ceux-ci. Il aborde donc le domaine avec une vision plus globale.

Là où ces deux points de vue se rejoignent, c'est que des résultats réutilisables permettent de minimiser l'effort d'évaluation. Et les résultats d'évaluation des composants des systèmes P2P permettent de guider les choix de conception. Dans les deux cas, la clef est de mieux comprendre l'architecture des systèmes P2P, la façon dont les composants s'assemblent pour former un système.

Cette analyse se détache d'une vision « catégories de systèmes » - ex. *pur, structuré, hiérarchique* - pour se rapprocher d'un graphe des systèmes P2P, dans lequel on passe d'un système à l'autre en modifiant un choix de conception.

Pour aider à la conception de systèmes P2P, nous allons donc chercher à décrire de manière **exhaustive** l'ensemble des solutions P2P et l'ensemble des mécanismes P2P disponibles par notre état de l'art. Nous avons vu qu'une implémentation d'une solution P2P est une combinaison de mécanismes P2P. Nous étudions dans le chapitre 5 comment faire l'évaluation de performances d'un mécanisme P2P, et comment combiner les résultats d'évaluation de plusieurs mécanismes pour reconstituer les performances d'un système complexe. En établissant toutes les implémentations possibles de toutes les solutions P2P, nous pouvons donc retrouver tous les rapports qualité/coût atteignables par chaque solution P2P.

On tire de cela deux outils :

1. Si on connaît les performances atteignables par chaque solution P2P, on est en position de choisir la solution la plus adaptée face à un problème donné, décrit par un contexte d'utilisation et les propriétés du jeu de données.

2. Ensuite, dans la mesure où l'on comprend quelle solution est efficace dans quelle situation, on peut positionner les solutions les unes par rapport aux autres, par critères de performance. Et comme on maîtrise quels mécanismes forment l'implémentation qui permet d'atteindre chaque niveau de performances, on dispose d'un bon outil pour comprendre comment chaque mécanisme influence les performances du système global. Vu sous un autre angle, on peut ainsi mieux comprendre comment tel ou tel mécanisme peut compenser telle ou telle « mauvaise » propriété du jeu de données.

Le premier outil est plutôt orienté conception d'un système « from scratch », le second serait plutôt utile pour faire évoluer un système en fonction de caractéristiques mesurées après déploiement. Mais l'idée reste toujours de permettre de faire des choix de conception/re-conception d'un système en fonction des propriétés de l'application, du contexte d'application et du jeu de données.

Pour cela, l'aspect évaluation de performances étant traité dans le chapitre 5, il nous reste à faire un inventaire exhaustif des solutions P2P et mécanismes P2P par rapport à notre état de l'art et comprendre les règles de composition de ceux-ci pour former un système P2P complet, ce que nous traîtons dans le chapitre 6.

4.4 Composants : caractéristiques et utilisation pour l'évaluation

Les composants sont un outil que nous utilisons pour évaluation des systèmes P2P. Avant de présenter notre analyse et processus d'évaluation de PlanetP et pSearch, nous souhaitons donner plus de détails sur les composants, leurs entrées/sorties et la façon dont ils peuvent être assemblés.

Notre approche par composants s'appuie sur une idée de (Schmitz and Löser, 2006), qui introduit l'idée de « Evaluation as a function » (l'évaluation comme une fonction). Les auteurs proposent de voir le système à évaluer comme une fonction. Cette fonction a comme ensemble de départ l'ensemble des paramètres de l'évaluation et comme ensemble d'arrivée l'ensemble des critères de qualité à mesurer. L'essentiel des paramètres d'évaluation, c'est le jeu de données ; les critères de qualité sont généralement présentés sous forme de courbes. Le but de l'évaluation est alors d'expliciter cette fonction, que nous désignerons dans la suite par le terme *fonction caractéristique* du système. C'est-à-dire que l'évaluation vise à comprendre comment les performances du systèmes évoluent en fonction des conditions dans lesquelles il est utilisé. Ces conditions d'utilisation sont décrites par les paramètres de l'évaluation.

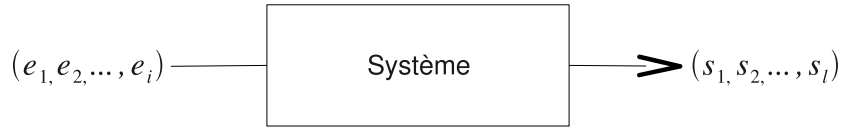


FIG. 4.3 – L'évaluation comme une fonction. Les critères de qualité sont une fonction des paramètres d'entrée : $(s_1, s_2, ..., s_i) = f(e_1, e_2, ..., e_j)$

Les auteurs de (Schmitz and Löser, 2006) opposent cette approche à une approche qui consiste à « se fixer certains critères qui doivent être atteints pour satisfaire l'objectif de l'évaluation » (traduction libre). Les auteurs ne précisent pas plus cette idée, mais nous proposons un exemple qui, il nous semble, l'illustre bien. Nous n'avons pas connaissance que l'évaluation du coût mémoire induit par le système Gnutella ait jamais été faite (occupation du disque dur). Cela se comprend bien, parce qu'on sent intuitivement que Gnutella consomme peu de mémoire, et que donc ce facteur peut difficilement devenir gênant. Mais si l'on prend l'exemple du système PlanetP, tout pair dans ce système connaît possiblement dans ses tables de routage tous les autres pairs du réseau. Et il maintient pour chacun un filtre de Bloom - représentation du contenu du pair - qui fait communément plusieurs milliers de bits. Donc si l'on veut comparer précisément PlanetP et Gnutella, et si l'on veut cerner le seuil à partir duquel il est préférable d'utiliser l'un ou l'autre de ces systèmes, il faut être capable d'exprimer à

quel point Gnutella est économe en ressources mémoire. C'est ce type de situation qui a motivé la mise-en-place de notre méthodologie d'évaluation ; elle permet d'introduire plus de rigueur dans le choix des paramètres à prendre en compte dans l'évaluation d'un système P2P.

4.4.1 En entrée de l'évaluation : notion de paramètre

Nous proposons d'enrichir le modèle de (Schmitz and Löser, 2006) en distinguant dans les paramètres de l'évaluation les notions de *paramètre système* et de *paramètre du jeu de données*. En effet, il existe la plupart du temps (à notre connaissance, toujours) plusieurs façons d'implémenter un même modèle RI-P2P. Prenons l'exemple de Gnutella. Les spécifications ne précisent pas la valeur du *Time To Live* associé aux requêtes. Elles ne précisent pas non plus le nombre de voisins auxquels les requêtes doivent être propagées. À un autre niveau de détails, le système PlanetP (Cuenca-Acuna et al., 2003) met en place un fichier réparti des index de pair. Les mécanismes utilisés pour construire les index et pour les partager sont des paramètres du système. Donc si on pose la question « Comment évaluer Gnutella ? » ou « Comment évaluer PlanetP ? », on ne parle en fait pas d'un seul système, mais d'une classe de systèmes.

Pour introduire cette notion de classe de systèmes dans notre modèle, nous distinguons deux classes particulières d'entrées de l'évaluation :

les paramètres système contrôlent le comportement du système. Le produit cartésien de l'ensemble des valeurs de l'ensemble de ces paramètres, décrit l'ensemble des systèmes d'une classe. Le *Time To live* et la largeur de propagation comptent parmi les paramètres système de Gnutella.

les paramètres du jeu de données sont définis par opposition à cette définition des paramètres. Nous prenons donc le mot « jeu de données » dans son sens large : Il inclut l'ensemble des paramètres *extérieurs au système* qui influent sur ses performances, comme le comportement utilisateur, les ressources et requêtes, les propriétés du réseau...

Cette notion de paramètre brouille la différence entre évaluation et comparaison. En effet, ici nous voulions évaluer un système, mais c'est toute une classe de systèmes qu'on en vient à évaluer : l'ensemble des systèmes décrits par l'ensemble des valeurs que peuvent prendre les paramètres. Donc mécaniquement, on en est conduit à faire la comparaison de ces systèmes. Et on retombe toujours dans ce cas de figure. Évaluer un système P2P, c'est comparer entre eux les éléments d'une classe de systèmes.

En principe, le concepteur d'une application P2P vient avec un certain nombre de spécifications, les plus importantes étant 1) le contexte d'utilisation de l'application, qui donne notamment le jeu de données avec lequel le système va être utilisé et 2) une idée des niveaux de rapport qualité/coût acceptables. Le but de l'évaluation d'une classe de systèmes P2P sera donc de comprendre quelles valeurs il faut assigner aux

paramètres, pour atteindre le rapport qualité/coût requis, dans le contexte donné. Vu autrement, l'évaluation consiste à comparer les niveaux de rapport qualité/coût atteints dans différents contextes, par les éléments de notre classe de systèmes.

Les paramètres sont un modèle du système étudié, puisque l'ensemble des valeurs que peuvent prendre les paramètres décrit l'ensemble des systèmes d'une classe. Le défi serait de considérer qu'il existe une « classe des systèmes P2P ». Il faudrait par exemple que *PlanetP* ou *pSearch* ou *CAN* ou... puisse être un premier niveau de paramétrage, à l'intérieur duquel on retrouve des niveaux de paramétrage plus précis. On aurait ainsi un espace des systèmes P2P. En y appliquant notre approche, le but serait de déterminer le niveau de rapport qualité/coût atteignable par chaque système P2P (et chaque sous-système...), en fonction du jeu de données sur lequel il est utilisé. Nous abordons cette problématique dans le chapitre 6.

4.4.2 En sortie de l'évaluation

Se pose maintenant la question de ce qu'on veut évaluer. Que veut-on obtenir en sortie du simulateur ? Quelles sont les grandeurs caractéristiques d'un systèmes P2P ? La mesure la plus usuelle est sans doute le nombre de messages (souvent exprimé en nombre de sauts). On retrouve aussi la charge mémoire, impactée entre autres par le nombre de voisins que chaque nœud maintient et la complexité des informations concernant chacun d'eux. On peut analyser que toutes ces grandeurs concernent les ressources consommées. Nous les appelons *critères de coût*. Le type de ressources consommé par un système P2P est très lié à sa nature de systèmes réparti - tout système P2P est un système réparti. Du coup, les critères de coût sont des grandeurs transversales, qui concernent tous les systèmes P2P. Nous les analysons en détails dans une première section.

Mais il y a aussi des mesures plus spécifiques à chaque système. Par exemple, si le système est utilisé pour faire de la Recherche d'Information, on sera tenté de construire une courbe Précision/Rappel. Tandis que s'il est utilisé pour faire du partage de fichiers, on préfèrera mesurer le nombre de copies obtenues pour chaque réponse. La différence par rapport aux critères de coût, c'est que cette fois on s'intéresse aux résultats retournés. Et la plupart des modèles P2P peuvent être utilisés pour implémenter plusieurs applications. Nous avons donc toute une série d'indicateurs supplémentaires à évaluer, qui concernent la fonction du système P2P et vont évoluer suivant l'application qu'il implémente. Nous les appelons *critères de qualité* et les étudions dans une seconde section.

4.4.2.1 Les critères de coût

Comme nous l'avons vu, la mesure de coût la plus courante concerne la bande-passante. On la retrouve sous plusieurs formes. Certains comptent le nombre de mes-

sages - ce qui abstrait les différences de taille entre messages. D'autres au contraire descendent à un niveau de détails plus fin, et mesurent par exemple la charge des liens de peering - routeurs reliant les réseaux de plusieurs FAI. Pour faire notre choix, nous considérons qu'un système P2P concerne la couche applicative, au sens du modèle OSI ou du modèle Internet. Nous observons donc le réseau d'un point de vue applicatif. Nous calculons la bande-passante au niveau de la couche applicative ; ce qui correspond à la donnée du nombre de sauts et de la taille des messages - contenu des champs applicatifs. Si on cherche à être plus précis, on entre selon nous dans le domaine de l'*optimisation cross layer*, et nous ne souhaitons pas aborder ce thème ici.

Nous mesurons aussi le coût mémoire induit par le système sur les pairs, c'est-à-dire la complexité des informations à maintenir sur chaque pair pour faire fonctionner le système. Enfin, nous mesurons le coût de calcul induit sur les pairs par le système. Celui-ci s'exprime en termes d'utilisation du processeur et de la mémoire vive.

On peut donc donner dès à présent une liste exhaustive des critères de coût à mesurer pour chaque système :

- nombre de messages (nombre de sauts)
- taille des messages
- coût mémoire
- charge de calcul

4.4.2.2 Les critères de qualité

Comme nous l'avons vu, pour un même système on a deux types de composants :

- un premier composant correspond au module RI installé localement sur chaque pair, qui implémente les spécifications applicatives (modèle IR)
- un ensemble de composants y sont attachés, qui correspondent aux mécanismes répartis utiles à l'implémentation P2P du système

La question des critères de qualité se pose très différemment pour ces deux types de composants.

Nous considérons uniquement des applications de type Recherche d'Information. Nous verrons dans le chapitre 5 que nous analysons plusieurs étapes de modification des spécification applicatives, pour aller des spécifications implémentables sur un serveur aux spécifications du module de comportement local, implémentables sur chaque pair d'un réseau P2P. Ces étapes de modification des spécifications correspondent par exemple à l'approximation de l'IDF sur le corpus local dans PlanetP, à l'utilisation d'une structure de données des descripteurs de ressources pour la résolution des requêtes où à la reclustérisation des ressources sur les pairs.

Toutes ces modifications du modèle RI ; qu'elles soient nécessaires (ex. intégrer la notion de pair) ou optionnelles (compresser les veteurs sémantiques en filtres de Bloom) peuvent être évaluées comme des options d'un système RI classique : on construit deux

implémentations, l'une uilisant cette composante et l'autre non. On évalue chaque implémentation par comparaison des résultats obtenus à ceux fournis par un oracle (courbe précision/rappel). Comme nous l'avons sous-entendu, toutes les modifications peuvent être implémentées aussi bien dans un système centralisé que P2P, c'est pourquoi nous utilisons le terme de spécifications.

Par contre, les spécifications applicatives peuvent être implémentées en P2P de différentes manières. Il y a par exemple plusieurs façons de gérer la cohérence d'un index global des pairs ou de maintenir le réseau logique (overlay). Nous verrons comment l'implémentation des spécifications applicatives s'appuie sur des mécanismes P2P. Un mécanisme P2P gére le partage d'un objet défini par les spécifications applicatives ; soit pour maintenir sa cohérence, soit pour le calculer de façon répartie.

Il est donc possible de paramétrer une implémentation centralisée des spécifications applicatives par la qualité des mécanismes P2P qui l'implémentent. On abstrait ainsi le mécanisme P2P. Les performances de cette implémentation centralisée paramétrée peuvent être exprimées en termes de Précision/Rappel/temps. La notion de temps est introduite par le temps qu'il faut aux mécanismes P2P pour propager/router/maintenir les objets manipulés par les spécifications. Les critères de qualité des mécanismes P2P sont ajoutés au entrées du composant applicatif comme paramètres du jeu de données.

Parallèlement, on peut évaluer les performances des mécanismes P2P pour remplir le rôle défini par les spécifications. Leurs critères de qualité devront être définis de façon ad-hoc, suivant le rôle qu'ils remplissent. On verra qu'il peut être aussi nécessaire de paramétrer leur implémentation par certaines propriétés des spécifications. Par exemple, la qualité d'un mécanisme de gossiping varie suivant la taille des objets partagés ; si les descripteurs sont compressés par la technique des filtres de Bloom, ils sont plus faciles à partager.

4.4.3 En résumé : notion de composant

Le diagramme 4.4.3 présente la fonction caractéristique dans laquelle on a différencié les deux types d'entrées et les deux types de sorties. On voit que cela ressemble plus à un composant qu'à une fonction. Et pour cause, nous allons voir dans la section suivante que les fonctions caractéristiques se composent.

4.4.4 Le diagramme d'évaluation

Nous découpons les systèmes P2P pour les évaluer par morceau. Reste le problème de reconstituer la fonction caractéristique d'un système à partir des fonctions caractéristiques des morceaux. Cela se fait en associant les paramètres de sortie d'une fonction aux paramètres d'entrée d'une autre. Reprenons l'exemple de PlanetP. On a vu que

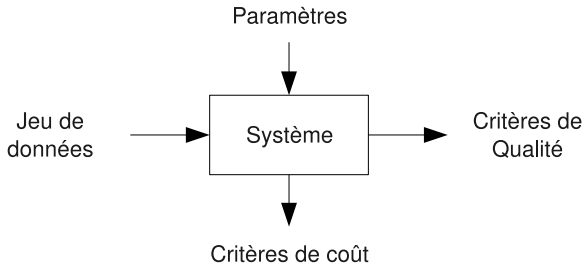
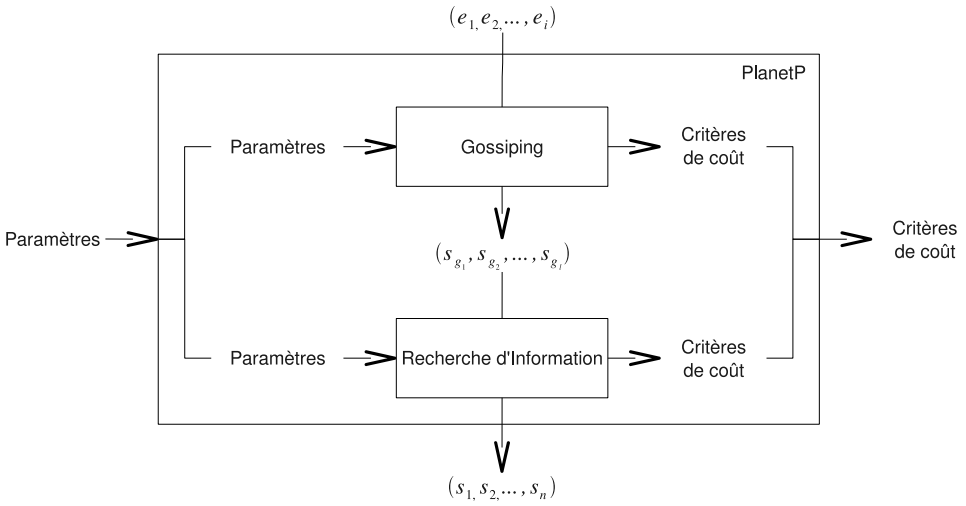


FIG. 4.4 – L'évaluation d'un système P2P explicite la valeur des critères de coût et les critères de qualité en fonction des paramètres et du jeu de données

la qualité de l'index des pairs agit sur la qualité de la recherche d'information. Donc nous proposons, dans notre implémentation du modèle RI, de paramétrer la qualité de l'index des pairs. Ainsi, on peut donner en argument de son évaluation les indicateurs de qualité de l'index, fournis par l'évaluation du mécanisme de gossiping : des critères de qualité du gossiping deviennent des paramètres du modèle RI. Graphiquement, nous le représentons comme suit :



Nous appelons ce type de représentation *diagramme d'évaluation*. Il décrit en fait le plan d'évaluation d'un système P2P. Bien sûr, ici le problème est extrêmement simplifié. En réalité, les fonctions caractéristiques forment des graphes plus complexes. Pour les construire, nous avons besoin d'une procédure moins empirique qui permette une analyse plus systématique des systèmes étudiés, et permette d'obtenir un diagramme plus fiable. Nous ne donnons pas ici une méthodologie précise. Par contre, nous énonçons les principes fondateurs et nous construisons les diagrammes d'évaluation des systèmes de notre état de l'art à titre d'exemple.

4.5 Synthèse

Notre objectif est donc de mettre en place une méthodologie permettant de découper les systèmes RI-P2P, de construire les fonctions caractéristiques des éléments et à partir de là de construire un scénario d'évaluation.

Pour cela, nous considérons qu'un systèmes RI-P2P est avant tout un système RI. La RI, c'est l'application, le service concrètement rendu ; le P2P, c'est un moyen, une façon d'implémenter le service, en principe (dans l'idéal) transparente pour l'utilisateur. Du coup, pour analyser les systèmes RI-P2P, nous pouvons extraire le modèle RI de base, et analyser comment il a été transformé pour être implémenté en P2P. Cette démarche fait émerger les mécanismes P2P en jeu. Au final, les diagrammes d'évaluation obtenus présentent un modèle RI paramétré par un ensemble de mécanismes répartis.

Pour chaque composant du diagramme d'évaluation, son évaluation nous permet de remplir une table de performances. Cette table donne, suivant les paramètres système et les paramètres du jeu de données, les niveaux obtenus pour les critètres de qualité et de coût, comme sur la figure 4.5.

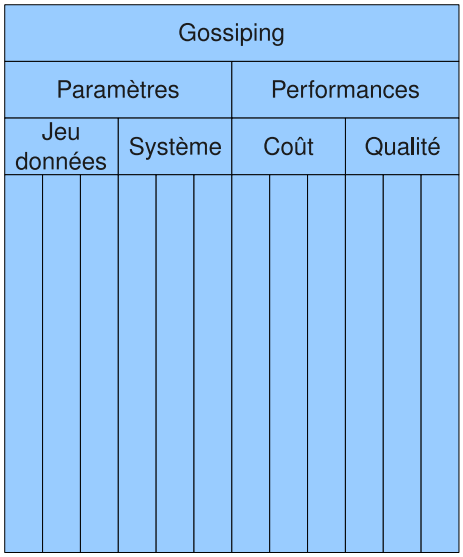


FIG. 4.5 – Le résultat de l'évaluation d'un composant est une table donnant les rapports qualité/coût atteignables en fonction des paramètre de jeu de données et paramètres systèmes.

À partir de ces tables, on peut reconstituer les performances du système global en faisant une fermeture transitive, comme indiqué sur la figure 4.6. En effet, un mécanisme réparti construit/partage un objet réparti de façon plus ou moins optimale ; cet objet est ensuite « consommé » par l'implémentation locale du modèle RI : son niveau de qualité devient donc un élément de jeu de données du point de vue de ce dernier module.

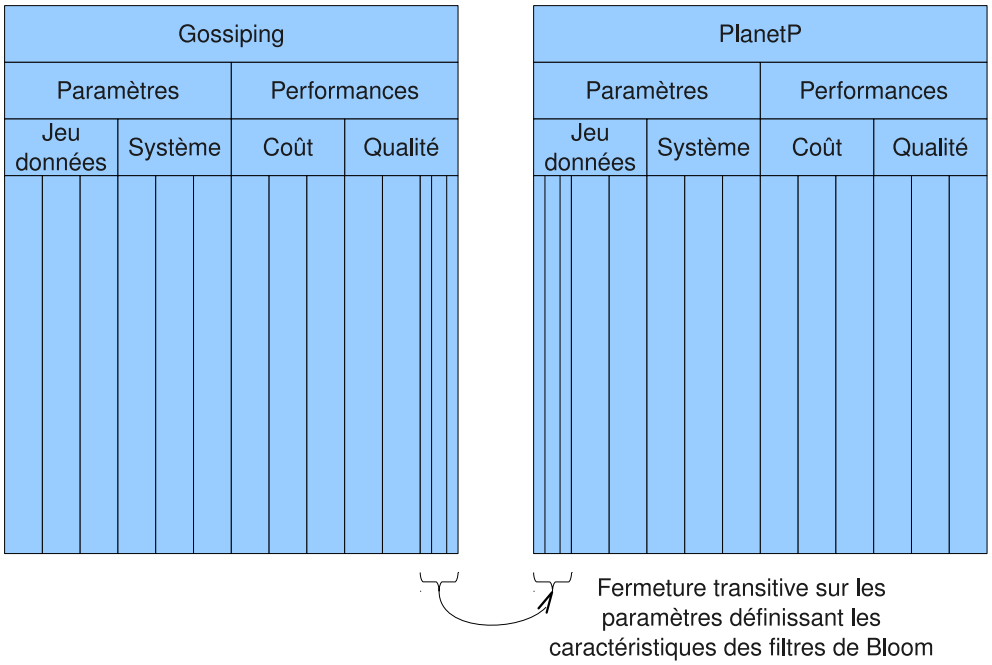


FIG. 4.6 – Certains critères de qualité du mécanisme de gossiping correspondent à des paramètres de jeu de données du composant applicatif de PlanetP.

Chapitre 5

Vers une démarche systématique d'évaluation : Un plan d'évaluation pour PlanetP et pSearch

Dans ce chapitre, nous présentons notre démarche d'évaluation des systèmes P2P. Celle-ci permet, à partir de l'étude d'un système - spécifications, diagrammes UML, code... - de construire un *diagramme d'évaluation*. Le diagramme d'évaluation d'un système est un plan d'évaluation. Il présente les composants du système, les interactions entre ces composants, et surtout les paramètres en entrée et en sortie de l'évaluation de chaque composant.

Ce chapitre ne présente pas à proprement parler une méthodologie d'évaluation des systèmes P2P, dans le sens où nous ne donnons pas un algorithme prenant en entrée un système et retournant son diagramme d'évaluation. Il s'agit plus d'une démarche, soutenue par ensemble d'outils, et motivée par des exemples.

Notre parti-pris est de fonder la discussion sur deux exemples : PlanetP (Cuenca-Acuna et al., 2003), décrit à la section 5.1 et pSearch (Tang et al., 2003), décrit à la section 5.3. Le choix de ces deux systèmes est motivé par ce qu'ils sont très différents ; et permettent, sur deux exemples seulement, de présenter un large éventail des techniques P2P.

