# UNIVERSITE DE NANTES

ÉCOLE DOCTORALE

« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE MATHEMATIQUES »

Année 2011

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE NANTES

Discipline : INFORMATIQUE

Spécialité : Bases de Données

*Présentée*
*et soutenue publiquement par*

**Kokou DEDZOE**

*Le 30 Novembre 2011*
*à l'UFR Sciences & Techniques, Université de Nantes*

# Traitement de Requêtes Top-k dans les Communautés Virtuelles P2P de Partage de Données

---

Jury

| | | | |
|---|---|---|---|
| Président | : | _____ | _____ |
| Rapporteurs | : | Bernd Amann, Pr | UMPC Paris 6 |
| | | Gabriel Antoniu, CR1 INRIA | INRIA Rennes |
| Examinateurs | : | David Gross-amblard, Pr | IRISA Rennes |
| | | Pascal Molli, Pr | Université de Nantes |
| | | Philippe Lamarre, Pr | INSA de Lyon |
| | | Patrick Valduriez, DR INRIA | INRIA Sophia Antipolis |

Directeur de thèse : Patrick Valduriez
Co-Directeur de thèse : Philippe Lamarre

Laboratoire: LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE.
CNRS FRE 2729. 2, rue de la Houssinire, BP 92 208 – 44 322 Nantes, CEDEX 3.

N° ED

# Traitement de Requêtes Top-k dans les Communautés Virtuelles P2P de Partage de Données

---

***Top-k Query Processing in P2P Data Sharing Virtual Communities***

**Kokou DEDZOE**

⋈

*favet neptunus eunti*

---

**Université de Nantes**

Kokou DEDZOE

***Traitement de Requêtes Top-k dans les Communautés Virtuelles P2P de Partage de Données***

IV+XVI+102 p.

This document was edited with `these-LINA` LaTeX2e class of the "Association of Young Researchers on Computer Science (LoGIN)" from the University of Nantes (available on : http://login.irin.sciences.univ-nantes.fr/). This LaTeX2e class is under the recommendations of the National Education Ministry of Undergraduate and Graduate Studies (circulaire n° 05-094 du 29 March 2005) of the University of Nantes and the Doctoral School of ≪ Technologies de l'Information et des Matériaux(ED-STIM) ≫.

*Print : thesis.tex – 16/11/2011 – 18:29.*

## Abstract

Top-$k$ queries have two main advantages for peer-to-peer (P2P) data sharing virtual communities. First, they allow participants to rank the results for their queries based on the existing data in the system as well as on their own preferences. Second, they avoid overwhelming participants with too many results. However, existing top-$k$ query processing techniques for P2P systems make users suffer from long waiting times. This becomes even more problematic in overloaded P2P systems. In this thesis, we first revisit the top-$k$ query processing problem and introduce two new measures: the *stabilization time* and the *cumulative quality gap*. These two novel measures, in addition to existing measures, allow for better evaluating the behavior of top-$k$ query processing techniques. We then propose a new family of top-$k$ query processing techniques (ASAP) that allows to return high quality results as soon as possible. Finally, we study the problem of top-$k$ query processing in overloaded systems. As a result, we propose a new approach, called QUAT, that relies on synthetic data descriptions of peers in order to allow peers to prioritize queries for which they can provide high quality results.



**Keywords:** Virtual communities, Peer-to-Peer systems, Top-$k$ query processing, Response time, Stabilization time, Cumulative quality gap, Peer synthetic description.



## Résumé

Dans les communautés virtuelles pair-à-pair (P2P) de partage de données, les requêtes top-$k$ présentent deux avantages principaux. Premièrement, elles permettent aux participants de qualifier les résultats de leurs requêtes par rapport aux données partagées dans le système et ceci en fonction de leurs préférences individuelles. Deuxièmement, elles évitent de submerger les participants avec un grand nombre de réponses. Cependant, les techniques existantes pour le traitement des requêtes top-$k$ dans un environnement complètement distribué présentent l'inconvénient d'un temps d'attente important pour l'utilisateur. Ce temps d'attente est encore très long plus le système est surchargé. Dans un premier temps, nous revisitons le problème du temps d'attente de l'utilisateur dans le traitement des requêtes top-$k$ en introduisant deux nouvelles mesures : le *temps de stabilisation* et la *qualité restante cumulée*. En complément des mesures existantes, elles permettent de qualifier plus précisément le comportement d'un algorithme top-$k$. Dans un deuxième temps, nous proposons une famille d'algorithmes (ASAP), permettant de retourner à l'utilisateur les résultats de bonne qualité le plus tôt possible. Enfin, nous nous intéressons au problème du traitement des requêtes top-$k$ dans le cadre des systèmes P2P surchargés, particulièrement critique pour les solutions classiques, en proposant une nouvelle approche (QUAT). Cette solution fait usage de descriptions synthétiques des données des pairs pour permettre aux pairs de traiter en priorité les requêtes pour les quelles ils peuvent fournir des résultats de bonne qualité.



**Mots-clés :** Communautés virtuelles, Systèmes pair-à-pair, Traitement de requêtes top-$k$, Temps de réponse, Temps de stabilisation, Qualité restante cumulée, Description synthétique d'un pair.


## acm Classification


**Categories and Subject Descriptors :** H.2.4 [**Database Management**]: Systems—*Distributed databases, Query processing.*

# Remerciements

# Table des matières

**Table des matières** vii

**Liste des tableaux** xi

**Table des figures** xiii

**Liste des exemples** xv

*— Corps du document —*

**1 INTRODUCTION GÉNÉRALE** 1
1.1 CONTEXTE . . . 1
1.2 PROBLÉMATIQUE . . . 2
1.3 RÉSUMÉ DES CONTRIBUTIONS . . . 3
1.4 ORGANISATION DU