Pour chacun des deux systèmes, nous procédons tout d'abord à une analyse, qui amène deux résultats. En premier lieu, nous extrayons une description précise du modèle RI implémenté par ces deux systèmes. Nous l'appelons *modèle RI « P2P-isable »*, c'est

la spécification la plus complète qui puisse être implémentée aussi bien en centralisé qu'en P2P. Elle spécifie notamment comment sont gérés des clusters de ressources et requêtes, assimilables aux pairs dans une implémentation P2P, aux utilisateurs dans une implémentation centralisée. Le *modèle RI P2P-isable* est en quelques sortes le PGCD entre un sysème P2P et un système centralisé.

Le second résultat, c'est qu'on a une décomposition précise de l'ensemble des modifications qui conduisent du modèle RI centralisé initial (VSM) au *modèle RI P2P-isable*, puis à son implémentation en P2P. L'impact de ces modifications, où de leur absence, peut être évalué dans une démarche d'évaluation classique, calculant une courbe précision/rappel par rapport à un oracle construit manuellement. Cette analyse permet aussi de situer précisément les deux systèmes l'un par rapport à l'autre, de bien comprendre leurs points communs et leurs différences.

Dans un second temps, sur la base de cette analyse/décomposition, nous construisons le diagramme d'évaluation de chacun des deux systèmes. Le diagramme d'évaluation spécifie l'évaluation modulaire d'un système ; de telle façon que l'évaluation du *modèle RI P2P-isable* soit paramétrée par la qualité des mécanismes répartis et que l'évaluation des mécanismes répartis soit paramétrée par les propriétés du modèle RI. Cette façon d'évaluer favorise la réutilisation de résultats et permet de mieux comprendre à quoi sont dûes les bonnes/mauvaises performances d'un système.

5.1 Analyse de PlanetP

5.1.1 Introduction

Dans cette section, nous décrivons le système PlanetP (Cuenca-Acuna et al., 2003). PlanetP est un système de Recherche d'Information en peer-to-peer (P2P-IR). Notre description repose sur l'idée que PlanetP doit être compris comme la rencontre entre un système de Recherche d'Information (RI), le modèle des filtres de Bloom, et une *technique de répartition en P2P*, basée sur un fichier réparti des pairs maintenu par *gossiping*. D'autres modèles RI, et plus généralement d'autres modèles applicatifs, peuvent être utilisés. Parallèlement, d'autres techniques de répartition ont été conçues. Nous souhaitons donc mettre en relief le processus de répartition appliqué par les concepteurs de PlanetP, afin d'être en mesure de comparer les différentes techniques de répartition.

Dans un premier temps, nous décrivons le modèle vectoriel (de Recherche d'Information). Celui-ci forme le modèle de Recherche d'Information à la base de PlanetP. Puis nous décrivons la technique du gossiping. Il s'agit d'un mécanisme permettant la propagation d'une donnée sur un réseau P2P par échanges de proche en proche. Elle est utilisée dans PlanetP pour maintenir une cohérence faible sur un fichier réparti des pairs.

Dans son implémentation standard, le modèle vectoriel est pensé comme un système centralisé. PlanetP propose de le répartir en P2P. La technique utilisée pour ce faire est de construire un fichier réparti des index de pair, dont une cohérence faible est maintenue par un mécanisme de gossiping. Dans le processus qui mène du modèle vectoriel à PlanetP, nous distinguons trois étapes :

1. *Intégrer la notion de clusters au modèle RI*, avec l'indexation de clusters et la résolution inter-clusters. Tous deux forment une extension naturelle du modèle des filtres de Bloom.
2. *Adapter le modèle RI au gossiping.* Nous distinguons une série de modifications qui visent à adapter le modèle RI clustérisé aux conditions de la technique de répartition. Dans PlanetP, il s'agit de compresser l'index des pairs par la technique des filtres de Bloom ; et d'introduire l'IPF, qui permet d'approximer l'IDF à partir de l'index réparti des pairs mis-en-place par gossiping dans le modèle de répartition.
3. *Appliquer la technique de répartition.* Il s'agit là de spécifier une implémentation P2P de chaque méthode du modèle sous-jacent. C'est le *modèle de répartition*, qui achève la description du système PlanetP.

5.1.2 Le modèle vectoriel

PlanetP est construit sur la base du modèle vectoriel (VSM, Vector Space Model) tel que décrit dans (Witten et al., 1999). Ressources et requêtes y sont représentées par des vecteurs réels. Chaque dimension représente un terme. La coordonnée d'un objet dans une dimension représente la pertinence du terme pour décrire la ressource. Différentes options sont disponibles (et combinables) pour optimiser la liste des termes caractéristiques (et donc le nombre de dimensions), parmi lesquelles la lemmatisation, dont le principe est de ne retenir que la racine des mots, ou les listes stop-word, qui permettent de retirer les mots peu porteurs de sens, comme les prépositions, conjonctions etc... Pour plus de précisions sur le sujet, voir (Berry et al., 1999).

Pour calculer la pertinence d'une ressource par rapport à une requête, on utilise une mesure d'IDF (Inverse Document Frequency). Le principe est que plus un mot est rare, plus il discrimine de façon importante la sémantique d'une ressource ; tandis qu'à l'opposé, les mots très courants sont moins porteurs de sens. L'IDF d'un terme est donc inversement proportionnel à la fréquence du terme dans le corpus ; il est calculé suivant la formule suivante :

$$IDF_t = log(1 + \frac{nombre\ de\ documents\ dans\ le\ corpus}{nombre\ de\ documents\ qui\ contiennent\ t\ dans\ le\ corpus})$$

Il est utilisé dans la mesure de similarité comme suit :

$$Sim(Q, R) = \frac{\sum_{t \in Q} IDF_t \times (1 + log(frequence\ de\ t\ dans\ R))}{\sqrt{|R|}}$$

On peut résumer en disant que le modèle vectoriel constitue le modèle RI à la base de PlanetP, en fournissant quatre méthodes (les arguments soulignés sont les objets dont l'ajout, suppression ou modification déclenche l'appel de méthode) :

- $vsm.IndexResource : \underline{Resource\ r} \rightarrow Resource\ rs$: indexation des ressources
- $vsm.IndexQuery : \underline{Query\ q} \rightarrow Query\ q$: indexation des requêtes
- $vsm.Sim : \underline{Query\ q} * Resource\ r * IDF \rightarrow float$: mesure de similarité
- $vsm.IDF : \underline{Resource\ r} \rightarrow IDF$: met à jour l'IDF - gère l'ajout et la suppression d'une ressource

On peut construire un diagramme d'activités UML correspondant (figure 5.1).

Il est clair que le diagramme ci-dessus ne décrit pas le système PlanetP. Pour le décrire, il faut étoffer notre modèle. Pour cela, nous posons tout d'abord la technique P2P utilisée pour répartir le modèle RI de base : on crée un fichier partagé des index de pair, dont la cohérence est assurée par un mécanisme de gossiping. Ensuite, nous précisons par étapes comment le modèle RI de base est étendu pour inclure ce mécanisme P2P.

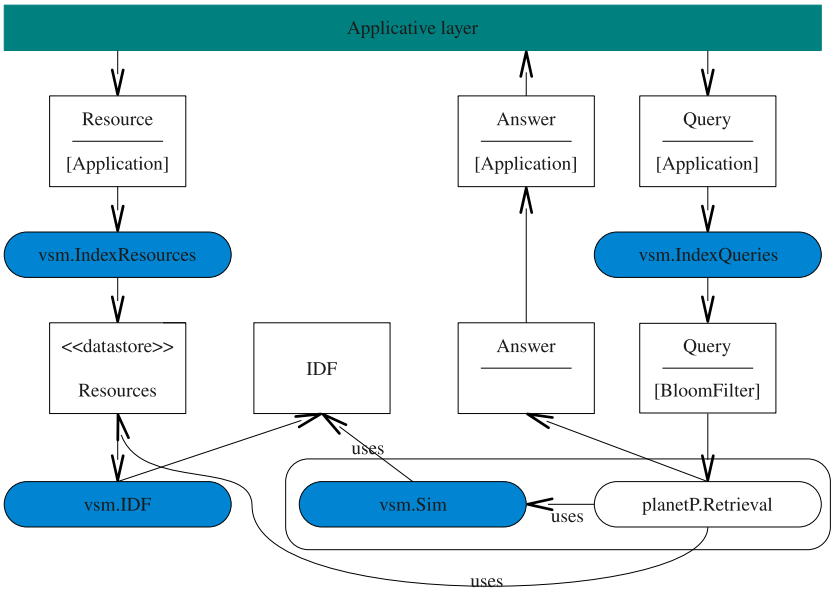


FIG. 5.1 – PlanetP : Modèle de Recherche d'Information, 1^{ere} étape

5.1.3 Le gossiping

Le gossiping est un terme générique désignant une classe de mécanismes répartis permettant la propagation d'un objet sur un réseau par échanges de proche-en-proche. Gossiping est un terme anglo-saxon signifiant rumeur : il permet de partager un objet par propagation épidémique. Dans PlanetP, ce mécanisme est utilisé pour maintenir la cohérence d'un fichier réparti des index de pairs. Chaque index étant propagé par gossiping sur le réseau, le système tend à ce que tous les pairs connaissent tous les index du réseau.

Le mécanisme utilisé par PlanetP se base sur le mécanisme décrit par (Demers et al., 1987), auquel a été ajouté un mécanisme *anti-entropie*. Il est donc constitué de trois mécanismes, *Push*, *Pull* et *partial-pull* (anti-entropie). Lorsqu'un filtre de Bloom est créé ou mis-à-jour, une *rumeur* correspondante est créée. Chaque pair p prend-en-charge ses rumeurs de la manière suivante :

- **Push** : toutes les T_g secondes, p envoie la rumeur à un voisin choisi aléatoirement. Si celui-ci n'a pas déjà reçu cette rumeur, il l'enregistre et la prend-en-charge de la même façon que p. Un pair cesse de propager une rumeur après que n voisins contactés consécutivement l'ont déjà entendue.
- **Pull** : tous les T_r cycles de *Push*, au lieu de faire un *Push*, on fait un *Pull*. Il s'agit de demander à un voisin choisi aléatoirement un résumé de ses rumeurs pour pouvoir en extraire et lui demander, les rumeur qu'on ne connaît pas. T_r diminue

automatiquement tant qu'aucune rumeur n'est propagée. Il est immédiatement réinitialisé dès qu'une nouvelle rumeur est émise.

- **Partial-Pull** (anti-entropie) : ce mécanisme consiste à envoyer avec la réponse au *Push* un résumé de ses m rumeurs les plus récentes. Le pairs qui a envoyé le *Push* traite la réponse comme un résumé reçu d'un *Pull*.

Il existe d'autres types de gossiping, tel le mécanisme Newscast (Jelasity et al., 2003). Ce mécanisme est plus simple : sur chaque pair, toutes les T_g secondes, on choisit un voisin aléatoirement et on échange avec lui ses dernières nouvelles (l'échange est bi-latéral). Newscast est plus économe en nombre de messages, mais comme ceux-ci sont plus gros, il est globalement plus gourmand en bande-passante. Il peut être intéressant dans les réseaux actuels qui privilégient la bande-passante à la latence.

5.1.4 Introduire la notion de cluster

Nous avons donc vu que PlanetP était construit sur la base du modèle vectoriel ; réparti en construisant un fichier partagé des index de pair, maintenu par un mécanisme de gossiping. Mais à ce stade, impossible de mettre en place le fichier réparti des index de pair. Et pour cause : dans le modèle VSM tel que nous venons de le décrire, il n'y a pas de notion de pair.

Pour l'instant, nous définissons un modèle de Recherche d'Information. On ne s'occupe donc pas des aspects réseau, donc on ne gère pas des pairs à proprement parler. On retiendra simplement que ressources et requêtes sont clustérisés a priori : des groupes de requêtes et ressources sont émis par les mêmes pairs ; toute ressource appartient à un groupe (elle est publiée par un pair), idem pour les requêtes, et il existe une bijection entre les groupes de requêtes et les groupes de ressources. Et ce qui nous intéresse en RI, ce n'est pas tant l'unicité de pair, mais surtout la probable unicité d'utilisateur entre les objets d'un même groupe.

Donc on va étoffer le modèle RI de base pour y ajouter la notion de cluster et les méthodes de gestion associées. Ceci doit être fait avec l'idée d'adapter notre modèle RI à la technique P2P utilisée. On rajoute donc une méthode d'indexation des clusters, une méthode de résolution des pairs et une méthode de fusion des résultats obtenus sur plusieurs pairs :

- $vsm.IndexResource : \underline{Resource\ r} \rightarrow Resource\ r$
- $vsm.IndexQuery : \underline{Query\ q} \rightarrow Query\ q$
- $vsm.IndexCluster : \underline{Cluster\ c} * ResourceSet\ rs \rightarrow Clusterc$: indexation des clusters
- $vsm.Sim : \underline{Query\ q} * Resource\ r * IDF \rightarrow float$
- $vsm.SimCl : \underline{Query\ q} * Cluster\ cl * IDF \rightarrow float$: mesure de similarité entre une requête et un cluster
- $vsm.IDF : \underline{Resource\ r} \rightarrow IDF$

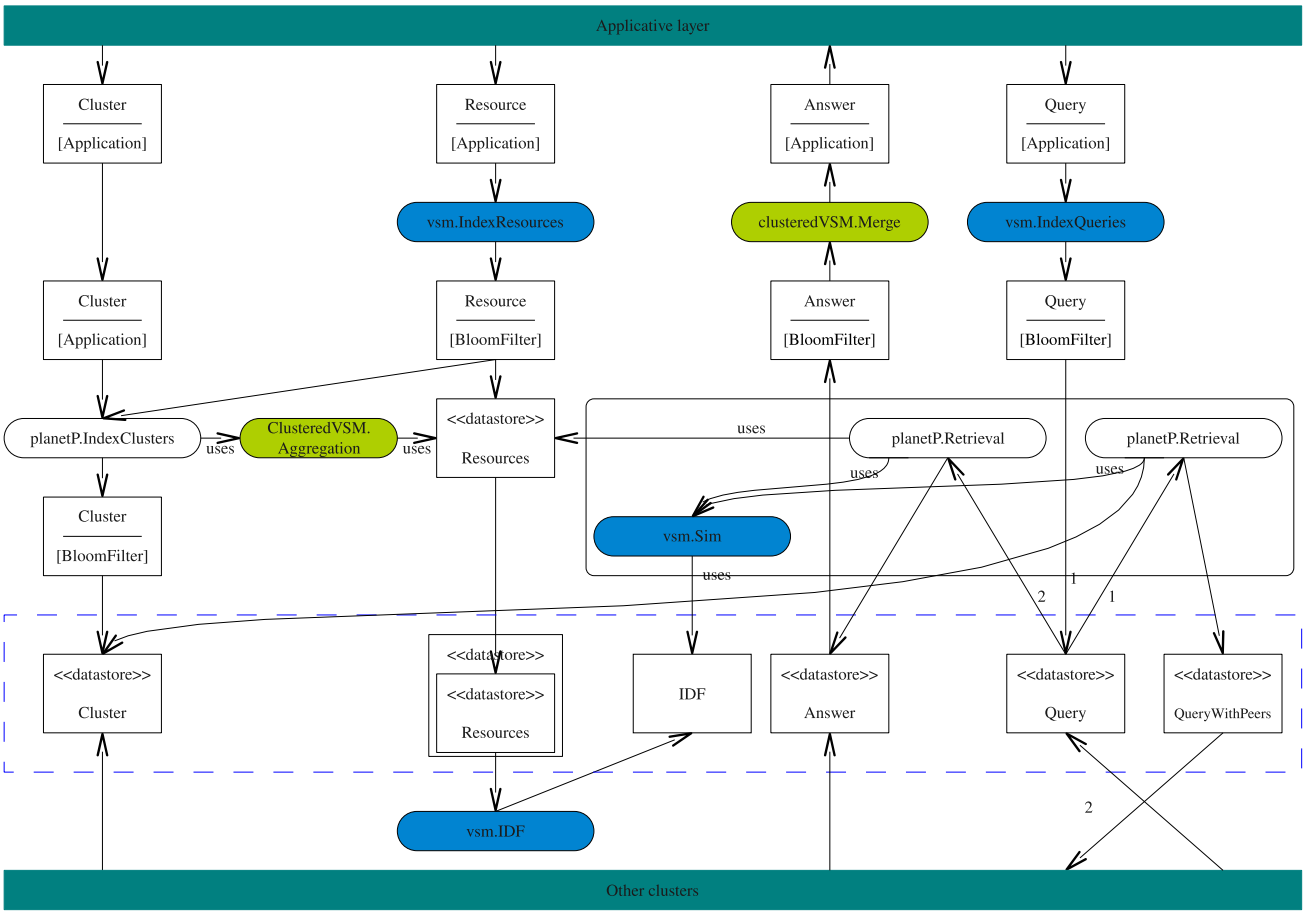


FIG. 5.2 – PlanetP : Introduction de la notion de cluster

On peut de nouveau construire un diagramme d'activités UML pour se rendre compte de la façon dont ces méthodes sont utilisées pour former un modèle RI. Le diagramme 5.2 représente en fait le traitement des objets d'un seul cluster. Les liens entre les différents clusters, c'est un ensemble d'objets partagés entre les clusters, représentés dans un rectangle en pointillés. Une exception cependant : la méthode de calcul de l'IDF est transversale aux clusters (appliquée globalement, une seule fois).

5.1.5 Adaptations à l'environnement P2P

Notre modèle VSM-clustérisé pourrait être réparti en P2P en l'état. Cependant, les auteurs de PlanetP lui ont appliqué deux modifications, pour l'adapter aux conditions réparties :

- L'index partagé des pairs est compressé par la méthode des filtres de Bloom
- L'IDF est remplacé par l'IPF

Nous allons tout de suite voir plus précisément en quoi consistent ces modifications.

Filtres de Bloom Les filtres de Bloom permettent de représenter de façon compacte un ensemble de termes. Ils ont été utilisés en Recherche d'Information pour représenter les termes caractéristiques des ressource et des requêtes, avec l'adjonction d'une fonction d'évaluation (Witten et al., 1999). Un filtre de Bloom se présente sous la forme d'un vecteur de bits. à chaque terme correspond une séquence de bits à 1. Pour le construire, on utilise un ensemble de k fonctions de hachage. L'application de ces fonctions sur le terme indexé désigne les k bits qui doivent être mis à 1. Ainsi, le filtre de Bloom d'un ensemble de termes peut être construit en calculant le OR logique des filtres de Bloom des termes (voir figure 5.3).

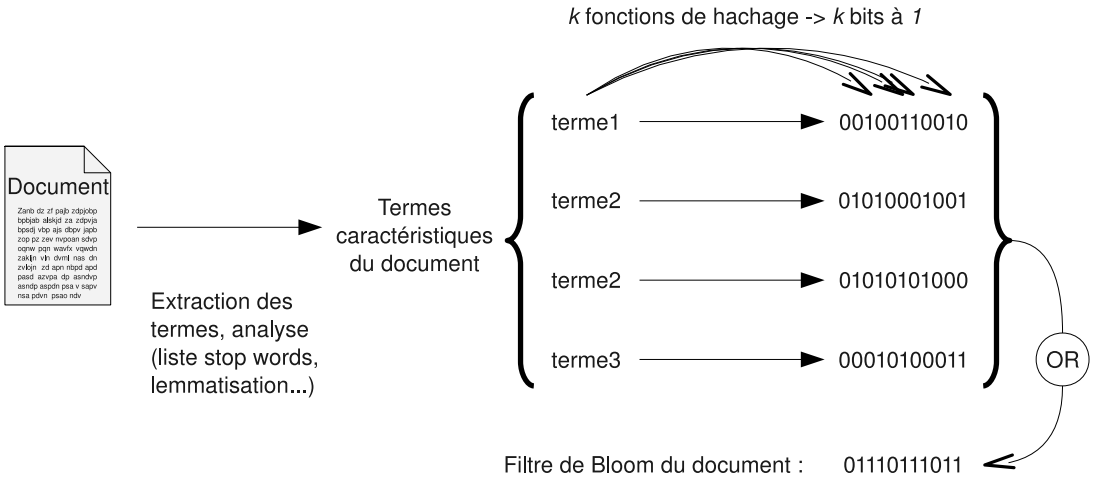


FIG. 5.3 – Construction du filtre de Bloom d'un document

L'utilisation des filtres de Bloom pour indexer documents et requêtes impose de :

- modifier la fonction de similarité. Utilisée sur des filtres de Bloom, elle devient

$$Sim(Q,R) = \sum_{t \in Q, t \in R} IDF_t$$

  En fait, c'est exactement la même fonction ; mais comme on sait qu'elle est utilisée sur des vecteurs de bits et non des vecteurs d'entiers, on peut la simplifier ; le principal bénéfice étant une économie de ressources de calcul.
- adapter le calcul de l'IDF : désormais, on calculera l'IDF d'un bit au lieu de l'IDF d'un terme, selon la formule

$$IDF_b = log(1 + \frac{nombre\ de\ documents\ dans\ le\ corpus}{nombre\ de\ fois\ ou\ b\ est\ a\ 1\ dans\ le\ corpus}$$

- adapter la méthode d'agrégation qui permet de construire l'index d'un pair à partir des index de ses documents. On utilise le OR logique, qui servait jusque là à calculer le filtre de Bloom d'un document à partir des filtres de Bloom des mots qui le composent.

L'IPF L'IPF est tout simplement un IDF calculé sur les index de pair au lieu des index de ressources. Cela permet de le calculer indépendamment sur chaque pair, à partir de l'index réparti des pairs.

Remarque et conclusion Notez que la méthode *Merge* reste inchangée. En effet, cette méthode utilise des mesures de similarité calculées antérieurement. Peu lui importe que ces mesures soient calculées à partir du modèle vectoriel « standard » ou de filtres de Bloom. La seule contrainte est que ces mesures soient calculées suivant un référentiel commun ; ce qui est supposé être le cas dans PlanetP. En effet, l'IPF est calculé à partir de l'index partagé des pairs. Les copies de cet index étant supposées les mêmes sur tous les pairs, l'IPF doit être le même sur tous les pairs.

Suite à ces adaptations, on obtient le véritable modèle RI sous-jacent à PlanetP (cf. figure 5.4). C'est-à-dire qu'il ne manque plus pour définir PlanetP qu'à définir un ensemble de mécanismes P2P permettant soit de gérer la répartition d'une variable partagée, soit l'exécution répartie d'un algorithme. C'est ce que nous voyons dans la prochaine section.

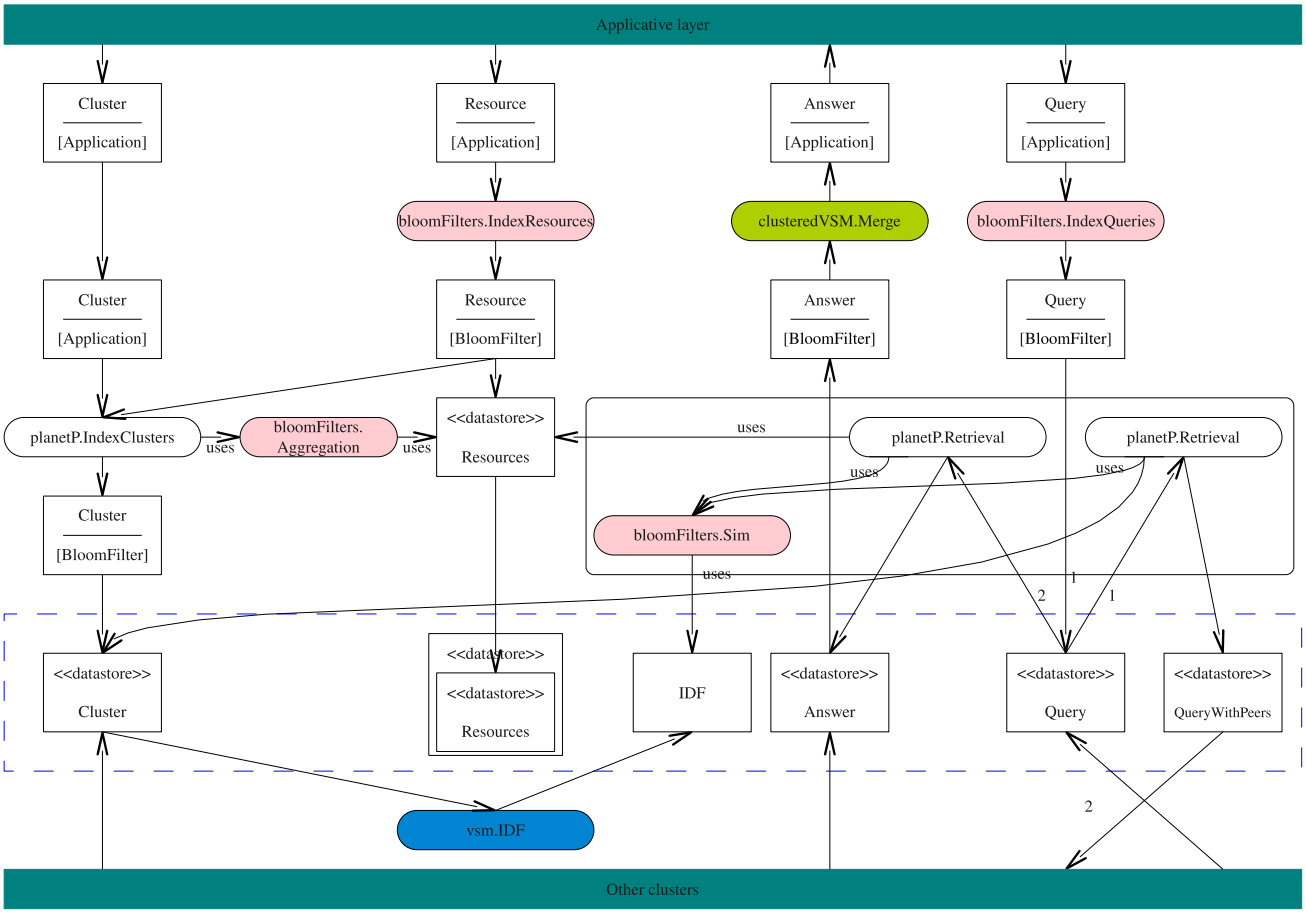


FIG. 5.4 – Diagramme d'activités de PlanetP

5.1.6 Modèle de distribution de PlanetP

À ce stade, le modèle RI mis en place par PlanetP est défini. Il ne manque que des mécanismes répartis pour compléter la description du système complet. Nous avons recensé deux types de mécanismes. Soit on répartit les données, en quel cas le mécanisme de répartition s'apparente à un mécanisme de gestion de cohérence d'une donnée dans un système réparti. Soit on distribue le calcul, en quel cas la donnée est créée participativement par un mécanisme de calcul distribué (parfois, cela peut s'apparenter à du calcul parallèle).

Concrètement, cela correspond à ajouter sur le diagramme de classes une ligne supplémentaire, dans laquelle des mécanismes répartis prennent en charge ces objets (figure 5.5). Donc simplement en lisant le diagramme 5.4, on voit que les objets partagés entre clusters sont l'ensemble des clusters (*«datastore» Cluster*), les réponses (*«datastore» Answers*), et les requêtes pour les deux étapes de résolution (*«datastore» Query* et *«datastore» QueryWithPeers*). Ici, on n'a que des objets répartis - donc pas d'algorithme réparti :

- L'idée de base de PlanetP, c'est d'utiliser un mécanisme de gossiping pour partager le magasin des clusters.
- Du coup, chaque cluster connaît l'ensemble des clusters et peut calculer l'IPF localement.
- Concernant les requêtes, comme le magasin des clusters est distribué, on peut effectuer la résolution inter-clusters localement sur chaque pair. Le seul mécanisme réparti nécessaire est alors une simple méthode d'envoi de la requête aux pairs sélectionnés. Celle-ci déclenche l'évaluation intra-clusters.
- L'évaluation intra-clusters utilise les ressources locales à chaque cluster et l'IPF, dont on a vu qu'il était disponible sur tous les clusters. Elle peut donc être exécutée localement elle aussi.

Voilà comment notre diagramme d'activités évolue :

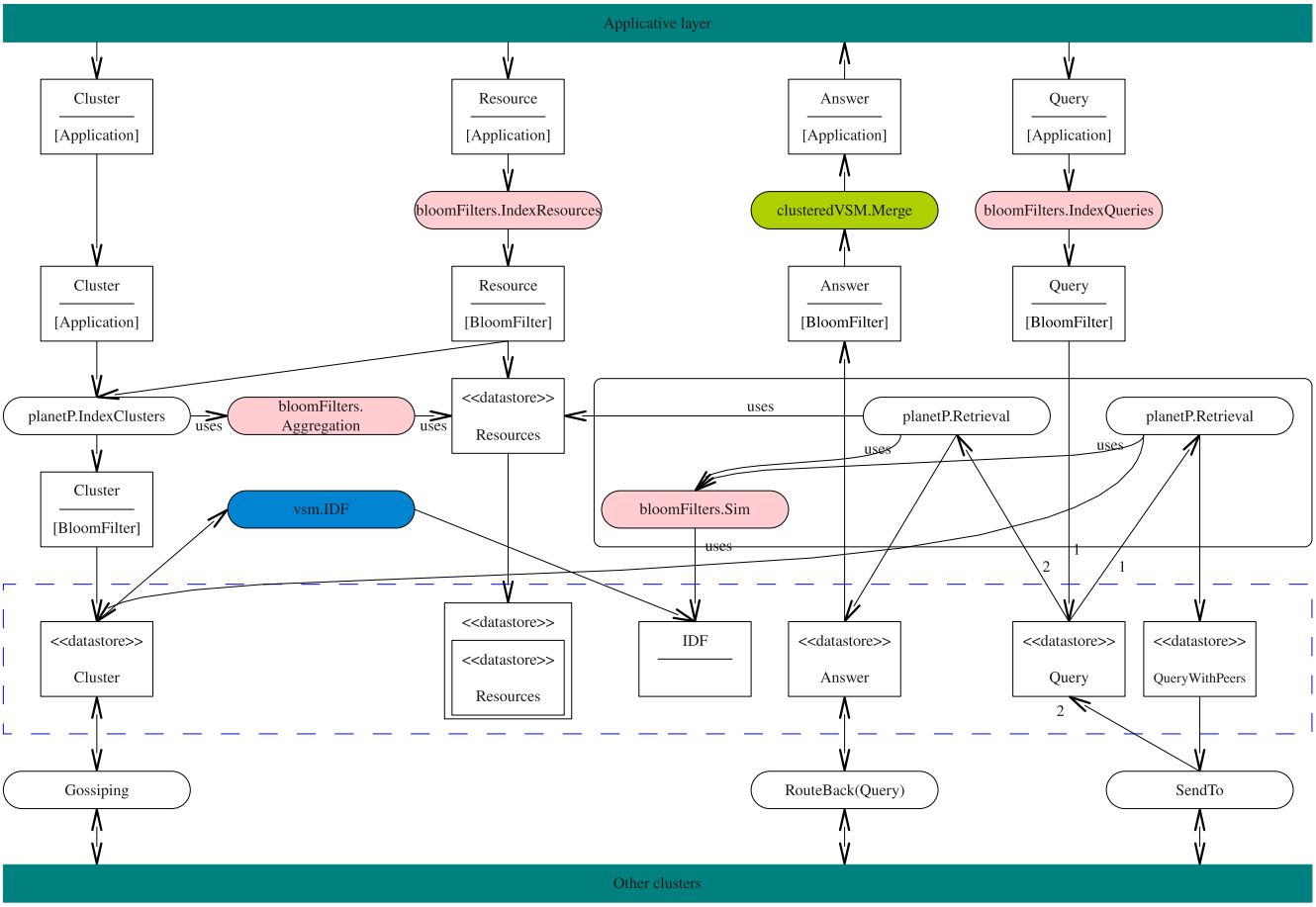


FIG. 5.5 – Diagramme d'activités de PlanetP

5.1.7 Précision sur les mécanismes répartis

Notez la différence de complexité entre les mécanismes de partage des requêtes et réponses, et le mécanisme de gossiping (diagramme d'activités reporté ci-dessous). Nous dirons que le gossiping est un *mécanisme distribué*, qui utilise plusieurs *algorithmes distribués* ; tandis que les mécanismes de partage des requêtes et réponses sont directement des algorithmes distribués. Nous appelons algorithme distribué un ensemble de deux méthodes, qui définissent la façon dont chaque pair prend-en-charge *un seul type de messages* ; ces deux méthodes définissent les comportements lorsqu'un message est reçu respectivement depuis la couche basse (réseau physique) et depuis la couche haute (application).

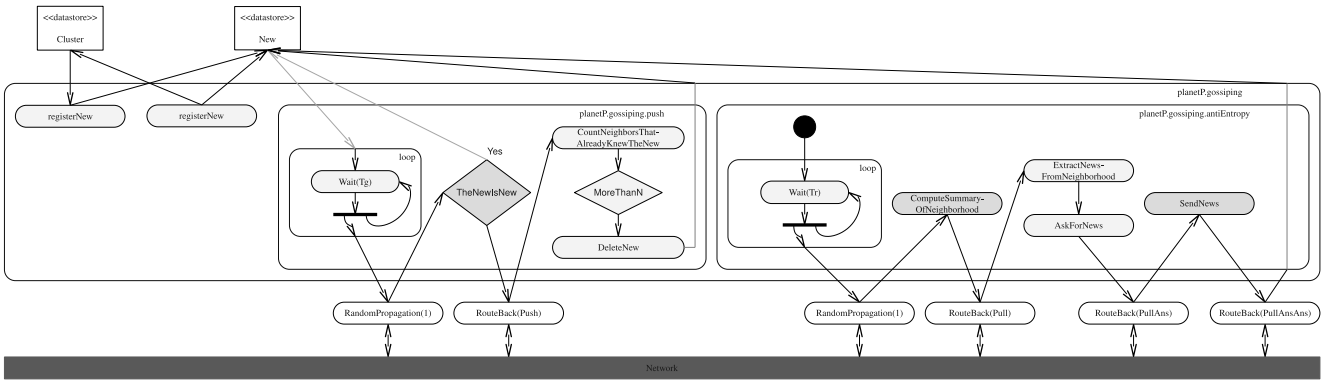


FIG. 5.6 – Diagramme d'activités de PlanetP : détails du mécanisme de Gossiping.

5.2 Plan d'évaluation de PlanetP

5.2.1 Les objets en jeu

Pour résumer, jusqu'à présent nous avons montré comment PlanetP pouvait être vu comme une implémentation P2P du modèle vectoriel. Pour construire cette implémentation, il a fallu transformer le modèle original : intégrer la notion de pair, compresser les index par la technique des filtres de Bloom, et remplacer l'IDF par l'IPF. Puis nous avons ajouté trois mécanismes répartis, permettant de prendre en charge les trois objets partagés entre les pairs, à savoir l'index des clusters, les requêtes et les réponses. Cette analyse nous a permis de construire un diagramme d'activités de PlanetP (diagramme 5.4).

Nous allons maintenant nous appuyer sur cette description du système pour construire son plan d'évaluation.

Le point de départ est relativement simple. On part de l'observation du diagramme d'activités. Celui-ci présente un modèle RI et trois mécanismes répartis : cela nous donne quatre composants. Les liens entre les composants, ce sont les objets qu'ils échangent : la qualité d'un résultat délivré par un algorithme dépend des propriétés des paramètres qui lui sont donnés en entrée. Ainsi :

- la qualité du mécanisme *Gossiping* impacte le comportement du modèle RI parce qu'il conditionne le rappel et la précision de l'index des pairs ;
- la qualité du mécanisme *RouteBack* impacte le comportement du modèle RI parce qu'il peut altèrer le rappel des réponses fournies à la méthode de fusion des réponses (des réponses peuvent être perdues) ;

– la qualité du mécanisme *SendTo* impacte le comportement du modèle RI parce qu'il peut altèrer l'ensemble des pairs sur lesquels les requêtes seront évaluées (des messages de requêtes peuvent être perdus).

Donc l'implémentation du modèle RI doit être paramétrée par la qualité de ces trois types d'objets, pour permettre de mesurer l'impact de la qualité des mécanismes distribués sur son comportement. En termes de composants, cela correspond à ajouter ces trois objets comme paramètres d'entrée et de sortie comme sur le diagramme 5.2.1 :

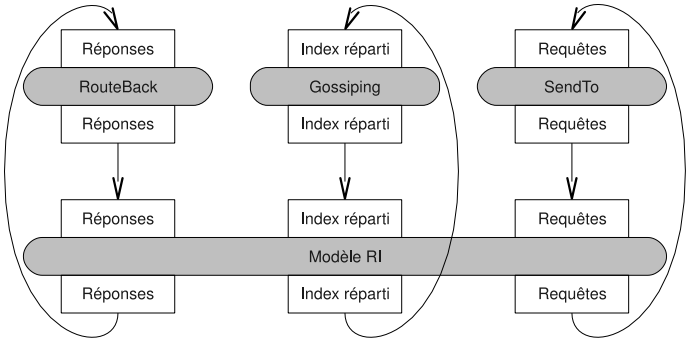


FIG. 5.7 – Diagramme d'évaluation de PlanetP - Première étape : objets à l'interface entre le modèle RI et les mécanismes répartis.

Nous avons donc recensé les objets qui font l'interface avec le modèle RI. Mais il y a d'autres objets en jeu, ce sont les éléments du jeu de données :

Gossiping nécessite en entrée un modèle du réseau physique, à partir duquel il construit le réseau logique. Dans la prochaine section, nous déduirons de la qualité du réseau logique la qualité de l'index des pairs (ex. rappel/précision moyens des tables de routage) ;

RouteBack* et *SendTo sont des mécanismes P2P ; leur évaluation doit donc être paramétrée par un modèle du réseau logique.

Modèle RI prend en entrée des clusters de ressources. On lui donnera donc un modèle des ressources (définitions) et un modèle de distribution des ressources dans les clusters. Il retourne des réponses ; dont la qualité détermine la qualité finale du système.

Cela nous permet de compléter le diagramme comme ci-dessous sur la figure 5.2.1 :

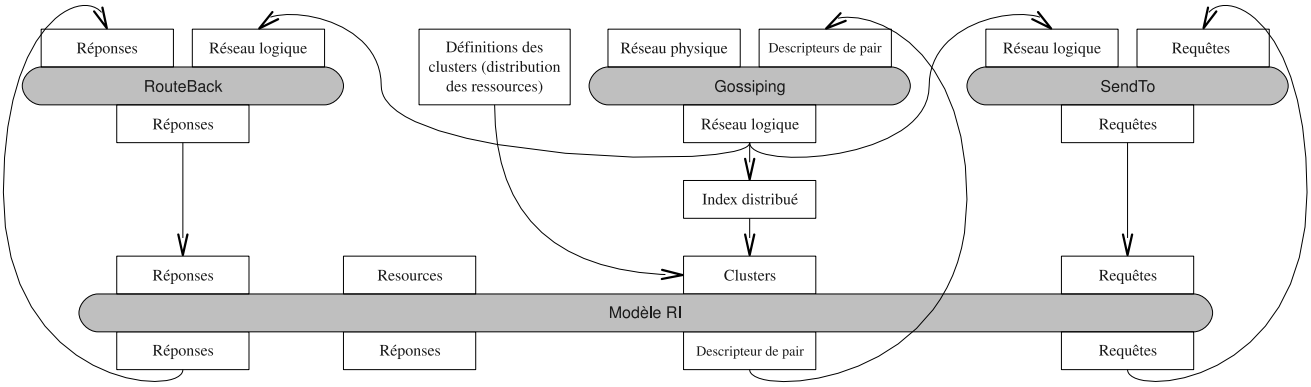


FIG. 5.8 – Diagramme d'évaluation de PlanetP - Deuxième étape : le jeu de données, objets à l'interface avec l'extérieur.

La qualité d'un mécanisme est déterminée par les propriétés de ses arguments (les objets que nous venons de lister). Pour évaluer un système, il nous faut donc lister l'ensemble des propriété que nous voulons prendre en compte. Cet ensemble de propriétés donne en fait un modèle des objets étudiés. Dans la section suivante, nous proposons des modèles des objets que nous venons de lister. On verra que cette réflexion pourra nous amener à modifier les objets pris-en-compte.

5.2.2 Modèles des objets échangés

Le mécanisme de gossiping Commençons par le mécanisme de gossiping. On commence par se poser la question de ce qu'on veut obtenir comme critères de qualité, pour savoir ce dont on a besoin comme paramètres d'entrée. En créant le réseau logique à partir du réseau physique, le gossiping peut modifier deux choses : la qualité des tables de routage (= copies de l'index distribué) et la façon dont ce mécanisme répercute la dynamicité des pairs sur cet index partagé.

La qualité des tables de routage, c'est leur cohérence à un instant donné par rapport à l'état réel du réseau. On peut la mesurer en termes de rappel et précision :

- la précision d'une table de routage, c'est la proportion de filtres de Bloom à jour :

$$P = \frac{nombre\ de\ filtres\ de\ Bloom\ a\ jour,\ de\ pairs\ actuellement\ connectes,\ dans\ la\ table}{nombre\ de\ filtres\ de\ Bloom\ dans\ la\ table\ de\ routage}$$

- le rappel, c'est la proportion du réseau qui est indexée dans la table de routage :

$$R = \frac{nombre\ de\ filtres\ de\ Bloom\ a\ jour,\ de\ pairs\ actuellement\ connectes,\ dans\ la\ table}{nombre\ de\ pairs\ actuellement\ connects}$$

Le mécanisme de gossiping peut aussi modifier la dynamicité perçue. En effet, le délai de propagation des mises à jour peut impliquer des pertes de certaines informations. Une information perdue, c'est un index qui n'est pas mis à jour. Localement, l'index semble être mis-à-jour moins souvent, donc sa dynamicité perçue est inférieure à la dynamicité réelle des pairs.

On peut maintenant se poser la question des paramètres qui influencent ces critères de qualité. La qualité du mécanisme de gossiping dépend des propriétés des données transportées et des propriétés du réseau physique :

Données partagées : Nous caractérisons les données partagées (ici, les filtres de Bloom) par :
- leur taille
- leur dynamicité

On parle ici de la dynamicité des filtres de Bloom, donc du contenu des pairs (ressources). En effet, plus la donnée est volumineuse, plus elle est difficile à transporter. Ensuite, plus la donnée est dynamique - fréquence de modification - plus le maintien de cohérence est coûteux.

Réseau physique : Nous caractérisons le réseau physique par :
- son échelle
- sa dynamicité
- sa topologie

L'échelle, c'est bien entendu le nombre de pairs en jeu. La dynamicité concerne les pairs, donc la façon dont ils joignent et quittent le réseau. La topologie pourra être décrite par différents « sous-modèles » (par exemple Barabasi-Albert (Albert and Barabási, 2002), Watts-Strogatz (Watts and Strogatz, 1998)). Donc le modèle de topologie est en fait l'ensemble des sous-modèles possibles. On verra que les modèles sont facilement extensibles et que l'expérimentation peut être menée de façon incrémentale, sous-modèle par sous-modèle.

RouteBack Passons à l'analyse du mécanisme *RouteBack*, qui rapatrie les réponses sur le pair source. Dans PlanetP, ceci se fait en un seul saut, puisque les requêtes ne sont propagées que sur un seul saut. Donc le vrai « risque » avec *RouteBack*, c'est qu'un pair soit supprimé pendant le calcul d'une réponse. Si on veut être plus rigoureux, on peut analyser les causes possibles d'un défaut de pair en sortie du mécanisme *RouteBack*. Un défaut de pair peut se produire :

1. pendant la résolution locale de la requête
2. pendant le calcul de propagation
3. pendant la transmission de la requête.

Nous pensons que ces deux derniers temps étant très courts par rapport au temps de connexion d'un pair, ces cas de figure sont très exceptionnels et peuvent être négligés

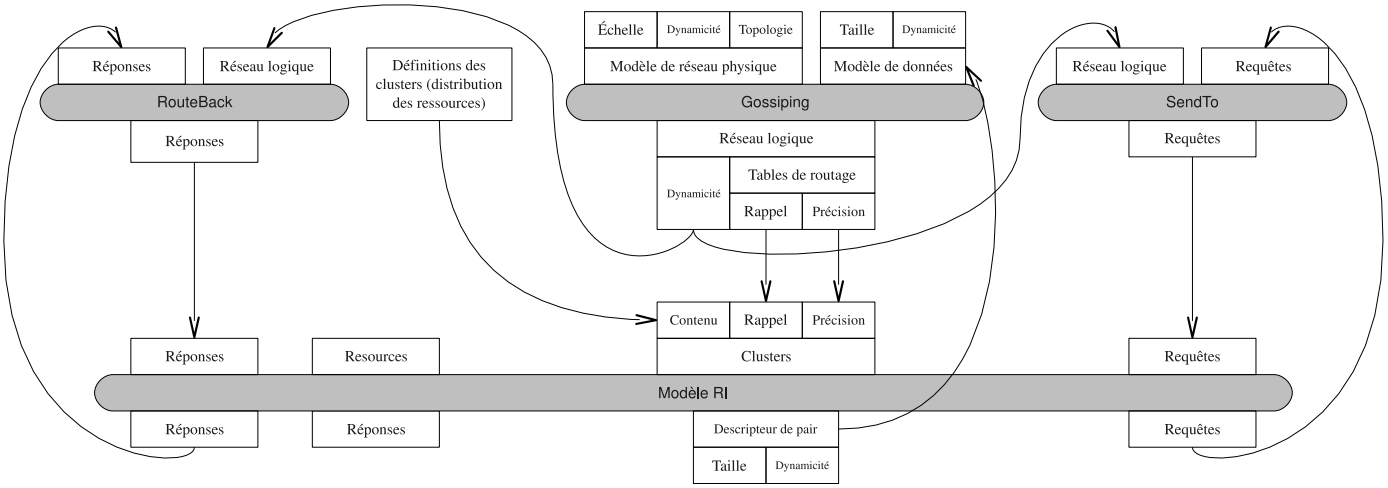


FIG. 5.9 – Diagramme d'évaluation de PlanetP - Troisième étape : modèles des objets échangés.

pour l'évaluation. Donc le véritable biais introduit par le routage arrière des réponses, c'est la possibilité d'une déconnexion du pair pendant le temps de la résolution locale d'une requête.

Le résultat d'une telle déconnexion, c'est une réponse qui ne sera pas envoyée à la méthode *Merge*. Donc on peut la simuler en agissant sur le jeu de données passé en argument du modèle RI : on supprime toutes les ressources attachées au cluster concerné, tout en conservant le même oracle. En faisant cela, le référentiel est calculé en considérant que le pair a des ressources (éventuellement des réponses) ; mais le modèle RI est évalué en considérant qu'il n'en a pas. Si une requête est évaluée sur ce cluster, la réponse retournée sera donc vide, ce qui simule l'impact d'un défaut de pair.

Reste à déterminer le nombre de clusters à « vider ». Pour cela, il faut comparer le modèle de dynamicité du réseau logique - modèle de connexion/déconnexion des pairs - avec le temps de calcul pour une évaluation locale ; pour donner une probabilité de déconnexion d'un pair pendant l'évaluation locale d'une requête. Donc il faut sortir du modèle RI le critère de qualité *temps d'évaluation locale*.

Cependant, on peut aussi bien considérer que les temps d'évaluation locale sont très similaires, quel que soit le modèle utilisé. On a au moins des résultats dans ce sens pour le Modèle vectoriel vs. filtres de Bloom ; mais il faut aussi voir que les temps d'évaluation locale sont très inférieurs aux temps globaux d'évaluation d'une requête par le système. Dans ce cas, tous les systèmes sont confrontés de la même façon au problème des défauts de pairs pendant le temps de la résolution locale. En même temps, le but

de notre évaluation n'est pas tant de calculer précisément les performances - même si elle peut en donner une idée précise - que de comparer les systèmes entre eux, pour comprendre quelle solution est la mieux adaptée à quelle situation. Dans ce cas, si tous les systèmes sont confrontés de manière égale au même problème, on peut éliminer ce composant des diagrammes d'évaluation ; sans changer la façon dont les performances des systèmes se placent les unes par rapport aux autres. On obtient donc le diagramme 5.10.

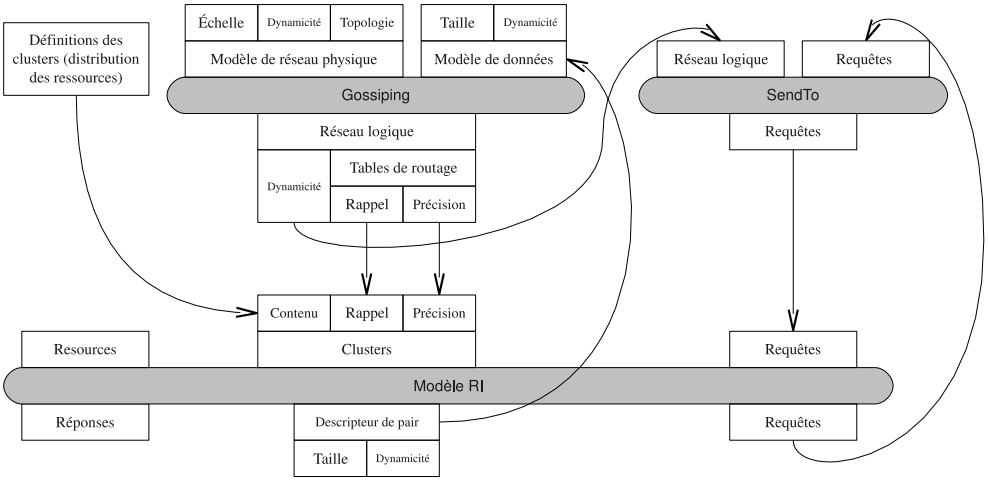


FIG. 5.10 – Diagramme d'évaluationd de PlanetP : simplification du composant *route-Back*.

Il nous semblait tout de même intéressant de montrer cette réflexion sur le mécanisme *RouteBack*, pour illustrer la façon dont nous construisons les modèles de données.

SendTo Voyons enfin l'activité *Sendto()*. Comme pour l'activité précédente, nous proposons de faire l'hypothèse qu'il ne peut pas y avoir de rupture de connexion pendant le temps du calcul de propagation d'une requête, ni pendant le temps de propagation de cette requête. Sous cette condition, toute perte de message au niveau de l'activité *SendTo* est due à un problème en amont : un défaut de pair dans les tables de routage. Or, la qualité des tables de routage est déjà paramétrée, elle est déterminée par la qualité du mécanisme de gossiping.

Donc sous l'hypothèse qu'il ne peut pas y avoir de rupture de connexion pendant le temps du calcul de propagation d'une requête, on peut simplement effacer l'activité *SendTo* du diagramme. Le diagramme d'évaluation devient alors :

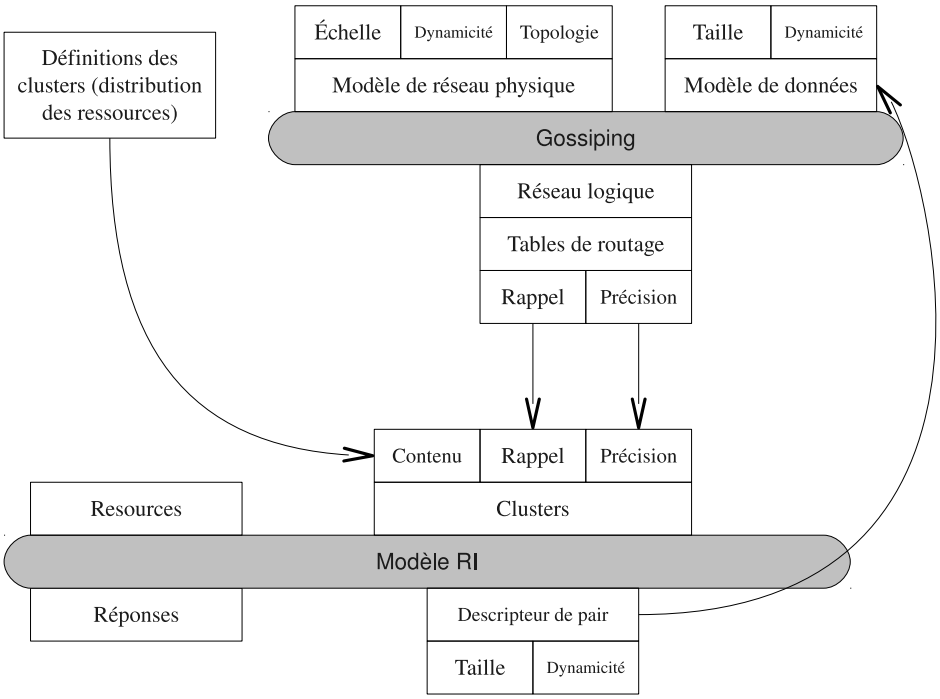


FIG. 5.11 – Diagramme d'évaluation de PlanetP

5.2.3 Conclusion

D'après son diagramme d'évaluation, l'évaluation de PlanetP se décompose en deux étapes indépendantes. D'un coté, on construit une implémentation centralisée « classique » du modèle des filtres de Bloom. De l'autre, on évalue le mécanisme de gossiping, ce qu'il nous semble raisonnable de faire à l'aide d'un simulateur.

5.2.3.1 Implémentation centralisée du modèle des filtres de Bloom

Nous proposons donc de construire une implémentation centralisée du modèle des filtres de Bloom. Cela donne un dispositif classique d'évaluation d'un modèle RI. Par contre, le modèle évalué est clustérisé, c'est-à-dire que les ressources sont regroupées dans des clusters - assimilables aux pairs. Il va falloir indexer ces clusters, et la recherche se fera en deux étapes : rechercher des clusters puis rechercher des ressources dans ces clusters.

Dans le cas idéal, on dispose de jeux de données complets. Un jeu de données complet définit des pairs, auxquels sont attachées des ressources documentaires, et on connaît l'évolution de la distribution des ressources sur les pairs dans le temps. On a par exemple une série de photos du contenu des pairs prises à intervalles de temps réguliers.

Pour intégrer l'effet du mécanisme de gossiping dans le processus de Recherche d'Information, nous devons dégrader artificiellement la table des filtres de Bloom des pairs. Cette dégradation doit être exprimée en termes de rappel et précision, pour permettre de faire ensuite le lien avec les résultats d'évaluation du mécanisme de gossiping. Le résultat qu'on obtient, c'est le comportement qu'on peut espérer obtenir sur un pair du réseau P2P.

Soit un couple (rappel, précision). Le rappel donne la proportion des filtres de Bloom connus en moyenne par un pair du réseau. À partir de notre jeu de données, on peut donc générer un ensemble de tables des filtres de Bloom satisfaisant un certain niveau de rappel, en supprimant des entrées de cette table. Nous pensons qu'on peut simplement supprimer des entrées au hasard. Chaque ensemble de suppressions donne un jeu de données, sur lequel on peut lancer un ensemble de requêtes, et mesurer la qualité du résultat obtenu. En générant un grand nombre de jeux de données à un niveau de rappel donné, on gagne en précision, mais on augmente la complexité et donc la durée de l'expérimentation. Un compromis doit donc être choisi par l'expérimentateur.

La précision décrit le niveau de fidélité du filtre de Bloom au contenu du pair. Les pertes de précision sont dues au délai nécessaire au gossiping pour propager l'information d'une modification d'un filtre de Bloom ; cette modification est due à un changement du contenu du pair. Pour retranscrire ces délais, il faut, pour la distribution des ressources à l'instant t, modifier les filtres de Bloom de certains pairs pour les mettre à la valeur qu'ils avaient à l'instant $t - x$. Il faut que la moyenne des valeurs de x choisies pour

chaque filtre de Bloom soit égale au taux de précision spécifié. Il existe donc beaucoup de possibilités pour créer artificiellement une dégradation d'un même taux de précision.

Il existe aussi des cas où le jeu de données n'est pas complet. Dans ce cas, il va falloir l'enrichir artificiellement. Notez que le but de l'évaluation est d'évaluer le composant sur une plage de valeurs pour chaque paramètre du jeu de données ; dans l'objectif de trouver autour de quelles valeurs on obtient les rapports qualité/coût les plus intéressants. *Un* jeu de données n'est donc pas satisfaisant. On a besoin d'*un ensemble* de jeux de données permettant d'obtenir plusieurs valeurs pour chacun des paramètres de jeu de données.

Premier manque possible, on n'a pas forcément la donnée de l'évolution de la distribution des ressources sur les pairs dans le temps. Dans ce cas, il va falloir générer artificiellement des modifications du contenu des pairs. Il y a deux types de modifications possibles, ajouter ou supprimer une ressource. Cela peut se simuler en :

- enlevant x_1 ressources du calcul du filtre de Bloom. Ces x_1 ressources ne sont donc pas prises en compte dans le calcul du filtre de Bloom, ce qui simule des ajouts de ressources non répercutés sur le filtre de Bloom du pair ;
- enlevant x_2 ressources du pair, qui seront quand-même prises en compte dans le calcul du filtre de Bloom. Cela crée x_2 possiblilités qu'un défaut de ressource se produise lorsque le pair est interrogé ; cela simule des suppressions de ressources ;

Il faut ensuite choisir des couples (x_1, x_2) tels que $x_1 + x_2 = x$. Il nous semble suffisant de choisir les ressources aléatoirement, mais les ensembles de x_1 ressources et x_2 ressources doivent être disjoints.

Il est possible qu'on n'ait pas la donnée de la distribution des ressources sur les pairs. En Recherche d'Information, notamment, nous n'avons pas connaissance d'un jeu de données dans lequel des utilisateurs différenciés publient des ressources sur le système. Comme nous l'avons déjà évoqué, même dans le cas où on dispose d'un tel jeu de données, il est intéressant de challenger le comportement du système dans d'autres conditions, donc sur d'autres types de distributions des données sur les pairs. Dans ce cas, il faut créer une distribution artificielle des ressources sur les pairs.

Un grand nombre de politiques de distribution des ressources sur les pairs ont été proposées dans la littérature, parmi lesquelles on peut citer une distribution en loi de puissance, une distribution en loi de Zipf ou une distribution Small-World.

Pour finir, quelle que soit la façon dont sont générés les jeux de données, l'évaluation du modèle RI de PlanetP doit rendre deux sorties :

- les réponses calculées, qui pourront être comparées aux réponses fournies par l'oracle à l'aide d'une courbe précision/rappel ;
- un modèle des filtres de Bloom utilisés, donnant leur taille et leur dynamicité - il est possible qu'un filtre de Bloom soit recalculé moins souvent que la ressource correspondante n'est modifiée.

5.2.3.2 Un simulateur pour évaluer le mécanisme de gossiping

D'après le diagramme d'évaluation de PlanetP, notre implémentation du mécanisme de gossiping doit prendre en entrée un modèle de réseau physique. Dans la grande majorité des simulateurs dont nous avons eu connaissance, on abstrait la topologie du réseau physique par ses conséquences sur les communications entre les pairs (gigue, pertes de messages...). Par contre, on évalue les performances du système suivant le nombre de pairs dans le système (échelle) et leur dynamicité. On rencontre des modèles de dynamicité des pairs plus ou moins complexes, allant d'une simple probabilité de déconnexion à une loi d'évolution de la connectivité au cours du temps.

L'implémentation du mécanisme de gossiping prend aussi en entrée un modèle des filtres de Bloom tel que donné en sortie du composant précédent : taille et dynamicité (fréquence des mises à jour).

Enfin, nous devons calculer en sortie la qualité des tables de routage, dont nous avons discuté à la section précédente.

5.2.3.3 Réutilisation de résultats

Comme nous l'avons déjà évoqué, à la fois le modèle des filtres de Bloom et le mécanisme de gossiping ont déjà été beaucoup étudiés. La question qui vient naturellement est donc : « peut-on réutiliser des résultats précédents ? ».

Coté modèle RI, l'originalité ici tient en ce qu'on donne en sortie de l'évaluation du modèle RI un modèle des filtres de Bloom : leur taille et leur dynamicité. Concernant leur taille, généralement celle-ci est plutôt un paramètre (système) de l'évaluation du modèle des filtres de Bloom : on évalue la qualité du modèle en fonction de la taille de filtres utilisée. Il suffit donc de transmettre ce paramètre dans les résultats d'évaluation ; cela ne pose a priori pas de difficulté. Concernant leur dynamicité, le problème de la fréquence de mise à jour des filtres de Bloom est orthogonal à celui de leur efficacité. Elle n'est donc généralement pas prise en compte dans l'évaluation du modèle. Mais on peut ré-injecter un modèle de dynamicité des filtres de Bloom a posteriori, avant de transmettre le jeu de données au simulateur du mécanisme de gossiping. Coté évaluation du modèle RI, nous pensons donc que réutiliser des résultats sur le modèle des filtres de Bloom ne pose pas de difficulté majeure.

Concernant le mécanisme de gossiping, réutiliser des résultats pré-existants nous semble difficile. En effet, à notre connaissance, l'évaluation de ce mécanisme prend rarement en compte la taille des données partagées. Réutiliser une évaluation précédente risque donc de donner des résultats pauvres, parce qu'elle n'évaluera en fait qu'une seule valeur du paramètre « taille des filtres de Bloom ». On peut cependant réutiliser des résultats existants et les compléter par une nouvelle étude.

5.3 Analyse de pSearch

Nous allons maintenant instancier notre méthodologie d'analyse des systèmes RI-P2P sur le système pSearch (Tang et al., 2003). Comme PlanetP, pSearch est construit sur la base du modèle vectoriel (VSM) - les auteurs spécifient explicitement l'implémentation LSI de ce modèle. Il diffère cependant en deux points. Tout d'abord, le modèle de répartition de pSearch s'appuie sur un réseau structuré. Ensuite, les ressources sont re-clutérisées, c'est-à-dire qu'elles sont déplacées pour redéfinir les clusters.

Dans la suite, après avoir rappelé les méthodes fournies par le modèle vectoriel, nous étudions la technique P2P sur laquelle s'appuie pSearch : les réseaux structurés.

Puis, comme pour PlanetP, la première étape d'adaptation au modèle de répartition consiste à introduire la notion de cluster. Par contre, nous allons voir qu'ici on ne construit pas les index de pairs en fonction de leur contenu. Les ressources étant re-clustérisées, on fait l'inverse : on choisit d'abord les index de pair, puis on ré-affecte les ressources en fonction de ces index.

Il faut ensuite définir la structure d'indexation qui sera implémentée sur le réseau P2P. Nous verrons qu'on a une double indexation, qui permet de retrouver le point sémantique correspondant à la requête, puis de parcourir son voisinage. Nous décrirons les algorithmes de résolution associés à ces deux indexations.

Avant-dernière étape, nous verrons que comme pour PlanetP le modèle vectoriel a été adapté pour faciliter sa répartition. Dans PlanetP, il s'agissait de réduire la taille de l'index des pairs. Ici, les auteurs optimisent la structure d'indexation, en appliquant une technique de réduction de la dimensionnalité des index de requêtes, de ressources et de pairs : les *rotated spaces*.

Enfin, nous verrons quels mécanismes P2P sont mis-en-place pour construire le système pSearch tel que nous le connaissons.

5.3.1 Un réseau structuré

pSearch utilise un réseau structuré. Le principe du réseau structuré a été introduit par les DHT (Distributed Hash Tables, tables de hachage distribuées). Le système CAN (Ratnasamy et al., 2001), dont pSearch revendique la filiation, en est un représentant.

Le principe est double. Premièrement, il s'agit d'implémenter une structure de données sur un réseau. Les objets de la structure (ou des groupes d'objets) sont assimilés aux pairs, les liens entre ces objets (e.g pointeurs) sont implémentés par des connexions réseau. Secondement, la structure de données est le support d'un algorithme (réparti) travaillant sur ces données, souvent un algorithme de recherche. L'apport d'une implémentation répartie de cette structure est double : permettre de partager le coût de

maintien de la structure de données entre les pairs, et partager l'accès à cette structure (possibilité d'initier l'algorithme) entre tous les pairs.

Dans pSearch - c'est probablement une règle générale - on distingue un algorithme de parcours de la structure de données et plusieurs fonctionnalités qui utilisent cet algorithme ; par exemple pour monitorer la structure, indexer de nouvelles données ou rechercher une donnée.

Note : Pour que chaque pair puisse avoir accès aux données dans les mêmes conditions, une bonne solution est de faire en sorte que le graphe de la structure de données soit sommet-transitif - il est probable que ce soit une condition nécessaire. Intuitivement, un graphe sommet-transitif est tel que si on lui applique n'importe quelle permutation des sommets, le graphe obtenu est toujours semblable au graphe initial (homomorphisme). Cela permet que n'importe quel pair puisse initialiser l'algorithme de parcours de la structure de données. En effet, quel que soit le pair qui initie l'algorithme de parcours de cette structure, le graphe qui se présente devant ce pair est le même, et donc le comportement de l'algorithme est semblable. Cette idée est inspirée du travail de (Ratajczak and Hellerstein, 2003), voir p.45.

Exemples : Dans une table de hachage, la structure de données est l'espace des clefs. Les objets sont des points ou des portions de cet espace, les liens entre ces objets sont de deux types : soit ils correspondent à la relation de voisinage dans l'espace des clefs, ils construisent l'implémentation répartie de la structure ; soit ils sont créés pour permettre un parcours efficace de la structure. La structure de données s'apparente donc à une structure d'index construite au-dessus des clefs, comme ce pourrait être fait dans une base de données. Deux différences notoires cependant : tous les nœuds de la structure d'index sont étiquetés par une donnée, et il n'y a pas d'entrée unique (e.g racine d'un arbre d'indexation) : tout nœud peut être choisi comme point d'entrée d'une recherche.

Dans *Chord* (Stoica et al., 2001), les clefs sont triées par ordre croissant, la plus grande clef étant voisine de la plus petite. Parallèlement, un ensemble de liens « longue distance » sont mis-en-place. En utilisant ces liens, les recherches de clef s'apparentent à des recherches dichotomiques (donc en temps logarithmique). N'importe quelle clef peut être choisie comme point d'entrée de la recherche. Dans *CAN*, les clefs appartiennent à un espace vectoriel à n dimensions ; le résultat du tri donne donc une structure où chaque donnée a $2n$ voisins - 1 voisins supérieur et 1 voisin inférieur dans chaque dimension.

Dans pSearch, les données sont les index de pair. La structure de données des index s'appuie sur la relation de proximité sémantique entre les index, définie par la fonction cosinus. Parcourir la structure, c'est alors parcourir l'espace sémantique. Les index étant des vecteurs de dimension n, en passant en coordonnées sphériques, on obtient une structure isomorphe à celle de CAN - nous précisons ce point dans la suite.

5.3.2 Le modèle vectoriel

Comme PlanetP (section 5.1.2), pSearch est construit sur la base du modèle vectoriel. À la nuance près cependant que dans LSI l'IDF est utilisé au moment de la construction des index, plutôt que dans la fonction d'évaluation. Cela ne change rien au résultat ; mais permet d'accélérer les calcul dans le cas où les résolutions de requête sont plus fréquentes que les mises-à-jour d'index ; ce qui est en général le cas. Nous conservons donc la même spécification du modèle vectoriel, mais le diagramme de d'activités est légèrement adapté :

- $vsm.IndexResource : \underline{Resource\ r} * IDF \rightarrow Resource\ r$: indexation des ressources
- $vsm.IndexQuery : \underline{Query\ q} * IDF \rightarrow Query\ q$: indexation des requêtes
- $vsm.Sim : \underline{Query\ q} * Resource\ r \rightarrow float$: mesure de similarité
- $vsm.IDF : \underline{Resource\ r} \rightarrow IDF$: calcul de l'IDF, basé sur une méthode de mise-à-jour suite à une modification du corpus

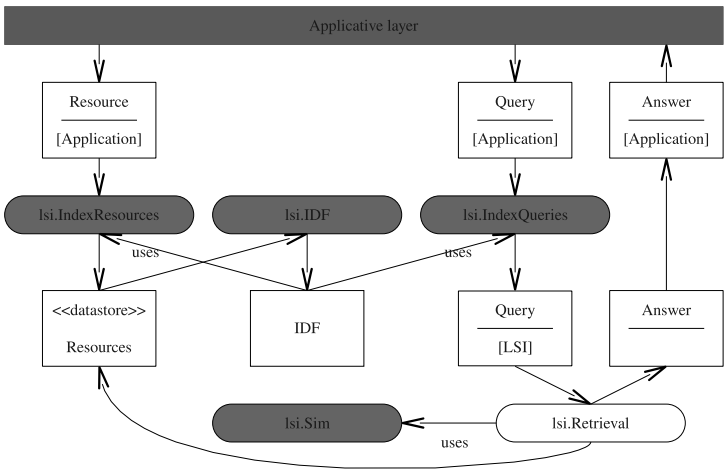


FIG. 5.12 – Notre point de départ : diagramme d'activités du modèle vectoriel.

5.3.3 Introduire la notion de cluster : re-clustérisation des ressources

Voyons maintenant comment est introduite la notion de cluster. Dans pSearch, les ressources sont re-clutérisées sur les pairs, c'est-à-dire qu'elles sont déplacées pour redéfinir les clusters. Donc ici, le principe n'est pas de décrire un domaine d'expertise - comme dans PlanetP - mais de le construire. C'est la re-clustérisation des ressources qui définit les domaines d'expertise des pairs. En conséquence, un domaine d'expertise de pair n'est plus un vecteur comme pour PlanetP mais une portion de l'espace sémantique (espace vectoriel des index de ressources créés par LSI). L'index d'un pair, en définissant

une portion de l'espace sémantique, définit l'ensemble des ressources gérées par lui. Les index sont choisis de façon à minimiser l'effort de re-clustérisation.

La re-clustérisation des ressources consiste à partitionner l'espace sémantique (espace des vecteurs LSI), pour attribuer à chaque cluster une portion d'espace - et les ressources qu'elle contient. LSI utilise comme mesure de distance le cosinus sur les vecteurs normalisés. Pour simplifier les calculs, on peut passer les vecteurs normalisés en coordonnées sphériques ; cela permet d'utiliser la mesure de distance canonique sur $n-1$ coordonnées, plutôt que de travailler avec des cosinus. Dit autrement, une fois les vecteurs de taille n normalisés, ils se retrouvent tous sur une hypersphère de rayon 1. Cette hypersphère appartient à un espace vectoriel de dimension $n-1$; pour la partitionner, on utilise la distance canonique entre les index passés en coordonnées sphériques.

La re-clustérisation des ressources s'appuie sur deux méthodes : *Join* et *Leave*. Cette structure en deux méthodes permet de mettre à jour les clusters pendant l'exécution, au fur et à mesure des ajouts et suppressions de clusters (entrées et sorties d'utilisateurs dans le système). La méthode *Join* repose sur l'attribution de coordonnées d'entrée à chaque cluster : les *coordinates*. Les *coordinates* d'un cluster sont l'index d'une de ses ressources choisie au hasard. Le premier cluster inséré se voit attribuer l'ensemble de l'espace ; ensuite, chaque nouveau cluster prend en charge la moitié de l'espace de l'« ancien » cluster auquel appartiennent ses coordonnées.

La méthode *leave* permet la restitution d'une portion d'espace lors de la suppression d'un pair. On peut alors se trouver dans deux cas de figure. Première possibilité : la portion d'espace peut être fusionnée avec une portion d'espace voisine (les deux portions d'espace sont adjacentes dans une dimension et se recouvrent dans toutes les autres). Dans ce cas, on attribue au voisin l'union des deux portions d'espace. Sinon, il faut trouver deux portions d'espace qui peuvent être fusionnées - il existe au moins pour cela les portions d'espace résultant du dernier *Join*. En les fusionnant, on libère un cluster qui peut prendre-en-charge l'espace orphelin.

Par rapport au modèle vectoriel, on rajoute donc une méthode qui calcule et met-à-jour les coordonnées des clusters : $pSearch.Compute\ Coordinates$, suivie des méthodes *Join* et *leave*, qui calculent les espaces associés aux clusters. Elles sont accompagnées d'une méthode $DistributeResources$, qui gère la re-distribution des ressources dans ces clusters :

- $vsm.IndexResource : \underline{Resource\ r} * IDF \rightarrow Resource\ r$
- $vsm.IndexQuery : \underline{Query\ q} * IDF \rightarrow Query\ q$
- $vsm.IDF : \underline{Resource\ r} \rightarrow IDF$
- $pSearch.ComputeCoordinates : Cl * \{R | R \in Cl\} \rightarrow Cl.coordinates$
- $join : Cl.coordinates * \{Cl.space\} \rightarrow \{Cl.space\}$
- $leave : Cl * \{Cl.space\} \rightarrow \{Cl.space\}$
- $DistributeResources : \{\{R\}\} * \{Cl.space\} \rightarrow \{\{R\}\}$

– $vsm.Sim : (\underline{Query\ q\ OR\ Resource\ r\ OR\ Peer\ p})^2 \rightarrow float$

La figure 5.3.3 montre le diagramme d'activités correspondant. Il introduit notamment des méthodes de résolution inter et intra-cluster. *pSearch.IntraClusterRetrieval* utilise *Sim*() pour comparer les requêtes à toutes les ressources d'un même pair ; le comportement de *pSearch.InterClustersRetrieval* n'est pas spécifié à ce stade.

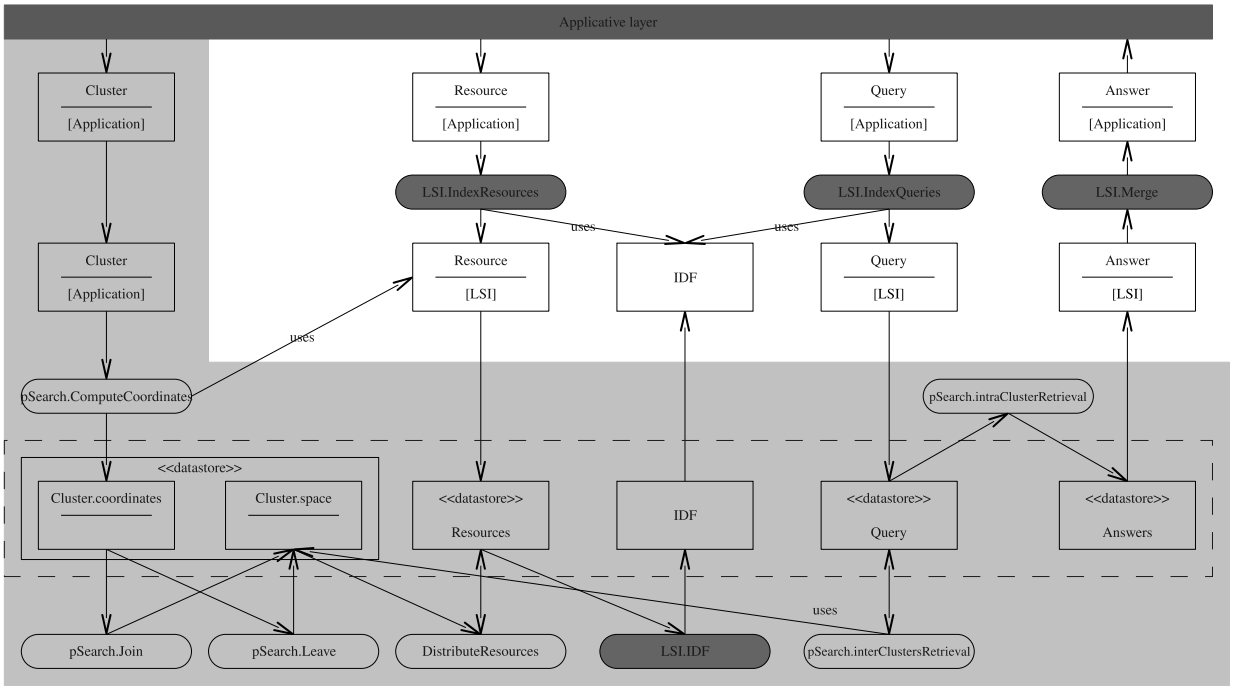


FIG. 5.13 – pSearch, première étape : introduire la notion de cluster

5.3.4 Structure d'indexation et algorithmes de parcours

Structure d'indexation Le mécanisme de re-clustérisation des ressources crée donc un premier type d'index des pairs en leur attribuant une portion de l'espace sémantique : *peer.space*. On s'appuie sur la relation de voisinage qui existe entre les portions d'espace pour organiser les clusters en structure : deux clusters administrant des espaces voisins sont voisins dans la structure d'indexation. Cette structure permet de rechercher n'importe quel cluster à partir de n'importe quel autre cluster, et donc d'atteindre n'importe quel point de l'espace sémantique à partir de n'importe quel autre point. On utilise pour cela un algorithme de parcours. Il consiste à parcourir la structure de proche-en-proche à partir d'un cluster de départ. En choisissant à chaque étape

le voisin le plus proche du point à atteindre, l'algorithme converge vers le cluster qui administre ce point. La notion d'appartenance d'un point à un espace peut être déduite de la mesure de similarité entre index (fonction cosinus).

Notez que cette structure ne peut être construite que parce que l'espace vectoriel des index peut être ordonné a priori. C'est-à-dire qu'on peut classer les index de ressources dès l'initialisation du système - c'est la structure d'indexation - et utiliser ce même classement pour toutes les résolutions de requêtes. Cela n'a rien d'évident en général. En effet, généralement en RI on définit un algorithme permettant de classer les ressources pour chaque requête. Rien n'assure que les classements pour les différentes requêtes soient compatibles et qu'on puisse en tirer une classification a priori. Si on n'a pas de moyen de classifier a priori, on ne peut pas construire de structure de résolution à la pSearch.

Comme dans la plupart des systèmes RI-P2P s'appuyant sur un réseau structuré, la résolution d'une requête sous pSearch s'effectue en deux étapes. La première étape consiste à trouver le point sémantique correspondant à la requête. La seconde consiste à parcourir une boule au voisinage de cette requête, pour trouver les réponses qui s'en approchent. La première étape utilise directement l'algorithme de parcours que nous venons de voir. La seconde s'appuie sur un second type d'index : les *Sample Sets*.

Second type d'index : les *Sample Sets* À ce stade, nous considérons une relation *SampleSets()*. Nous verrons dans une prochaine section comment cette relation est implémentée en P2P dans pSearch, à l'aide des objets *Sample Sets*. La relation *SampleSets()* est un second type d'index, dont le but est de contourner le problème de *curse of dimensionality*. Ce problème vient du très grand nombre de dimensions de l'espace vectoriel des index de pair. Le grand nombre de dimensions a pour effet de tasser les distances. Le voisinage d'une requête devient très grand, et deux réponses très proches de la requête peuvent fournir des niveaux de satisfaction très différents. Pour contourner cette difficulté, une solution est de repérer des portions d'espace denses en réponses. On oriente ensuite la recherche en parcourant ces groupes de réponses et en ignorant le reste de l'espace.

Concrètement, les *Sample Sets* forment un *étiquetage orienté de la relation de voisinage*. Cet étiquetage est construit comme suit. On construit tout d'abord un centroide pour chaque cluster. Le centroide est le barycentre des index des ressources et requêtes émises par le cluster. Le *Sample Set* de Cl_1 vers Cl_2 est un ensemble de ressources de Cl_2, composé des k_c ressources les plus proches du centroide de Cl_1 (au sens de $sim()$) et de k_r ressources choisies aléatoirement.

Nous proposons d'abstraire ce processus d'indexation par deux méthodes :
- $pSearch.ComputeCentroid$:
  $Cl \ * \ \{R|R \in Cl\} \ * \ \{Q|Q \in Cl\} \rightarrow Cl.centroid$
- $pSearch.SampleSets$:
  $\{(\ Cl_1.centroid, \{R|R \in Cl_2\}\ ) \mid (Cl_1 \ \mathcal{R} \ Cl_2)\} \rightarrow sampleSet(Cl * Cl \rightarrow \{R\})$, où $\mathcal{R}$ est la relation de voisinage

Seconde phase de recherche Comme nous l'avons dit, *SampleSets()* sert de support à la deuxième phase de l'algorithme de recherche. Ce parcours repose sur une mesure de similarité : $sim2(Q, Cl_Q, Cl) = max_{\ V \in sampleSet(Cl_Q, Cl)}\ cos(V, Q)$, où Cl_Q est le cluster courant de Q et $Cl_Q \ \mathcal{R} \ Cl$ ($\mathcal{R}$ est la relation de voisinage). Les clusters sont sélectionnés dans l'ordre par groupe de b, jusqu'à ce que les T derniers clusters sélectionnés ne rapportent pas de meilleure réponse. b et T sont calculés suivant le système suivant :

$$b = min(d, T/2)$$

$$T = max(5, F - 5) * 0.8^w$$

$$w = min_{Cl \in N} distance(Cl_0, Cl)$$

où N est la file des clusters sélectionnés, et
Cl_0 le cluster obtenu par la première phase de recherche

Nous choisissons de définir la recherche par deux méthodes de similarité et un algorithme de recherche :
- $SimCl.1 : Q * Cl.space \rightarrow float \leftarrow Sim(Q, Cl.space)$
- $SimCl.2 : Q * Cl_Q * Cl(Cl_Q \mathcal{R} Cl) * (sampleSet : Cl * Cl \rightarrow \{R\}) \rightarrow float$
- $pSearch.interClusterRetrieval.2 : Q * Cl_0 * \{Cl\} \rightarrow \{Cl\}$
  (l'argument $\{Cl\}$ désigne l'ensemble des clusters du corpus)

Bilan : diagramme d'activités Nous obtenons le diagramme d'activités ci-dessous :

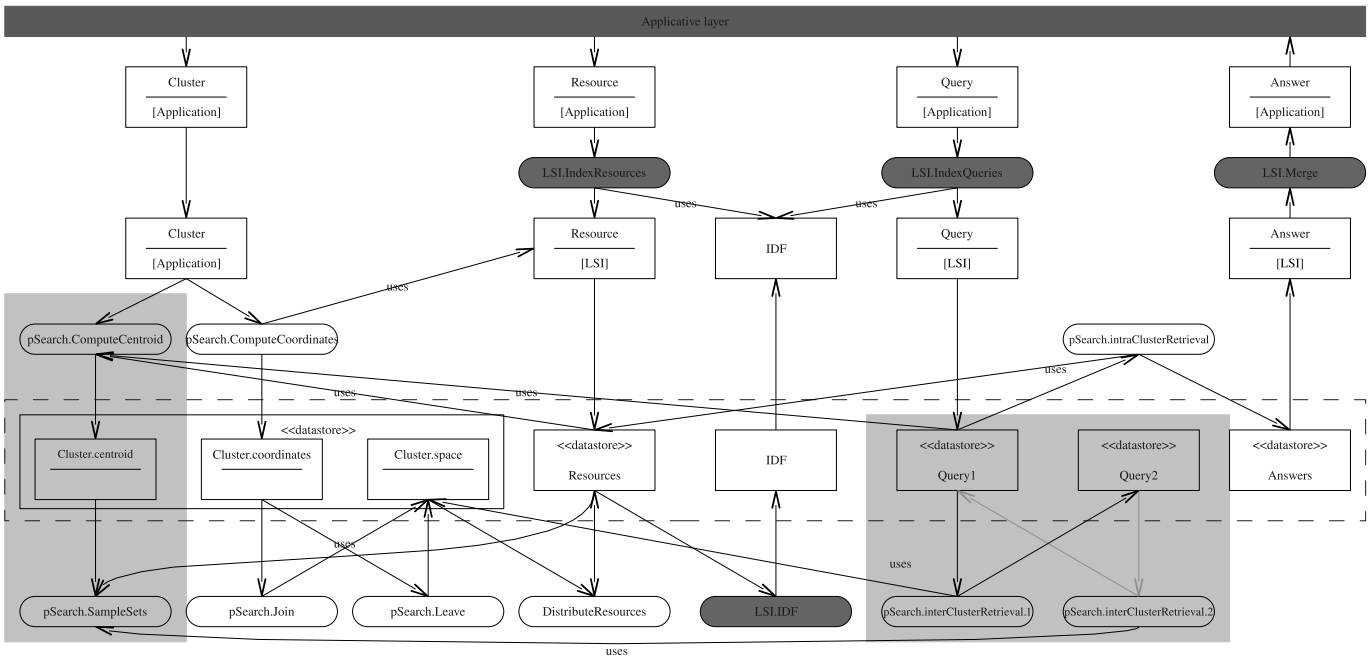


FIG. 5.14 – pSearch, deuxième étape : spécification de la structure d'indexation et de l'algorithme de recherche

5.3.5 Adaptations à l'environnement P2P : les *rotated spaces*

pSearch est un système structuré. C'est-à-dire que la structure des clusters est implémentée sur le réseau, et que la résolution se fait par parcours de cette structure, donc par parcours du réseau. Les auteurs proposent un mécanisme permettant de réduire la taille des index pour qu'ils puissent être pris-en-charge de manière efficace sur le réseau.

Réduire la taille des index Voyons d'abord comment les index de ressources sont réduits. Les auteurs proposent de découper les index de ressource en p sous-index de taille m. Le résultat de la fonction d'indexation devient donc un ensemble d'index, qui sont p projections sur p ensembles de m dimensions. Les ensembles de dimensions sont appelés *rotated spaces*. Ils sont choisis pour maximiser leur pouvoir discriminant et sont codés « en dur » dans le système. En général, les premières dimensions correspondent aux termes les plus discriminants. m est déterminé par l'équation $m = 2.3\ ln(nombre\ de\ pairs)$ - 2.3 déterminé de manière empirique, non détaillé par les auteurs. Certaines dimensions

peuvent être ignorées, donc $p * m$ peut être plus petit que la taille des vecteurs créés par LSI. Chaque ressource est donc indexée p fois, suivant chacun de ces sous-index.

Il en va de même des requêtes, qui sont indexées p fois suivant les mêmes p projections sur m dimensions du vecteur obtenu via LSI. Nous modélisons cette manipulation des index par une méthode *Cut* qui découpe un vecteur en p sous-vecteurs :

– $pSearch.Cut : R \rightarrow \{R\}$

Impact sur l'algorithme de recherche Ces modifications impactent l'algorithme de recherche en deux points. $SimCl.1$ va être utilisé en parallèle sur chaque sous-requête. On obtient donc un ensemble de clusters, un pour chaque sous-requête. C'est donc cette fois un ensemble de clusters de départ qui est passé à $pSearch.interClustersRetrieval.2$. Tout cluster peut donc posséder jusqu'à p mesures de similarité à la requête, calculées respectivement par rapport aux p sous-vecteurs de la requête. La mesure la plus proche est utilisée. On a donc une modification de la méthode $pSearch.interClustersRetrieval.2$, qui récupère en entrée un *ensemble* de requêtes et un *ensemble* de clusters de départ :

– $pSearch.interClustersRetrieval.2 : \{Q\} * \{Cl_0\} * \{Cl\} \rightarrow \{Cl\}$

Diagramme d'activités Nous obtenons le diagramme d'activités ci-dessous :

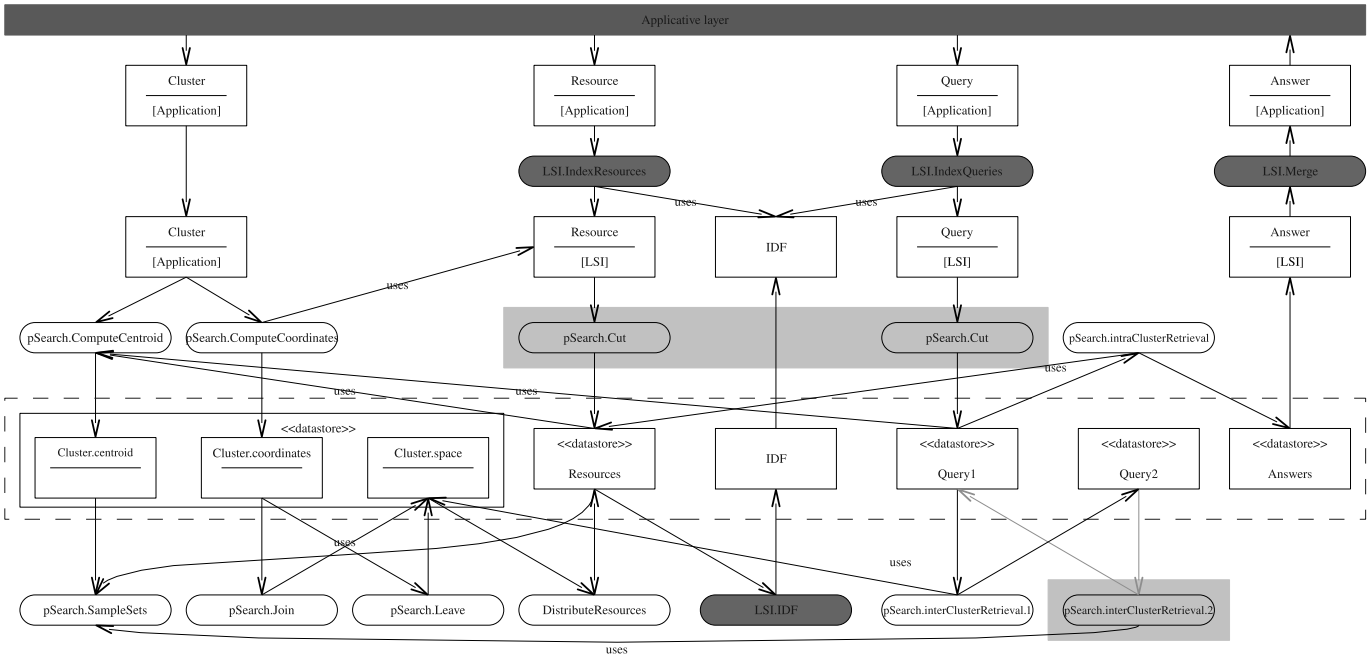


FIG. 5.15 – pSearch, troisième étape : adaptations à l'environnement P2P, les rotating spaces

5.3.6 Modèle de répartition de pSearch

En se référant au diagramme d'activités (figure 5.15), on peut lister les objets et activités à répartir. Les objets à répartir sont les objets partagés entre les clusters, ce sont tous les objets inclus dans le cadre en pointillés. Pour beaucoup de ces objets, une ou plusieurs activités leur sont déjà attachées. Dans ce cas, au moment du passage en P2P, cette activité devient une activité P2P. Il faut donc définir comment elle est implémentée de façon distribuée. Remarquez que pour PlanetP, on avait surtout des objets à répartir, alors qu'ici, on a surtout des activités à répartir. Cela traduit une plus grande complexité du modèle de répartition de pSearch par rapport à celui de PlanetP. pSearch met donc en œuvre plus de mécanismes P2P que PlanetP ; mais on ne peut pas en conclure à ce stade que pSearch sera pour autant plus gourmand en ressources réseau.

Les objets à répartir :
- $pSearch.SampleSets()$
- $pSearch.IDF$

Les activités à répartir :
- $pSearch.Join$
- $pSearch.Leave$
- $DistributeResources$
- $pSearch.interClusterRetrieval.1$
- $pSearch.interClusterRetrieval.2$

L'IDF Les auteurs proposent d'utiliser une technique de *Combining Tree* pour gérer les statistiques globales telles l'IDF : *a combining tree is used to sample documents, merge statistics and disseminate the combined statistics.*

pSearch.SampleSets() L'idée de base du modèle de répartition de pSearch est d'implémenter la structure des clusters sur le réseau. En maintenant sur chaque pair la connaissance des espaces maintenus par ses voisins, on obtient une implémentation répartie de la structure des clusters : la relation de voisinage « sémantique » introduite par le modèle RI clustérisé est implémentée par les connexions entre les pairs. À cette structure est associé un algorithme de parcours basé sur une mesure de similarité déduite du modèle RI, qui servira de base pour répartir plusieurs méthodes :

- $crawlOverlay : V * \{Cl.space\} \rightarrow Cl$
  $(\leftarrow use - PropagateNearer\ (Similarity : V * Cl.space \rightarrow float) :$
  $V * \{Cl.space\} \rightarrow Cl)$

Indexation des ressources et requêtes Traitons tout d'abord les méthodes d'indexation des ressources et requêtes. Il y a disjonction de localité entre la ressource (resp. requête) indexée et le corpus passé en argument. De plus, ce même corpus est un objet réparti. Pour résoudre cette difficulté, les auteur proposent de calculer les données statistiques calculées sur le corpus (tel l'IDF) en utilisant un « combining tree » :

- $indexResources : R * CombiningTree(\ IDF(\{R\})\ ) \rightarrow \{R\}$
- $indexQueries : Q * CombiningTree(\ IDF(\{R\})\ ) \rightarrow \{Q\}$

Indexation des pairs Concernant l'indexation des pairs, les méthodes *indexCluster.coordinates* et *indexCluster.centroid* prennent en argument des objets tous localisés sur le même pair. Les *coordinates* et *centroid* d'un pair seront donc calculés localement à ce pair.

Pour la méthode *join*, la difficulté réside dans la recherche du pair maintenant les coordonnées du pair à insérer. Cela peut se faire par parcours de la structure des clusters, définie par une mesure de proximité entre clusters. On utilise ici la même mesure que pour la propagation des requêtes ; c'est donc en fait le même algorithme qui est exécuté :

- $simClCl : Cl_1.coordinates * Cl_2.space \rightarrow float$

Une fois le pair cible trouvé, il peut exécuter l'algorithme de partage de l'espace. Il calcule aussi son nouveau voisinage et celui du nouveau pair. En maintenant sur chaque pair la connaissance des espaces maintenus par ses voisins, on obtient une implémentation répartie de la structure des clusters : la relation de voisinage « sémantique » est implémentée par les connexions entre pairs. Le résultat (espace + voisinage) est retourné par routage direct au nouveau pair.

La méthode *leave* consiste à céder l'administration de sa portion d'espace à un voisin. Le voisin concerné est choisi localement, le calcul de ses nouveaux espace et voisinage est également local. La partie répartie du processus consiste à envoyer son nouvel espace au voisin choisi, et à notifier tous ses voisins de leur nouveau voisinage.

Nous avons vu que la structure devait être maintenue fortement. Cela impose d'implémenter en parallèle deux méthodes réparties *takeover* et *maintenance* pour gérer la dynamicité du système. La première méthode détecte les défaillances et ré-attribue les portions d'espace orphelines. Elle repose sur un mécanisme d'élection du pair maintenant le plus petit espace, parmi les voisins du pair défaillant. La seconde permet de ré-équilibrer les portions d'espace entre les pairs, en permettant à un pair qui gère plusieurs portions d'espace disjointes (à cause du mécanisme *takeover*) de les céder aux voisins à qui elles reviennent. Cette méthode est implémentée par un mécanisme réparti de recherche des *siblings* - portion d'espace complémentaire à la portion qui doit être cédée.

Enfin, *computeSampleSets* est répartie ; elle repose simplement sur des échanges entre voisins.

- $indexCluster.coordinates : Cl * \{R | R \in Cl\} \rightarrow Cl.coordinates$
- $join : Cl_{nv} * [\ (PropagateNearer(DistanceFunction))(Cl_{nv}.coordinates) * \{Cl.space\} \rightarrow Cl_{anc}.space\ ] \rightarrow (RouteTo(Cl_{nv}))(Cl_{nv}.space * Cl_{nv}.Neighbordhood) * (RouteTo(Cl_{anc}))(Cl_{anc}.space * Cl_{anc}.Neighbordhood)$
- $leave : Cl.space * \{N.space \mid Cl\ \mathcal{R}\ N\} \rightarrow SendToNeighbor(\{N.space \mid Cl\ \mathcal{R}\ N\})$
- $takeover : (NeighborFailureDetectionSystem : \rightarrow Cl\ ) * (Election\ (Cl.space.Size()))\ (\{N.space \mid Cl\ \mathcal{R}\ N\}) \rightarrow N.space$
- $maintenance : distributedRecursion(Cl.space * \{N.space \mid Cl\ \mathcal{R}\ N\} \rightarrow Cl.space * (SendTo(N))\ (N.space))$
- $indexCluster.centroid : Cl * \{R | R \in Cl\} * \{Q | Q \in Cl\} \rightarrow Cl.centroid$
- $computeSampleSet\ :\ (SendTo(N\ \mid\ Cl\ \mathcal{R}\ N))\ (Cl.centroid), \{R\ \mid\ R \in N\} \rightarrow SendBack(\ sampleSet(Cl * N \rightarrow \{R\})\ )$

Algorithme de recherche Enfin, le processus de résolution des requêtes :

- $simCl.1 : (PropagateNearer(DistanceFunction))(Q) * Cl.space \rightarrow float$
- $simCl.2 : Q * Cl_Q * (N \mid Cl_Q \mathcal{R} N) * sampleSet(Cl_Q, N) \rightarrow float$
- $retrievalProcess : (PropagateNearer(simCl.2))\ (Q) * \{Cl_0\} * \{Cl\} \rightarrow \{Cl\}$

Bilan : diagramme d'activités On obtient donc au final pour pSearch le diagramme d'activités présenté sur la figure 5.16.

À titre indicatif, la figure 5.17 présente un diagramme d'activités précis du package *structuredNetwork*, en charge de la gestion du réseau logique dans pSearch. Notez que les concepteurs de pSearch considèrent qu'il *utilise* le système P2P structuré CAN en couche basse. Pourtant, notre analyse montre que l'intersection entre pSearch et CAN est le package *structured Network*, qui ne constitue pas le système CAN en entier, mais seulement une partie. Nous pensons qu'il faut considérer que CAN et pSearch utilisent tous deux le même système de gestion d'un réseau structuré (représenté par le package *structuredNetwork*), pour implémenter deux applications différentes, une table de hachage pour le premier, un système RI pour l'autre.

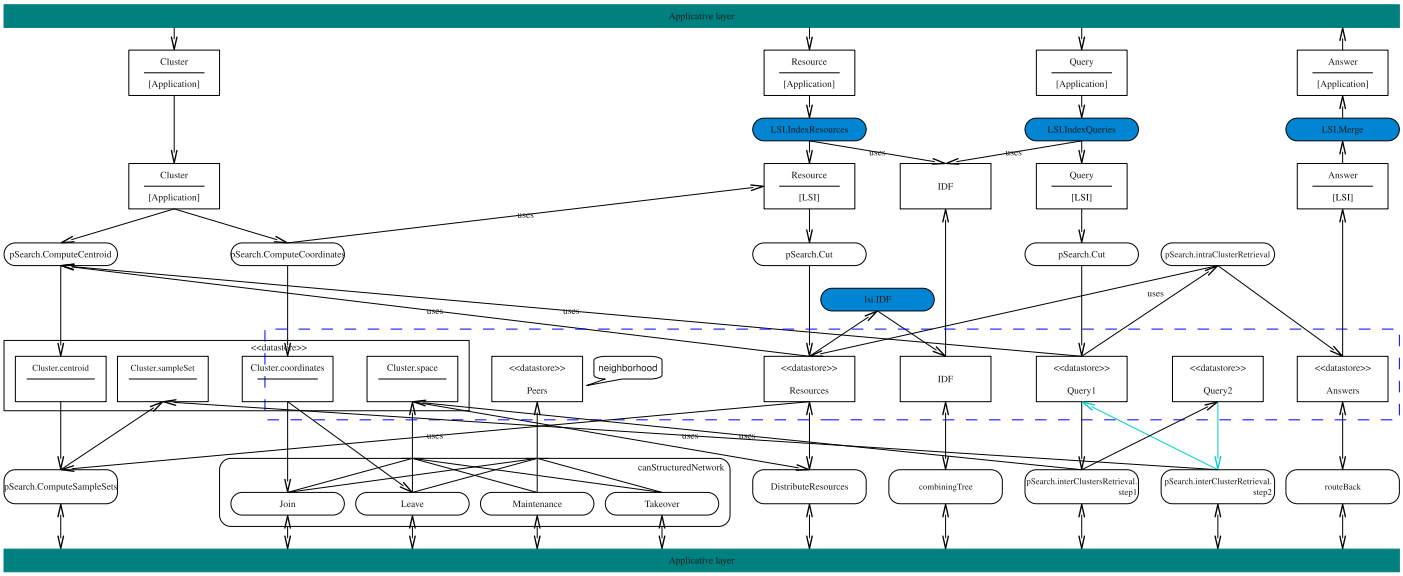


FIG. 5.16 – Diagramme d'activités de pSearch

5.4 Plan d'évaluation de pSearch

D'après le diagramme d'évaluation de pSearch (figure 5.16), on peut lister l'ensemble des mécanismes P2P mis-en-jeu par ce système :

- *pSearch.ComputeSampleSets* met-en-place et maintient les *sample sets*
- *canStructuredNetwork* est l'ensemble de mécanismes en charge du maintien de la topologie du système CAN
- *DistributeResources* reclustérise les ressources sur les pairs
- *combiningTree* est le mécanisme permettant d'approximer l'IDF
- *pSearch.ClustersRetrievalStep1* première phase de recherche, qui s'appuie sur les index VSM
- *pSearch.ClustersRetrievalStep2* seconde phase de recherche, qui s'appuie sur les *sample sets*
- *routeBack* route les réponses

pSearch.ClustersRetrievalStep1 et *pSearch.ClustersRetrievalStep2* peuvent être implémentés de façon centralisée, sur une structure de données où des objets sont associés aux pairs et des pointeurs remplacent les liens de communication. Ainsi, on peut évaluer la validité du modèle RI sous-jacent à pSearch. Par contre, cette implémentation centralisée doit être paramétrée, de façon à prendre en compte les biais introduits par la nature répartie du système. Voyons comment mécanisme par mécanisme :

- Si *pSearch.ComputeSampleSets* n'est pas optimale, elle induit une mauvaise précision des *sample sets* ; c'est-à-dire qu'ils vont contenir des index de ressources/re-

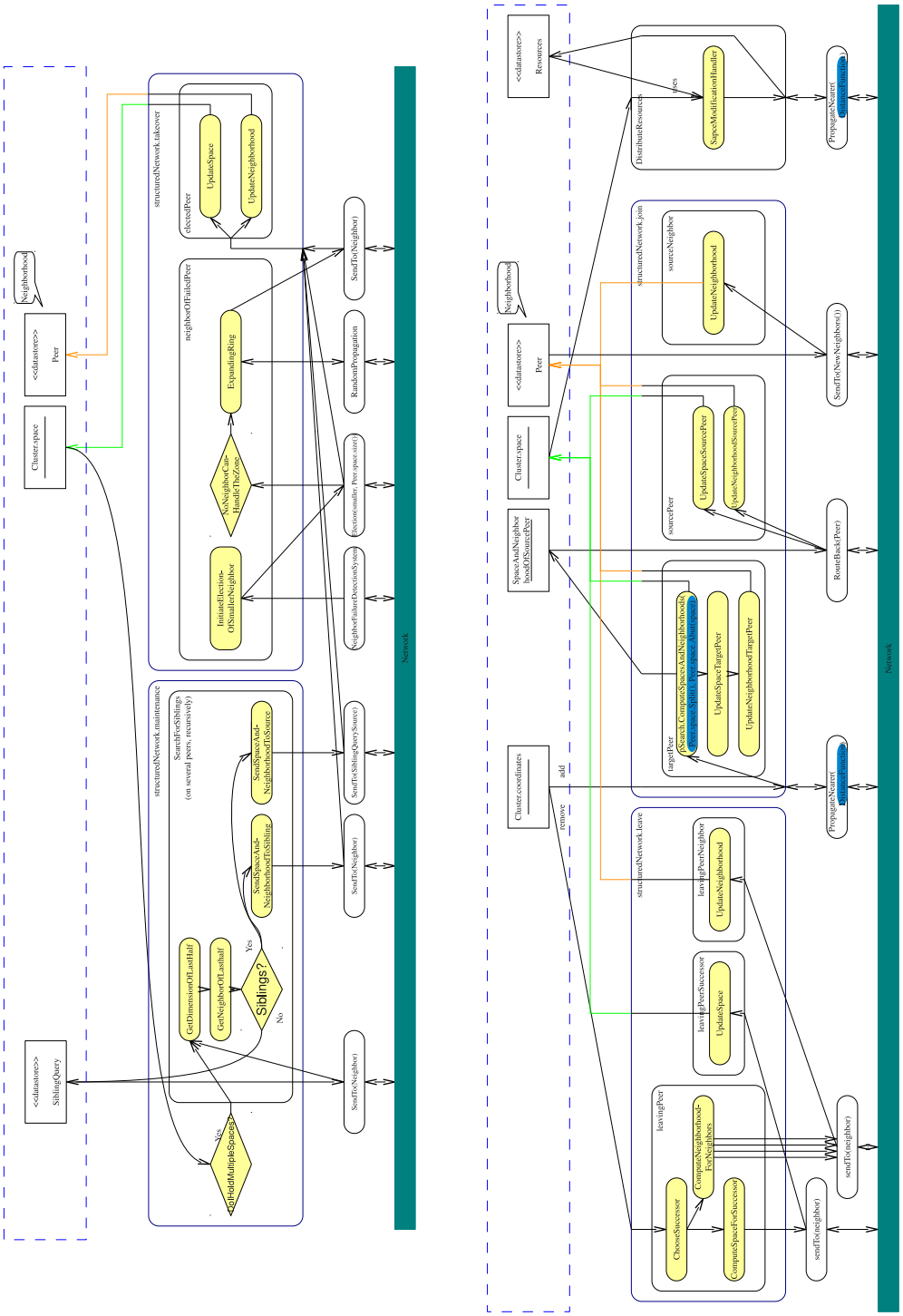


FIG. 5.17 – Diagramme d'activités du package de gestion du réseau structuré dans pSearch

quêtes qui ne devraient pas y être ou qui n'existent plus, et que des index qui devraient y être n'y seront pas. On introduit donc les paramètres *rappel* et *précision* des *sample sets*. Si le paramètre *rappel* n'est pas à 1, des index sont supprimés des *sample sets* (au hasard). Si le paramètre *précision* n'est pas à 1, des index sont ajoutés dans les *sample sets* ; sélectionnés au hasard parmi le corpus de ressources et requêtes du pair décrit par le *sample set*.

- Si *canStructuredNetwork* n'est pas optimale, cela signifie que la structure de données sur laquelle travaillent les algorithmes de résolution de requêtes sera imparfaite. La structure de données relie des clusters de pairs (clusters de clusters de ressources), par proximité sémantique. Si elle est imparfaite, cela signifie que des pairs manqueront, des liaisons manqueront et/ou d'autres seront en trop. On aura donc trois paramètres en interface entre *canStructuredNetwork* et l'implémentation du modèle RI : rappel et précision des liaisons, rappel des pairs. Comme pour *pSearch.ComputeSampleSets*, la valeur de ces paramètres provoque des ajouts/suppressions de liaisons/clusters dans la structure de données.
- Si *DistributeResources* n'est pas optimale, alors la reclustérisation des ressources sur les pairs ne sera pas optimale : des ressources ne seront pas dans le bon cluster. Nous créons donc un paramètre *shuffle* à notre implémentation du modèle RI. La valeur de ce paramètre détermine le nombre de ressources qui seront déplacées de leur cluster vers un autre choisi au hasard.
- Si *combiningTree* n'est pas optimale, la valeur de l'IDF sera approchée. Nous proposons d'introduire un paramètre *précision* au mécanisme de propagation des requêtes, qui permette de biaiser la valeur de l'IDF, pour prendre-en-compte l'imperfection (éventuelle) du mécanisme *combiningTree*.
- *routeBack* : par héritage des mécanismes CAN, le routage des réponses dans pSearch est direct, c'est-à-dire que les coordonnées de l'émetteur sont transportées avec sa requête, permettant ainsi de lui renvoyer la réponse directement.

Finalement, on obtient le diagramme d'évaluation 5.18.

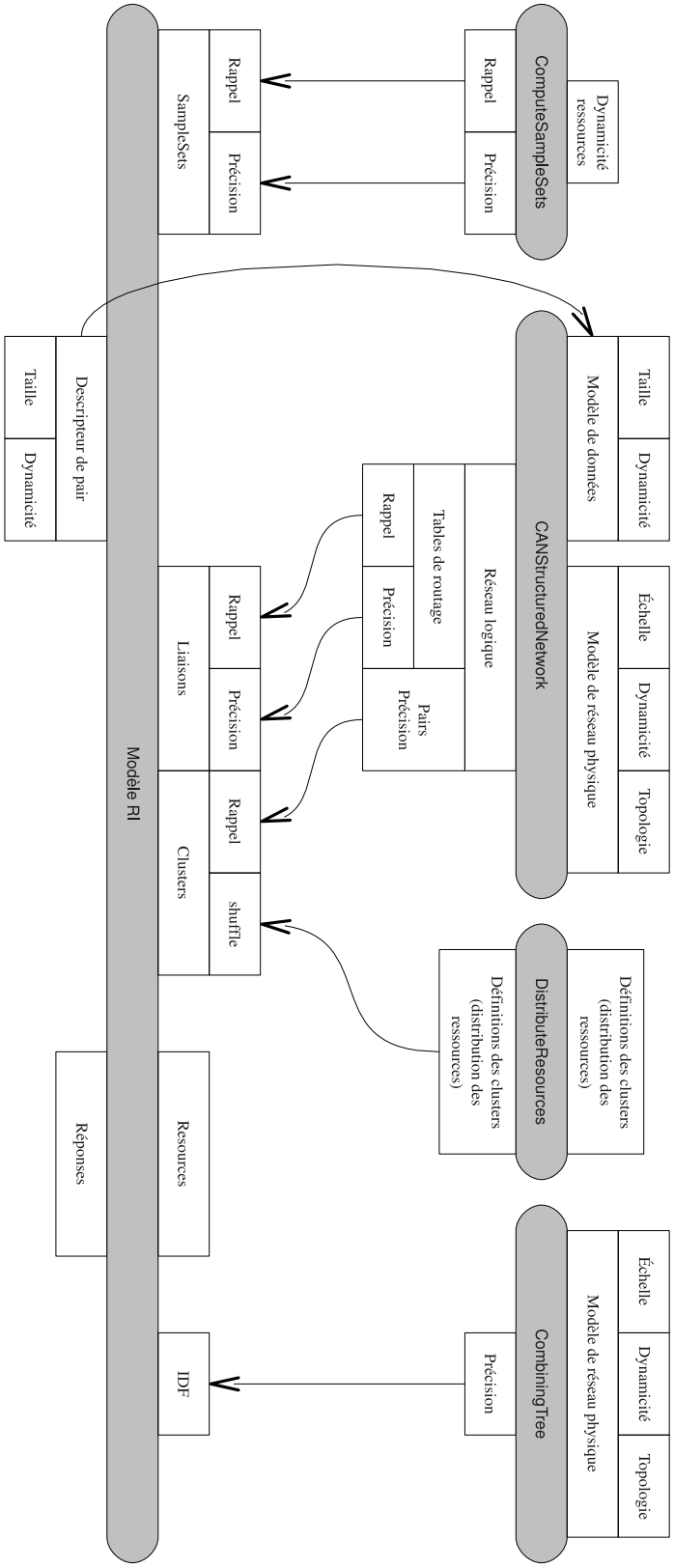


FIG. 5.18 – Diagramme d'évaluation de pSearch

5.5 Conclusion

En résumé

Pour résumer notre démarche, nous avons utilisé des diagrammes d'activités pour extraire les différentes phases des modèles RI de PlanetP et pSearch. Nous résumons ces phases dans la figure 5.19. Cette figure met en évidence trois étapes dans le processus de conception d'un système P2P :

- À partir d'un modèle RI de base,
- Introduire la notion de cluster (indexation, recherche...)
- Choisir une solution P2P
- Optionnellement, faire des adaptations du modèle aux conditions P2P spécifiques à la solution choisie.

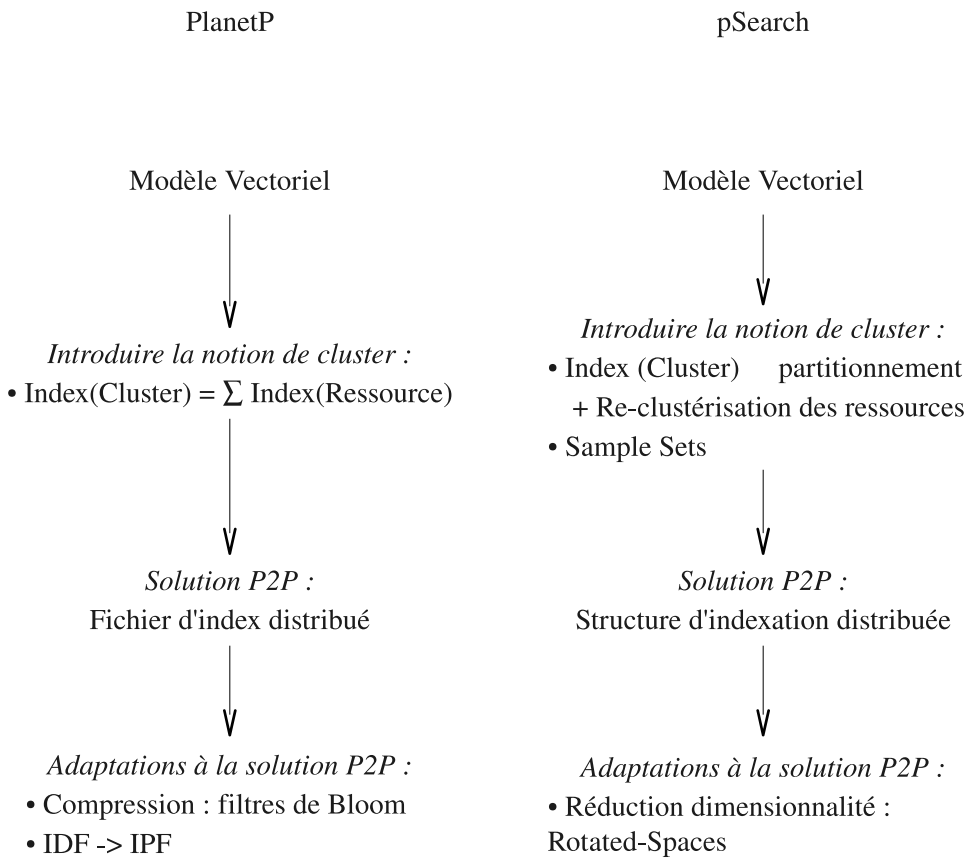


FIG. 5.19 – Comparatif des modèles RI implémentés par PlanetP et pSearch.

Comparer PlanetP et pSearch

On peut ensuite rassembler l'ensemble des options mises-en-place par ces deux systèmes en un arbre de choix des options du modèle RI, présenté sur la figure 5.20. On y voit qu'on peut choisir entre une indexation des clusters suivant le contenu - à la PlanetP - et une répartition du contenu suivant les index - à la pSearch. La mise-en-place de *Sample Sets* est une option valable dans les deux cas de figure. On peut ensuite choisir sa solution P2P, indépendamment du mode de prise en charge des clusters. Enfin, nous mettons en évidence que, alors que la compression par filtres de Bloom et l'utilisation de *rotated spaces* sont clairement des option spécifiques aux solutions P2P « Fichier d'index distribué » et « Structure d'indexation distribuée » respectivement, l'utilisation de l'IPF à la place de l'IDF est utilisable dans les deux cas de figure. Tous les modèles RI correspondant à tous les chemins possibles dans cet arbre peuvent être évalués dans un schéma classique d'évaluation d'un modèle RI ; en faisant tourner une implémentation centralisée sur un jeu de donn ées test, et en comparant les résultats à un oracle.

PlanetP et pSearch utilisant des solutions P2P différentes, faire une comparaison de performances de leurs implémentations n'a pas de sens. Cependant, des ponts existent entre les implémentations de solutions P2P différentes. Par exemple, (Beaumont et al., 2007) propose de maintenir une structure d'indexation distribuée apparentée à celle utilisée par pSearch, par un mécanisme de type gossiping, tel celui utilisé par PlanetP.

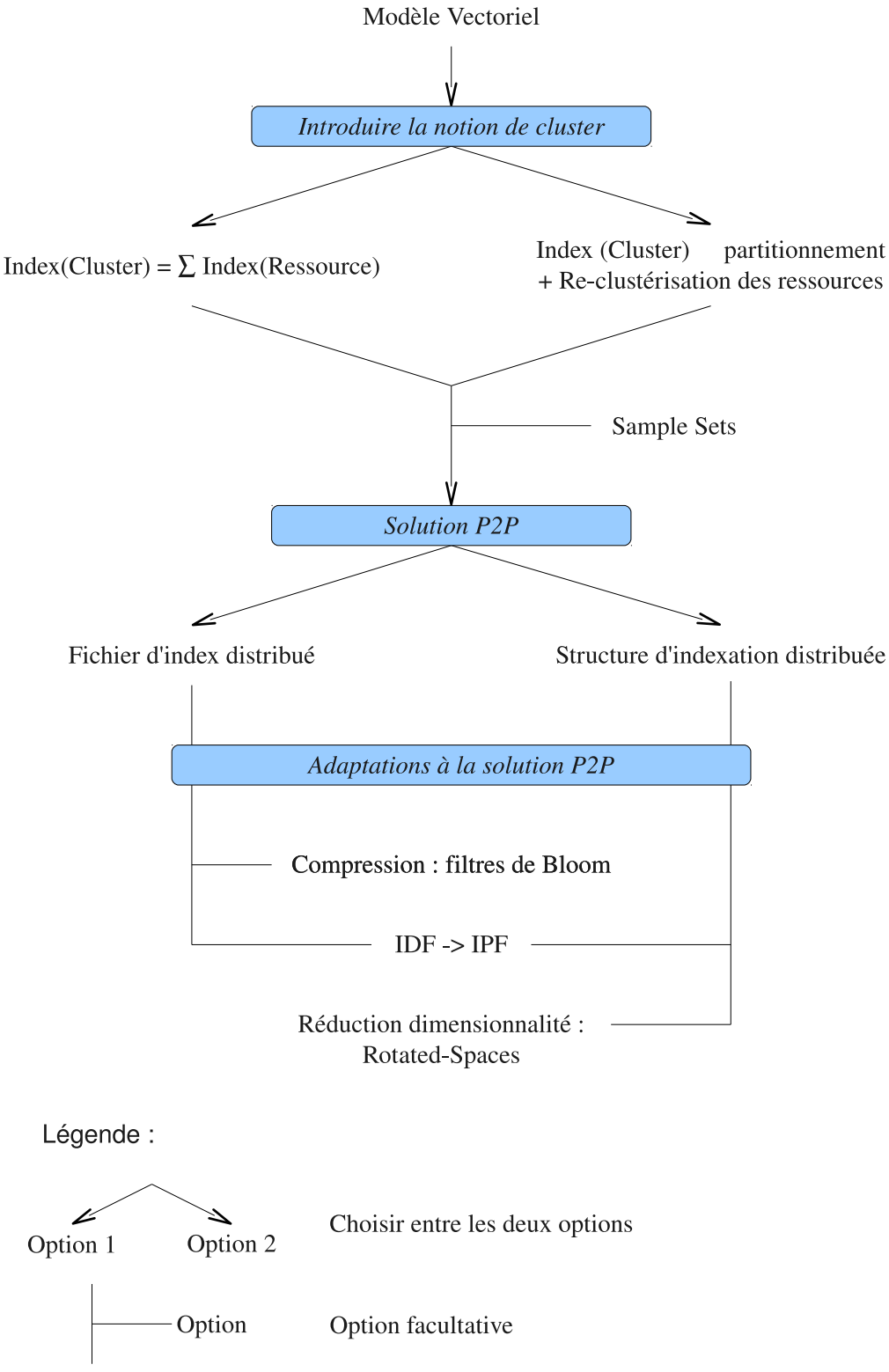


FIG. 5.20 – Arbre des options utilisées dans les modèles RI de PlanetP et pSearch.

Chapitre 6

Contributions à la conception de systèmes P2P

Nous venons d'étudier PlanetP et pSearch, à titre d'exemples. Nous souhaitons maintenant formaliser mieux notre démarche, voir dans quelle mesure elle peut être généralisée à d'autres systèmes P2P et enfin voir comment cette façon d'évaluer - et de comprendre - peut aider à la conception de nouveaux systèmes.

Pour cela, nous rappelons dans une première section que notre démarche repose sur une conception originale de l'articulation entre une application et son implémentation en P2P. Cela pose au cœur du système P2P ce que nous avons appelé le mécanisme P2P : un système P2P est un ensemble de mécanismes P2P qui implémentent une application. Partant de cette idée, pour généraliser notre approche, deux pistes doivent être creusées : recenser l'ensemble des mécanismes P2P disponibles, et comprendre comment ces mécanismes s'assemblent en un système P2P cohérent. Le but est de comprendre quel est l'ensemble des techniques P2P disponibles pour implémenter une application donnée - une technique est une combinaison de mécanismes - et d'évaluer toutes ces techniques, pour situer quelle technique est la plus performante, suivant le type d'application et le contexte dans lequel elle est utilisée.

C'est ce que nous faisons dans les deux section suivantes, à partir des systèmes de notre état de l'art. Tout d'abord, nous analysons la structure des systèmes P2P. L'architecture d'un système P2P peut être vue comme une structure arborescente, dont la racine est l'application implémentée et les feuilles sont des mécanismes P2P. Cette structure nous montre comment des mécanismes P2P doivent être combinés pour former un système complet. Notez que cela ne donne pas toutes les contraintes d'assemblage de mécanismes P2P, mais cela pose les principes de base. Ensuite, nous recensons l'ensemble des mécanismes P2P utilisés dans les systèmes de notre état de l'art. Nous les classons

dans un « arbre des mécanismes P2P », dont *les arrêtes sont des choix de conception* et dont *les feuilles sont des mécanismes P2P*.

Notre dernier objectif est de voir comment cette réflexion peut être utilisée pour aider à la conception de nouveaux systèmes. Le principe est que si l'on peut classer les mécanismes P2P dans un arbre dont les arrêtes sont des choix de conception, alors il doit être possible d'utiliser cet arbre pour concevoir un système P2P. On reboucle ici sur les problématiques d'évaluation. D'après ce que l'on sait des performances des techniques P2P, étant donné un jeu de données et un contexte d'utilisation, il doit être possible de choisir un chemin optimal dans l'arbre des mécanismes P2P. À ce niveau, notre contribution se limite à des pistes de réflexion.

6.1 Articulation entre l'application et la technique P2P

Pour résumer l'approche qui nous a permis d'analyser puis de construire un plan d'évaluation de PlanetP et pSearch, nous avons d'abord remarqué qu'un système P2P est l'implémentation P2P d'une application. Pour chaque système, nous avons donc extrait l'application répartie en P2P : ici, un modèle de Recherche d'Information. Nous avons représenté la spécification de cette application : un ensemble de signatures de méthodes, qui manipulent des variables (paramètres d'entrée et sortie). On peut analyser comment cette spécification est implémentée en P2P : un ensemble de mécanismes P2P implémentent les méthodes et/ou maintiennent la cohérence des variables partagées entre plusieurs pairs. Un mécanisme P2P est donc attaché à un objet ; soit parce qu'il le calcule, soit parce qu'il en maintient la cohérence. L'ensemble des mécanismes P2P forme l'implémentation P2P d'une application.

6.2 Composition de mécanismes P2P : structure des systèmes P2P

Nous allons maintenant entrer dans le détail de la structure des systèmes P2P. Le but est de comprendre comment les mécanismes P2P sont assemblés pour former un système P2P - suivre sur la figure 6.2. Nous commençons par distinguer les services d'indexation et de stockage. Cette première distinction nous permet d'amorcer l'analyse qui nous conduit à reconstituer la structure arborescente des systèmes P2P.

6.2.1 Ne pas confondre les problématiques d'indexation et de stockage

Nous introduisons ici une première distinction entre les problématiques de stockage et d'indexation, en s'appuyant sur quatre exemples, comparés dans le tableau de la figure 6.1.

Le DNS et Google indexent tous deux des sites Web. Le premier permet de retrouver l'adresse d'un site à partir de son URL ; le second permet de retrouver l'URL d'un site répondant sur certains mots-clefs. Pour ce faire, Google met en place un serveur d'indexation (centralisé, donc) ; tandis que le DNS est un service d'indexation décentralisé. On a donc des techniques d'indexation différentes, pour au final indexer un même objet : un site Web, identifié par son URL, qui est stocké de façon centralisée. C'est-à-dire que le lieu physique où est stockée une page Web est un serveur. Ce lieu est choisi par son créateur, et non par une procédure décentralisée. L'ensemble des sites qui offrent les ressources est différent et nettement plus petit que l'ensemble des sites qui y accèdent. L'accès aux sites se fait par l'envoi d'une requête à ce serveur.

On peut comparer de la même façon Napster et Gnutella. Tous deux partagent des fichiers MP3. Mais cette fois, ces fichiers ne sont plus confiés à un serveur, mais au réseau, qui en assure le stockage de manière décentralisée. Des systèmes semblables à Gnutella permettent d'ailleurs de télécharger les fichiers par morceaux, en parallèle depuis plusieurs sites. D'un autre coté, Napster utilise un serveur d'indexation, tandis que dans Gnutella la résolution des requêtes est un service P2P (donc décentralisé).

	Service de stockage	
Service d'indexation	centralisé	décentralisé
centralisé	Google	Napster
décentralisé	DNS	Gnutella

FIG. 6.1 – Classification de quelques systèmes suivant la façon dont sont implémentés les services d'indexation et de stockage.

Nous parlerons de service de stockage et service d'indexation. Cela nous permet de préciser la définition d'un système P2P : on désigne communément comme système P2P un système dans lequel le service de stockage est P2P.

6.2.2 Pour aller plus loin

Nous considérons que l'indexation et le stockage sont des services. À ce titre, le terme « indexation » est trop vague. L'indexation peut en effet servir plusieurs types de service, comme la Recherche d'Information ou une base de données distribuée...

Ensuite, l'index peut être implémenté de façon hiérarchique, en recherchant les clusters de pairs qui répondent à la requête, puis en recherchant des pairs dans ces clusters, puis enfin en recherchant à l'intérieur de ces pairs les ressources qui répondent à la requête. Tant qu'on est dans un contexte P2P, on aura au moins deux niveaux de recherche : les pairs puis les ressources.

À chaque niveau, le service est implémenté par un ensemble de mécanismes distribués, chacun gérant la répartition d'un type d'objet. Nous avons remarqué que dans tous les systèmes que nous avons rencontrés, la fonctionnalité d'un mécanisme distribué peut toujours se ramener à l'une des deux fonctionnalités suivantes :

- maintien de cohérence d'une variable répartie,
- implémentation un calcul réparti.

On peut avoir des sous-mécanismes. Par exemple, dans PlanetP, le gossiping est le mécanisme P2P associé aux descripteurs de pair. Il se décompose en trois sous-mécanismes : push, pull et partial-pull. Ces trois mécanismes sont déclenchés de façon périodique. Mais les (sous-)mécanismes sont souvent liés à des événements. De ce point de vue, le mécanisme de maintien de la structure CAN est un exemple très complet - mécanisme aussi associé aux descripteurs de pair. Il met-en-place quatre sous-mécanismes :

join est abonné à l'événement d'ajout/mise-à-jour d'un descripteur,

leave est abonné à l'événement de suppression d'un descripteur,

maintenance est déclenché de façon périodique, pour réparer la structure du réseau,

takeover est déclenché par un mécanisme de détection de disparition de voisin.

Enfin, au niveau le plus bas, on trouve des algorithmes distribués. Un algorithme distribué est un mécanisme qui prend-en-charge un type de messages sur le réseau, par deux modules, le premier définit comment traiter un message provenant de la couche haute, le second définit comment traiter un message provenant de la couche basse. Cette architecture de base - très classique - permet de définir des comportements très complexes, en mettant en œuvre plusieurs types de messages pour gérer un même type d'objet.

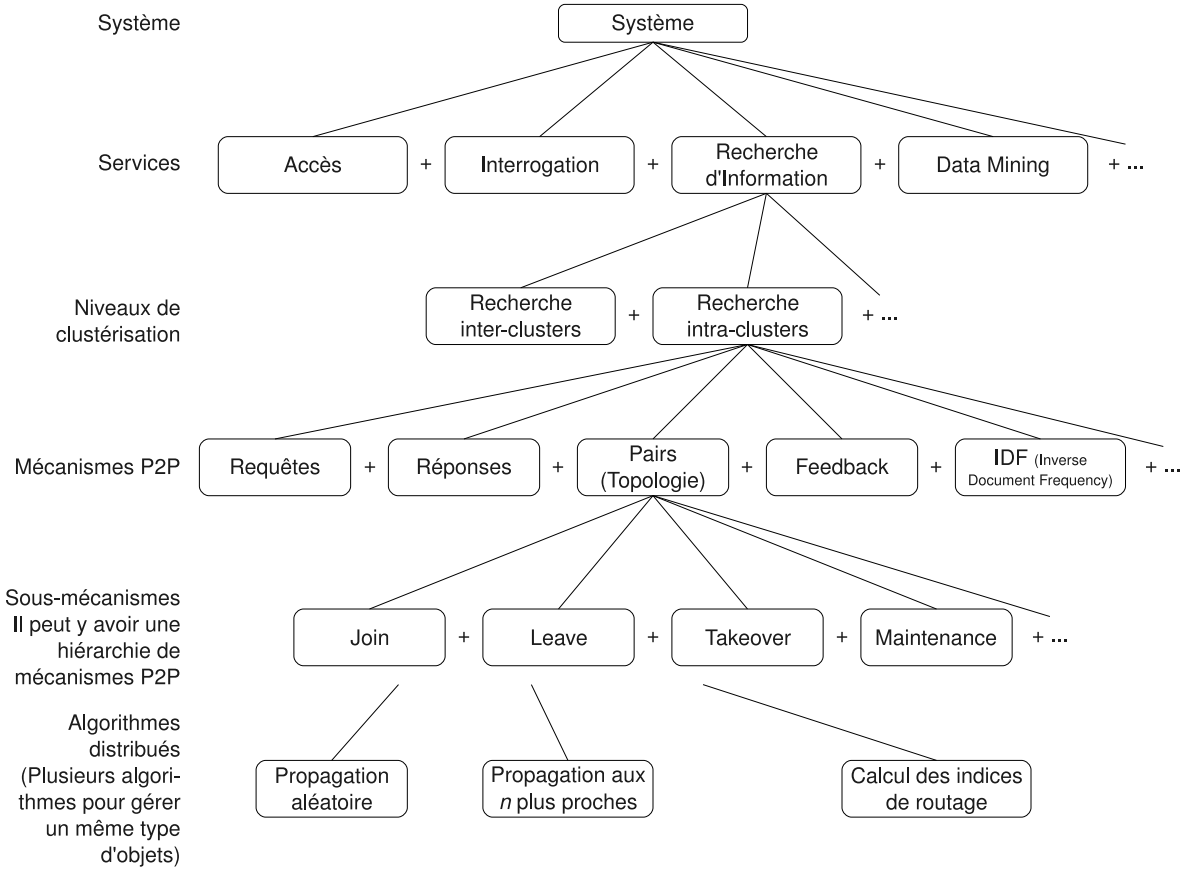


FIG. 6.2 – Architecture des systèmes P2P

6.3 Inventaire des mécanismes P2P : arbre de choix

Un système P2P est composé de plusieurs mécanismes distribués, associés chacun à un type d'objets impliqué dans le processus de Recherche d'Information : pair, requête, ressource, cluster de pair ou de ressources, *KRDB* (Nakauchi et al., 2004), IDF (voir section 5.1 à propos de PlanetP), information de feedback... Nous pensons que tous les mécanismes répartis d'un système P2P sont attachés à un type d'objet et un seul - nous n'avons pas rencontré de contre-exemple. On trouve deux types de mécanismes P2P :

Mécanismes de calcul distribué Un mécanisme de calcul distribué permet de construire l'objet associé de façon distribuée, à partir d'informations extraites ou construites localement. En principe, à la fin d'un algorithme distribué, tous les nœuds qui doivent connaître l'objet calculé le connaissent. La cohérence de l'objet est donc assurée par effet de bord.

Mécanismes de gestion de cohérence La grande majorité des mécanismes P2P est du type « gestion de cohérence ». Notamment, dans tous les systèmes que nous avons étudiés, le mécanisme associé aux requêtes (le « mécanisme de routage ») peut être compris comme un mécanisme de maintien de cohérence : il vise à assurer la cohérence des requêtes sur tous et uniquement les pairs qui peuvent y répondre. On maximise ainsi la qualité des réponses, tout en minimisant le coût de recherche. Parallèlement, le routage des requêtes est défini par rapport à une topologie particulière. La topologie d'un réseau P2P, c'est la donnée de quel pair connaît quel pair ; quelle que soit la façon dont cette connaissance est maintenue (liens TCP, tables de routage...). Pour déterminer qui connaît qui, il faut propager l'information d'existence de chaque pair. Là encore, il s'agit de maintenir la connaissance de l'existence de chaque pair sur tous et uniquement les pairs qui doivent avoir cette connaissance.

On peut observer ces derniers plus en détails. La première distinction est de voir si le mécanisme utilise des index de pair.

Propagation sans index Si aucun index de pair n'est mis-en-place, on peut donner quelques exemples de solutions - qui sont des exemples, et non une liste exhaustive :

- **Propagation aléatoire**. On retrouve ici le mécanisme de propagation des requêtes utilisé par Gnutella, qui construit un arbre de recouvrement partiel du réseau. On peut aussi citer le gossiping utilisé par PlanetP pour maintenir la cohérence des index de pair.
- **Propagation jointe**. On s'appuie sur un autre mécanisme, en joignant l'objet à ses messages. Par exemple, (Nakauchi et al., 2004) propose de joindre les *KRDB* aux messages de requêtes.

- **On sait à qui on doit propager l'information**. Dans pSearch, les *Sample Sets* sont transmis aux voisins. Dans (Nakauchi et al., 2004), le feedback est créé sur l'émetteur des requêtes et utile sur le pair répondant.
- ***Combining tree***. Ce mécanisme est utilisé pour gérer l'IDF dans pSearch.
- ...

On utilise des index de pair pour propager l'information Si des index de pair sont utilisés, on peut recenser trois façons de les utiliser :
- **Réseau étiqueté**
- **Fichier distribué**
- **Structure d'indexation distribuée**

Pour savoir à laquelle des solutions *{Réseau étiqueté, Fichier distribué, Structure d'indexation répartie}* on a affaire, il faut regarder quel type de mécanisme est utilisé pour gérer les descripteurs de pair.

Si on a un mécanisme de calcul distribué des descripteurs, on a affaire à un **réseau étiqueté**. La particularité de cette solution, c'est que le mécanisme de calcul distribué n'influe pas sur la topologie du réseau. L'étiquetage sert à discriminer les voisins, ou les sous-réseaux accessibles par eux, susceptibles de contenir le maximum de pairs destinataires de l'objet propagé. Cette solution est adoptée dans (Crespo and Garcia-Molina, 2002a) et pour la seconde phase de recherche dans pSearch.

Si on a un mécanisme de maintien de cohérence, on se trouve dans un des deux autres cas. Pour décider entre eux, il faut regarder deux choses (cf. figure 6.3) :
- Est-ce que le routage des index utilise les index ?
- Les index sont-ils gérés localement ? Ce que nous appelons « gestion » des index, c'est soit sélectionner ceux qui sont conservés dans les tables de routage par rapport à d'autres qui sont éliminés, soit les trier localement. Remarquez que la sélection peut se ramener à un tri, dans la mesure où on peut ranger des index dans une catégorie *corbeille*.

Si le routage des index n'utilise pas les index et qu'on n'a ni tri, ni sélection ; soit on a un réseau complet (cohérence totale des index), soit on a une topologie aléatoire (cohérence partielle). Dans ce cas, on a affaire à un mécanisme de type **fichier distribué** des index de pair. La résolution se fait localement, c'est-à-dire que l'ensemble des pairs vers lesquels la cohérence de l'objet doit être assurée est calculé sur le pair émetteur de la requête, sur son réplicat du fichier (table de routage locale). On n'a donc pas de propagation à proprement parler.

Dans le cas opposé, si la propagation des index utilise les index (typiquement, parcours de proche en proche) et que chaque pair discrimine localement les index qu'il conserve (ses voisins), où qu'il trie ces index ; dans ce cas on a affaire à ce qu'on appelle généralement un *réseau P2P structuré*, et que nous préférons nommer *implémentation distribué de la structure d'indexation*. Dans ce cas, la résolution s'apparente à un calcul distribué. Chaque pair est assimilé à un nœud de la structure d'indexation ; et trouver

les destinataires d'une information consiste à parcourir la structure d'indexation, donc le réseau P2P.

Il y a une forte corrélation à faire entre cette structure d'indexation et une structure d'index classique, comme on en construit au-dessus d'une table d'une base de données, par exemple. La différence tient en deux points. Tout d'abord, la structure d'index classique construite dans une base de données est un arbre, toujours parcouru à partir de la même racine. Ensuite, la plupart du temps, les objets indexés se trouvent uniquement aux feuilles de l'arbre. À l'inverse, dans un réseau P2P, chaque nœud de la structure d'indexation est associé à un pair. Chacun doit donc pouvoir servir de point de départ, comme de point d'arrivée d'une recherche. (Ratajczak and Hellerstein, 2003) suggère qu'une structure sommet-symétrique satisfait à ces conditions (propriété issue de la théorie des graphes). Il est très probable que cette condition soit nécessaire - mais ce n'est pas prouvé.

Il nous reste deux cas à traiter : si l'on a une propagation guidée par les index sans tri et/ou sélection locale ; ou l'inverse. Si l'on a un au moins de ces deux éléments, cela signifie qu'on ne place pas les descripteurs aléatoirement sur les pairs. Ces mécanismes décident d'une topologie particulière pour le réseau, qui s'appuie sur les index de pair. Ils mettent donc en place une structure d'indexation distribuée. Par contre, si l'un des éléments manque (propagation guidée ou tri/sélection), on ne peut pas garantir la cohérence totale de la structure d'indexation. Nous dirons donc qu'on obtient des structures d'index distribuées faiblement cohérentes.

(Handurukande et al., 2004) met en place une structure d'indexation faiblement cohérente. En effet, la topologie est décidée par un mécanisme tiers non spécifié, typiquement Gnutella. Les index de pair sont construits localement par chaque pair pour chacun de ses voisins, par observation du comportement - qui répond à quelle requête, voir section 6.4.2 sur le feedback. Les voisins sont ensuite triés en au moins deux catégories : les pairs « utiles », qui répondent bien aux requêtes, et les pairs « inutiles », qui répondent peu ou mal aux requêtes initiées par le pair courant. On peut avoir plus de catégories si on différencie des types de requêtes. Finalement, si on retient uniquement les liens vers des pairs utiles, le réseau a une structure, dans laquelle les pairs sont classés par proximité intérêt - expertise. En parcourant ces liens, on se déplace dans l'espace des domaines d'expertise. Nous n'avons pas connaissance de système utilisant une propagation guidée, sans sélection/tri local, sans doute parce que ce type de calcul local étant peu coûteux, on tire peu de bénéfice à s'en priver.

Choix de routage	Gestion locale des tables de routage	
	Sélection aléatoire Pas de tri	Tri et/ou sélection
Routage aléatoire	**Fichier** **distribué**	**Réseau** **faiblement structuré**
Routage des index en fonction des index	**Réseau** **faiblement structuré**	**Réseau** **fortement structuré**

FIG. 6.3 – Classification des types de topologie en fonction des caractéristiques du mécanisme de répartition des descripteurs de pair.

6.4 Recherche d'Information, Feedback et Indexation

D'une certaine façon, nous avons détaillé jusque là la façon dont on peut utiliser des index dans un système P2P. Seulement avant de les utiliser, il faut les construire. Et la construction des index n'est pas un problème de technique P2P ; c'est un problème applicatif. En Recherche d'Information, par exemple, les index de pair décrivent généralement le contenu du pair. Concevoir un langage de description, ou une fonction d'appariement, savoir comment prendre en compte des aspects sémantiques dans ce processus n'est clairement pas un problème de systèmes répartis. Le spécialiste en Recherche d'Information spécifie une application. Le spécialiste en systèmes P2P conçoit une implémentation de cette application.

Nous avons choisi d'étudier la Recherche d'Information comme application type. Nous pensons que c'est une application suffisamment complexe pour que les enseignements qu'on en tire soient applicables à un maximum d'autres domaines applicatifs. En effet, pour comparer les requêtes aux documents, il faut souvent prendre en compte des aspects sémantiques. Pour cela, certains modèles calculent des statistiques sur le corpus global, ou capturent du feedback (retour utilisateur). Bref, l'indexation en Recherche d'Information n'est pas un problème trivial ; elle nous permet de challenger la complexité et les performances des mécanismes P2P. Cependant, nous savons déjà qu'elle ne couvre pas tous les types de problèmes qui peuvent se poser. Par exemple, on n'a qu'une seule mesure de pertinence ; alors que dans le domaine de la recherche de services Web, on a plusieurs critères de pertinence. Nous n'étudierons donc pas ici comment répartir une application utilisant plusieurs critères d'appariement requête-ressource.

6.4.1 Les index en Recherche d'Information

Ici, nous essayons de cibler les enjeux P2P posés par la construction d'index - de pair ou de ressource - en Recherche d'Information. Pour cela, on peut commencer par analyser les différents types d'index :

1. Tout d'abord, un index est porteur d'une information. Cette information peut être de trois types :
   (a) **Un domaine d'intérêt** Caractérise le type de requête qui peut être posé par un pair.
   (b) **Un domaine d'expertise** Soit la qualité de service qu'un pair peut rendre sur un domaine d'intérêt.
   (c) **Une relation intérêt-expertise** Typiquement, une information du type « le pair p_1 répond bien aux requêtes du pair p_2 ».
2. Ensuite, on peut étudier les différents types d'information qui peuvent être utilisés pour construire ces descripteurs. On peut construire un index de pair à partir du contenu du pair, soit les requêtes qu'il émet et les ressources qu'il maintient. On peut aussi construire ces descripteurs par observation du comportement : à quelles requêtes il répond et qui répond à ses requêtes. Nous donnons des détails sur ce second point dans la section traitant du feedback (6.4.2). On a donc trois sources d'informations pour construire les descripteurs de pair :
   - **Contenu : requêtes**
   - **Contenu : ressources**
   - **Observation du comportement : les différents types de feedback**
3. On peut enfin aborder la dernière question, qui va nous raccrocher aux problématiques P2P : **D'où vient cette information ?**. Le lieu où l'information est créée ne correspond pas toujours au lieu où elle est utile. On parle ici à la fois de l'index lui-même où d'une information utile à sa construction. Si une information manque localement, on peut construire l'index à distance puis le rapatrier dans un second temps, ou rapatrier les informations utiles pour le construire localement, ou bien enfin le construire de manière coopérative par l'intermédiaire d'un mécanisme distribué. Quelque soit le cas de figure, on peut reprendre le processus décrit à la section précédente pour choisir le(s) mécanisme(s) P2P adapté(s).

6.4.2 Le feedback : quels retours sur la qualité des réponses ?

Dans cette section, nous identifions trois points de capture d'un retour sur la qualité des réponses fournies par le système (feedback). Nous étudions la portée des informations ainsi obtenues. Cette étude a un objectif double : comprendre où et comment des informations de feedback se mettent en place dans les systèmes de Recherche d'Information ; fournir un outil permettant, face à un problème donné, d'identifier et de capturer les informations utiles à sa résolution.

6.4.2.1 Où capturer du feedback ?

Nous avons identifié trois niveaux de réponse, qui sont :

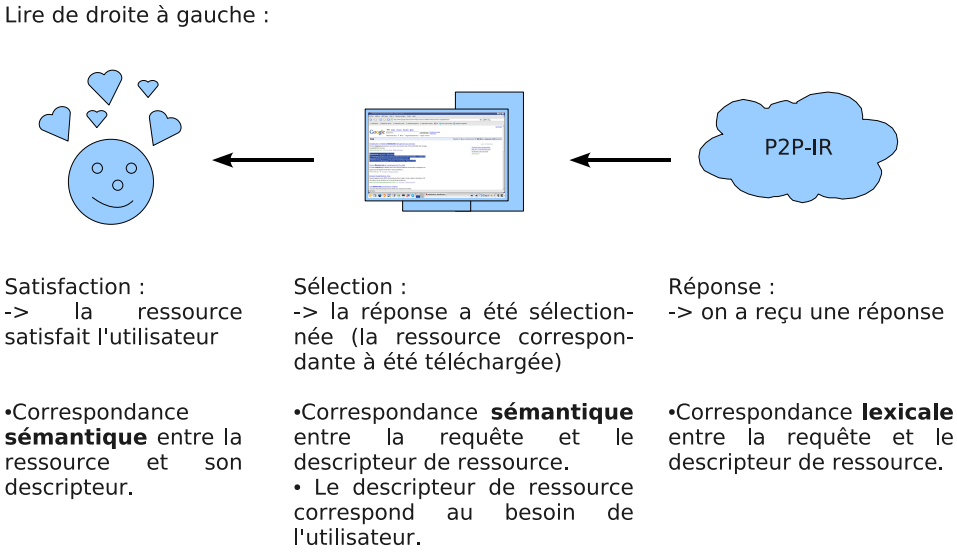


FIG. 6.4 – Informations de feedback : où les capturer, que nous disent-elles ?

On a reçu une réponse : Ceci nous informe qu'il existe une **correspondance lexicale** entre le descripteur de requête et le descripteur de ressource (la réponse)

La réponse a été sélectionnée : L'utilisateur a sélectionné un descripteur de ressource retourné par le système. On ne sait pas si la ressource correspondante l'a satisfait, mais il l'a téléchargée. Ici, l'information essentielle est que le descripteur de ressource a intéressé l'utilisateur. On peut donc considérer qu'il s'agit d'une réécriture pertinente de la requête. On a donc une **correspondance sémantique** entre la requête de l'utilisateur et le descripteur de ressource, dans le sens où l'on a bien interprété ce que recherche l'utilisateur.

La réponse satisfait l'utilisateur : On a obtenu, peu importe le moyen, l'information que la ressource satisfait (ou pas) l'utilisateur. Cette information peut être obtenue de façon explicite - l'utilisateur « note » la ressource - ou implicite - l'application peut déterminer que l'utilisateur est satisfait ; par exemple, si la ressource est un texte, parce qu'il est resté ouvert longtemps. Il faut remarquer que si l'on a une information de *satisfaction*, on a forcément une information de *sélection* correspondante, parce que l'utilisateur a nécessairement consulté la ressource avant d'indiquer si elle est pertinente. On a ici l'information que la réponse (un descripteur de ressource) décrit bien la ressource correspondante. Il s'agit également d'une **correspondance sémantique**.

6.4.2.2 Quelles informations en tirer ?

Supposons qu'on dispose de trois méthodes $Reponse(ressource, requete)$, $Selection(ressource, requete)$ et $Satisfaction(ressource, requete)$, correspondant aux trois niveaux de feedback étudiés plus haut. Voyons explicitement les 6 informations qu'on peut en tirer :

Reponse(ressource, requete) : *information lexicale*, le pair et/ou la ressource cible sait répondre sur les mots clés de la requête.

$\neg$ **Reponse(requete) :** *information lexicale*, le pair et/ou la ressource cible ne sait pas répondre sur les mots clés de la requête.

Selection(ressource, requete) : *information sémantique*, le descripteur de ressource constitue une bonne interprétation de la requête.

Reponse(ressource, requete) $\wedge\neg$ **Selection(ressource, requete) :** *information sémantique*, le descripteur de ressource constitue une mauvaise interprétation de la requête (mauvais « mapping dans l'espace des concepts »).

Satisfaction(ressource, requete) : *information sémantique*, le descripteur de ressource exprime correctement le domaine d'expertise de la ressource.

Selection(ressource, requete) $\wedge\neg$ **Satisfaction(ressource, requete) :** *information sémantique*, le descripteur de ressource exprime mal le domaine d'expertise de la ressource.

Remarque : Reponse et $\neg$ Reponse sont faciles à capturer (un pair cible a répondu ou pas), de même que la Sélection (une ressource a été téléchargée). On peut inférer la non sélection lorsque les réponses sont ordonnées, si une réponse r_i n'est pas téléchargée alors qu'il existe une réponse r_j de rang inférieur (plus loin dans la liste) téléchargée. S'il n'existe pas de réponse r_j, alors on n'a ni la Sélection, ni la non-Sélection. La Satisfaction est une donnée applicative. Elle peut être explicitement obtenue par interrogation de l'utilisateur, ou bien capturée implicitement par observation de son comportement. Parmi les possibilités, on peut citer : observer combien de temps la ressource est utilisée, pour deviner si un texte a été lu ou une vidéo regardée ; voir si une ressource est enregistrée localement ; voir combien de temps elle est gardée en mémoire.

6.4.2.3 Où utiliser ces informations ?

La figure 6.5 résume ces idées, en montrant où peuvent être utilisées les informations de feedback dans le processus d'indexation qui conduit d'une requête à une ressource qui y répond.

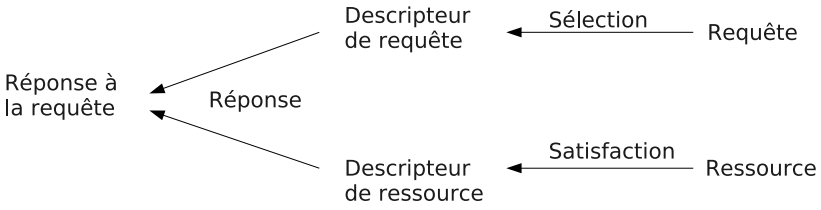


FIG. 6.5 – Du besoin utilisateur à la ressource : à quelles étapes servent les différents types de feedback.

L'information de Satisfaction permet de juger de la qualité du mécanisme de construction des descripteurs de ressources. L'information de Sélection permet de juger de la qualité du mécanisme de construction des descripteurs de requêtes. L'information Réponse permet de juger de la qualité du mécanisme de liaison des requêtes aux ressources correspondantes, par exemple en répondant à la question « arrive-t-on à retrouver toutes les ressources qui répondent à une requête donnée ? ».

6.4.2.4 Conclusion

Nous avons étudié dans cette section les points de capture d'un retour sur la qualité des réponses, les informations qu'on pouvait en tirer et leur utilité dans le processus de résolution des requêtes. Ceci constitue la base théorique de notre travail sur le feedback.

6.4.3 Bilan

Nous avons résumé la réflexion menée dans cette partie dans la figure 6.6. Elle présente un arbre de choix, qui donne les questions qui se posent au concepteur de systèmes P2P et amène à l'ensemble des mécanismes P2P constructibles à partir des techniques extraites de notre état de l'art.

Les cercles désignent des questions à se poser, les rectangles donnent les solutions possibles. Les connecteurs directs représentent un choix entre les réponses possibles à une question. Les connecteurs à angle droit signifient qu'il faut répondre à plusieurs questions en parallèle. L'arbre étant relativement grand, nous avons pris quelques libertés avec les usages pour le compacter. Le cercle *Exécution incrémentale ?* attaché au nœud *Rôle du mécanisme ?* signifie que tout mécanisme peut être piloté pour être rejoué de façon incrémentale. Ceci est typiquement utilisé pour relancer une mise en cohérence d'une requête lorsque les réponses obtenues pour celle-ci sont jugées insatisfaisantes. Le groupe de techniques autour de *Routage aléatoire* et *Tri et/ou sélection* répond aux questions *Le routage est-il aléatoire ?* et *Comment les tables de routage sont-elles gérées localement ?* indiquées au-dessus d'eux.

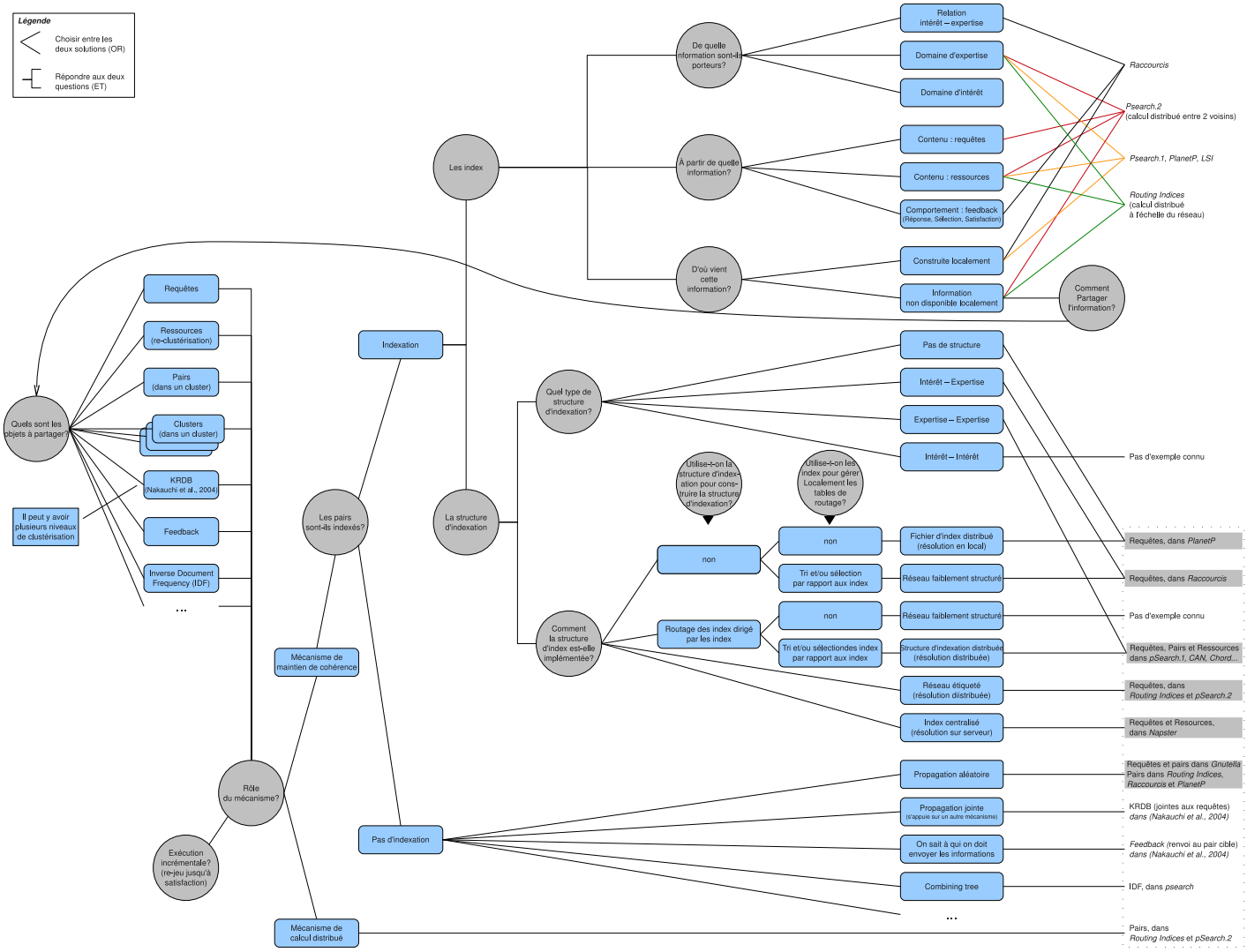


FIG. 6.6 – Arbre de choix pour construire son système P2P

6.5 Perspective : Tableau de choix

Lorsqu'on a mis-en-évidence les différentes techniques de construction d'un système P2P, on peut les évaluer, et construire une méthode de choix de la solution P2P en fonction des caractéristiques du jeu de données. Nous donnons ici à titre indicatif un début de réflexion dans ce sens. Ce travail peut être présenté sous forme d'arbre :

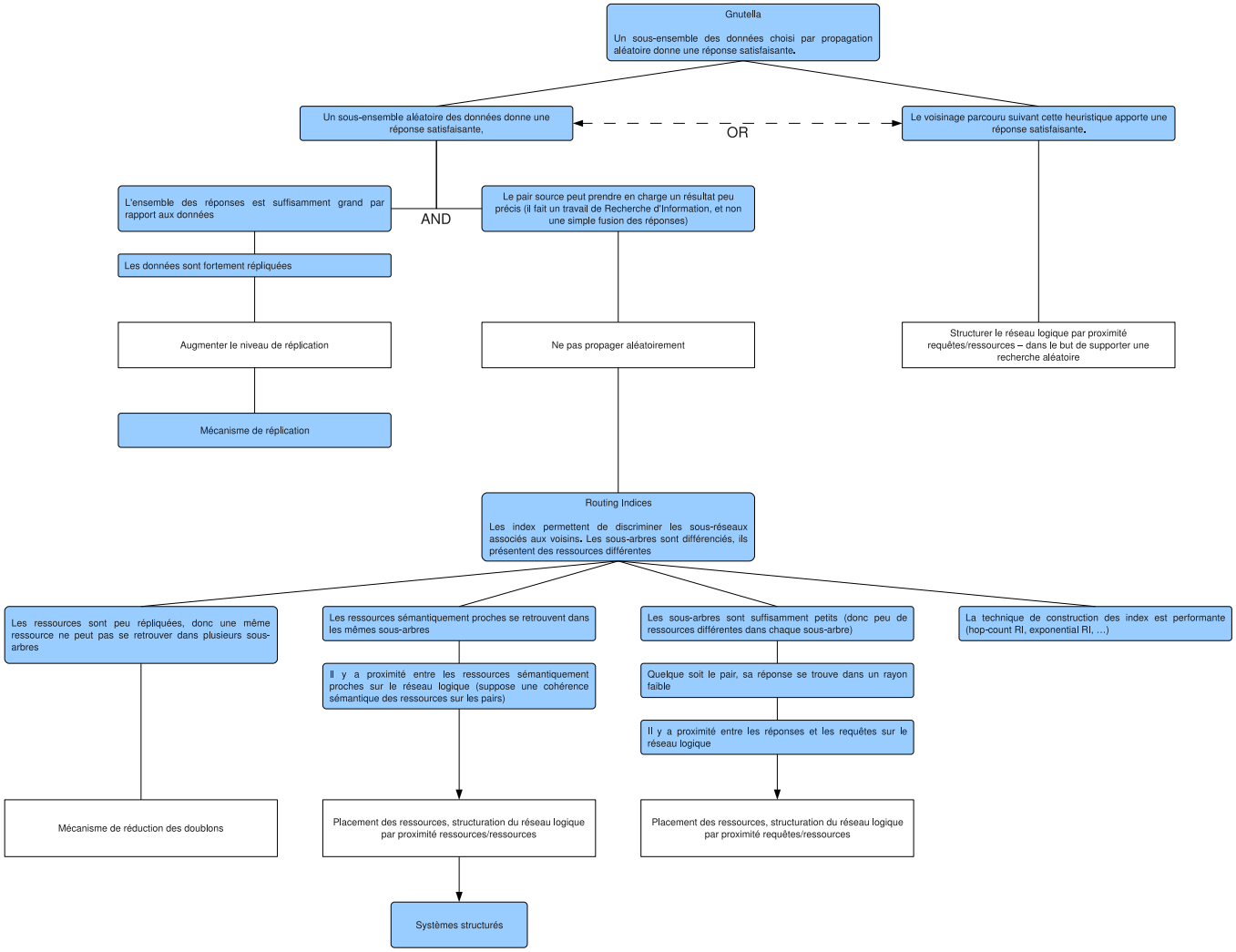


FIG. 6.7 – Arbre de choix d'une solution P2P en fonction des caractéristiques du jeu de données.

Ou sous forme de tableau :

<table>
<tr><th colspan="2">Propriétés
sur les données</th><th>Type de système/solution</th><th>Conséquences
Contraintes ascendantes
(de la couche P2P vers la couche IR)</th></tr>
<tr><td colspan="2">Données fortement répliquées

L'ensemble des réponses est suffisamment grand par rapport à l'ensemble des données
Un sous-ensemble des données choisi aléatoirement donne une réponse satisfaisante</td><td>Principe :
-> Systèmes aléatoires (ex : Gnutella)</td><td></td></tr>
<tr><td rowspan="2">Données moins fortement répliquées

le degré de réplication est trop faible pour utiliser directement un système probabiliste.</td><td></td><td>• Augmenter artificiellement le degré de réplication
• -> mécanismes de réplication
• créer des groupes de données similaires
-> Systèmes clusterisés</td><td rowspan="2">Nécessité d'une représentation unifiée des données</td></tr>
<tr><td>Données naturellement clustérisées sémantiquement (dans les pairs)

ou

Chaque pair partage peu de données</td><td>• créer des groupes de pairs au contenu similaire
-> Systèmes clusterisés
• Étiqueter les pairs pour éliminer lors de la propagation des pairs qui ne sauront pas répondre
-> PlanetP
• Étiqueter le réseau, pour optimiser l'arbre de propagation
-> Routing Indices</td></tr>
<tr><td rowspan="2">Données peu ou pas répliquées

l'ensemble des réponses est beaucoup plus grand celui des ressources</td><td></td><td>• Organiser la topologie du réseau de façon à permettre un parcours ordonné performant des données.
-> Systèmes fortement structurés (dérivés de DHTs)</td><td rowspan="2">Nécessité d'une représentation qui permette le tri (performant) des données

Certaines propriétés de l'espace de nommage peuvent être demandées a priori
(bornes, nombre de dimensions etc...)</td></tr>
<tr><td>Données naturellement clustérisées sémantiquement (dans l'absolu)</td><td>• Étiqueter le réseau
-> Psearch
• Créer des groupes de données similaires
-> SSW</td></tr>
<tr><td colspan="2">Données insuffisamment clustérisées sémantiquement dans les pairs</td><td>• Créer des sous-pairs (plusieurs pairs logiques par noeud)</td><td>Augmente le nombre de pairs dans le réseau</td></tr>
<tr><td colspan="2">Propriétés impliquant
des contraintes trop fortes,

ou

Peu ou pas de propriétés exhibées</td><td>• Concevoir un système «ﬁmoins pair-à-pairﬂ»
-> Super-pairs
-> Serveur distribué
-> Système centralisé</td><td>Tous les problèmes liés aux
systèmes centralisés
(coût, goûlots d'étranglement, sécurité...)</td></tr>
</table>

FIG. 6.8 – Tableau de choix d'une solution P2P en fonction des caractéristiques du jeu de données.

Chapitre 7

Conclusion

Cette conclusion se compose de quatre sections. Les deux premières rappellent les contributions théoriques et pratiques de notre travail. La troisième rappelle les objectifs que nous nous étions fixés pour l'évaluation des systèmes P2P. Enfin, la dernière section présente les perspectives ouvertes par notre travail.

7.1 Les bases théoriques

Cette thèse repose sur une vision originale des systèmes P2P, fondée sur le principe qu'un système P2P est l'implémentation en P2P d'une application. Le terme « application » est ici compris comme un ensemble de spécifications fonctionnelles. Ces fonctions manipulent des objets, qu'elles créent, s'échangent. Nous soutenons qu'implémenter une application sur un réseau P2P, c'est en implémenter une version centralisée sur chaque pair, et y ajouter ce que nous appelons des *mécanismes P2P* ; un mécanisme P2P attaché à chaque type d'objet partagé entre les pairs. Un mécanisme P2P assure le partage entre les pairs ou le calcul réparti, d'un des types d'objets (tous les objets d'un même type) définis par les spécifications applicatives. Bien sûr, il est utile d'adapter les spécifications pour faciliter leur implémentation puis favoriser de bonnes performances en environnement P2P. Sur cette base, nous posons un ensemble de définitions qui permettent de mieux appréhender l'architecture des systèmes P2P.

Nous montrons comment cette vision peut être utile sur deux axes complémentaires : évaluer et concevoir des systèmes P2P. Elle repose fondammentalement sur l'étude de systèmes de Recherche d'Information en P2P, mais nous pensons que, sur de nombreux aspects, elle peut être généralisée aux systèmes P2P en général. Dans la suite de cette conclusion, nous faisons l'inventaire des apports de notre approche, et nous essayons d'en cerner les limites.

7.2 Les apports pratiques

Notre approche permet d'extraire les étapes du modèle RI implémenté par un système RI-P2P. On retrouve des étapes similaires aux étapes d'indexation d'un modèle RI standard. Par exemple, dans un modèle RI standard, on peut appliquer un algorithme de lemmatisation, puis une liste stop-word, puis... Nous avons repéré dans ces étapes deux buts incontournables spécifiques aux modèles destinés à une implémentation en P2P, qui sont d'intégrer la notion de cluster (indexation de clusters, recherche inter-clusters...) et d'adapter le modèle aux conditions P2P (ex : compression ou approximation locale d'objets partagés). L'avantage, c'est que ces étapes peuvent être évaluées comme les étapes d'un modèle RI standard. On peut par exemple évaluer sur une implémentation centralisée dans quels cas il est utile de redistribuer les ressources sur les pairs (suivant

le type de distribution « naturelle » des ressources) ; indépendamment du type d'implémentation P2P. Le résultat sera donc valable pour PlanetP, *Routing Indices*, *Semantic overlay Networks*, pSearch...

Ensuite, notre approche permet d'extraire l'ensemble des mécanismes P2P mis en place pour implémenter un modèle RI. Les relations entre les mécanismes P2P et le modèle RI implémenté sont précisément identifiées ; on les déduit de ce qu'un mécanisme P2P assure le partage d'un type d'objet applicatif entre les pairs. En conséquence, les performances de chacun de ces mécanismes pourront être étudiées indépendamment.

En applicant cette analyse sur l'ensemble des systèmes de notre état de l'art, nous avons obtenu un **inventaire exhaustif des mécanismes P2P et solutions P2P**, basé sur des critères techniques précis. Nous livrons donc toutes les clefs pour élargir cet inventaire si de nouveaux mécanismes sont introduits par de nouveaux systèmes.

7.3 Bilans par rapport aux objectifs

Nous redonnons ici les objectifs que nous nous étions fixés en début de thèse. L'objectif est de faire la synthèse de ceux qui sont atteints, d'étudier le travail nécessaire pour atteindre les autres.

Les items que nous présentons ci-dessous posent notre vision de la problématique d'évaluation des systèmes RI-P2P, au travers d'une série de débouchés qu'on pourrait attendre d'un processus d'évaluation mieux structuré, appuyé sur une meilleure compréhension du domaine. Ces items donnent la philosophie qui guide notre travail.

1. Réinjecter dans l'évaluation les connaissances acquises en RI et en P2P, par exemple les connaissances sur le Gossiping et les filtres de Bloom dans l'évaluation de PlanetP.
2. Capitaliser, créer des bibliothèques des performances des composants.
3. Être exhaustif sur les paramètres à prendre en compte.
4. Disposer d'outils pour comprendre comment chaque paramètre influe sur les performances du système.
5. Les jeux de données n'existent pas toujours, il est nécessaire de savoir quand-même caractériser un système.
6. Sortir du tout simulation ; ne pas s'engager dans la conception d'un simulateur par principe, mais parce qu'on a analysé ce qu'on veut mesurer et comment ce peut être mesuré.
7. Abaisser la complexité de l'évaluation, en divisant le système en composants qui seront évalués indépendamment.

Nous pouvons maintenant analyser dans quelle mesure ces objectifs sont atteints :

1. Réinjecter dans l'évaluation les connaissances acquises en RI et en P2P, par exemple les connaissances sur le Gossiping et les filtres de Bloom dans l'évaluation de PlanetP.

   Notre méthode d'analyse permet de décomposer et évaluer séparément les étapes du modèle RI, indépendamment de l'implémentation P2P. De ce coté, la généricité est donc garantie. Coté P2P, comme nous l'avons déjà évoqué, un même mécanisme peut être utilisé pour remplir différentes fonctionnalités, et les critères de qualité correspondant à ces fonctionnalités ne sont pas nécessairement les mêmes. Nous pensons qu'il y a encore quelque chose à comprendre ici pour garantir la réutilisation des résultats d'évaluation des mécanismes P2P.

2. Capitaliser, créer des bibliothèques des performances des composants.

   On touche ici à la perspective qui découle immédiatement de notre travail : passer à une phase d'analyse/évaluation systématique des systèmes P2P, pour construire une bibliothèque des résultats d'évaluation de tous les mécanismes P2P/techniques RI connus. Cette thèse donne tous les outils pour faire ce travail.

3. Être exhaustif sur les paramètres à prendre en compte.

   Être exhaustif sur les paramètres à prendre en compte est selon nous un besoin fort du domaine ; c'était un frein majeur à la comparaison de PlanetP et pSearch, dans la section 3. Le découpage des systèmes en composants permet d'évaluer des « morceaux » de système plus petits. On maîtrise mieux leur comportement. Nous donnons aussi des moyens de mieux comprendre les inter-actions entre les composants, au travers de l'étude des objets échangés entre eux. On a là deux outils puissants pour mieux comprendre quels paramètres agissent sur le comportement des composants.

4. Disposer d'outils pour comprendre comment chaque paramètre influe sur les performances du système.

   Par le découpage en composants, les objets évalués sont plus petits, moins de paramètres influent sur leur comportement. Il est donc plus facile de comprendre l'action de chaque paramètre. De plus, une analyse des inter-actions entre composants dans le diagramme d'évaluation, peut permettre de tracer les effets des paramètres au sein du système.

5. Les jeux de données n'existent pas toujours, il est nécessaire de savoir quand-même caractériser un système.

   C'est un des apports principaux de ce travail. Il est possible d'évaluer les systèmes sur des jeux de données aussi bien réels qu'artificiels. Et le travail d'évaluation peut être augmenté incrémentalement. On peut complexifier le modèle, en

quel cas on rajoute des configurations à évaluer, qui feront de nouvelles lignes de rapports qualité/coût dans la table des performances du système étudié. On peut de la même façon ajouter des lignes tirées de l'évaluation du système sur une trace nouvellement capturée.

6. Sortir du tout simulation ; ne pas s'engager dans la conception d'un simulateur par principe, mais parce qu'on a analysé ce qu'on veut mesurer et comment ce peut être mesuré.

   l'évaluation par composant permet de choisir pour chacun la méthode la plus adéquate. On va notamment vraissemblablement pouvoir utiliser des méthodes analytiques.

7. Abaisser la complexité de l'évaluation, en divisant le système en composants qui seront évalués indépendamment.

   Si le système est composé de m modules, où le i^eme module a p_i paramètres, au lieu de faire une évaluation à $\sum_m p_i$ paramètres (de l'ordre de $e^{\sum_m p_i}$ situations à évaluer), on fera m évaluations à p_i paramètres (de l'ordre de $\sum_m e^{p_i}$ situations à évaluer).

7.4 Perspectives

Comme nous l'avons déjà évoqué, la perspective qui découle directement de ce travail, c'est un travail d'implémentation des modules RI et mécanismes P2P identifiés, pour en réaliser effectivement l'évaluation, comprendre quelle solution P2P est efficace dans quel contexte. On a quand-même un frein relatif dans ce domaine (ou un facilitateur potentiel), qui est le problème de généricité levée à la section pécédente, concernant les critères de qualité à mesurer lors de l'évaluation d'un mécanisme P2P.

D'un point de vue conception, nous avons fourni un travail d'analyse descendante des systèmes : comment décomposer un système en composants. Un travail intéressant serait d'étudier le sens inverse : comment des mécanismes puis des solutions peuvent être composés pour construire un système. D'un point de vue conception, on a deux types d'inter-dépendances entre mécanismes. Il y a des incompatibilités entre mécanismes et des situations où la mise en place d'un mécanisme appelle fortement la mise en place d'un autre. Comme exemple évident d'imcompatibilité, si la gestion des descripteurs de pair est aléatoire, on ne peut router les requêtes par rapport aux descripteurs. Dans l'autre sens, notre analyse permet de repérer qu'un réseau structuré est mis-en-place. Dans ce cas, il serait sans doute intéressant d'utiliser cette structure pour router plus efficacement les objets à partager - pair, requêtes etc...

Si l'on sait dire quelle implémentation est adaptée à quelle situation, on peut choisir la solution adaptée à une situation. Mais si l'on comprend aussi comment transformer une solution en une autre - par exemple par une étude de l'architecture des systèmes

P2P - on peut décider des transformations à appliquer à un système pour améliorer ses performances. Une piste de recherche intéressante serait donc de concevoir un atelier de génie logiciel dédié aidant à la composition de mécanismes et solutions pour concevoir des systèmes P2P. Une seconde piste de recherche, c'est qu'on peut aussi bien imaginer qu'à moyen terme ce processus d'analyse du contexte/adaptation du comportement soit automatisé. Notre travail laisse donc entrevoir qu'il existera un jour des systèmes P2P dont le comportement s'adaptera dynamiquement aux caractéristiques du jeu de données et au contexte d'utilisation.

Chapitre 8

Bibliographie

Bibliographie

Aberer, K., Klemm, F., Rajman, M., and Wu, J. (2004). An architecture for peer-to-peer information retrieval. In *Worshop on Peer-to-Peer Information Retrieval*.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. In *Reviews of modern Physics*, volume 74, pages 47–97.

Beaumont, O., Kermarrec, A.-M., and RiviÃ¨re, Ã. (2007). Peer to peer multidimensional overlays : Approximating complex structures. In *Principles of distributed systems : 11th international conference, OPODIS 2007*, volume LNCS 4878, pages 315–328, Guadeloupe, French West Indies.

Berry, M. W., Drmac, Z., and Jessup, E. R. (1999). Matrices, vector spaces, and information retrieval. *SIAM Rev.*, 41(2) :335–362.

Buckley, C. (1985). Implementation of the smart information retrieval system. Technical report, Ithaca, NY, USA.

Crespo, A. and Garcia-Molina, H. (2002a). Routing indices for peer-to-peer systems. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23, Washington, DC, USA. IEEE Computer Society.

Crespo, A. and Garcia-Molina, H. (2002b). Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University.

Cuenca-Acuna, F. M., Peery, C., Martin, R. P., and Nguyen, T. D. (2003). Planetp : Using gossiping to build content addressable peer-to-peer information sharing communities. In *12th IEEE International Symposium on High Performance Distributed Computing*. IEEE Press.

Demers, A., Greene, D., Hauser, C., Irish, D., Larsonn, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. (1987). Epidemic algorithms for replicated database maintenance. pages 1–12.

FreenetWebsite. *http ://www.freenet.sourceforge.net*.

Gkantsidis, C., mihail, M., and Saberi, A. (2004). Random walks in peer-to-peer networks.

GoogleWebsite. *http ://www.google.fr*.

Handurukande, S. B., Kermarrec, A.-M., Le Fessant, F. L., and Massoulié, L. (2004). Exploiting semantic clustering in the edonkey p2p network. In *EW11 : Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 20, New York, NY, USA. ACM.

Iamnitchi, A., Ripeanu, M., and Foster, I. (2004). Small-world file-sharing communities. volume 2, pages 952–963.

Jelasity, M., Kowalczyk, W., and van Steen, M. (2003). Newscast computing. Technical report, Vrije Universiteit Amsterdam, Department of Computer Science, Technical Report IR-CS-006.

Klingberg, T. and Manfredi, R. (2002). Gnutella protocol v.0.6.

Li, M., Lee, W.-C., and Sivasubramaniam, A. (2004). Semantic small world : An overlay network for peer-to-peer search. In *ICNP '04 : Proceedings of the 12th IEEE International Conference on Network Protocols*, pages 228–238, Washington, DC, USA. IEEE Computer Society.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7 :72–93.

Nakauchi, K., Ishikawa, Y., Morikawa, H., and Aoyama, T. (2004). Exploiting semantics in peer-to-peer networks. *IEICE TRANS. COMMUN.*, E87-B(7) :1806–1817.

Ratajczak, D. and Hellerstein, J. M. (2003). Deconstructing dhts. Technical report, Intel Research Technical Report IRB-TR-03-042.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). A scalable content-addressable network. pages 161–172.

Salton, G. (1968). *Automatic Information Organization and Retrieval.* McGraw Hill Text.

Saroiu, S., Gummadi, P. K., and Gribble, S. D. (2002). A measurement study of peer-to-peer file sharing systems.

Schmitz, C. and Löser, A. (2006). How to model semantic peer-to-peer overlays ? In *Proc. P2PIR Workshop, Informatik 2006*, Dresden.

Sripanidkulchai, K., Maggs, B., and Zhang, H. (2003). Efficient content location using interest-based locality in peer-to-peer systems. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, pages 2166–2176 vol.3. IEEE Societies.

Stoica, I., Morris, R., David, K., Frans, K., and Hari, B. (2001). Chord : A scalable peer-to-peer lookup service for internet applications. pages 149–160.

Tang, C., Xu, Z., and Dwarkadas, S. (2003). Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM '03 : Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186, New York, NY, USA. ACM.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. In *Nature*, volume 393, pages 409–10.

Witten, I., Moffat, A., and Bell, T. (1999). *Managing Gigabytes : Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco.

Yang, B. and Garcia-Molina, H. (2002). Efficient search in peer-to-peer networks. In *ICDCS*.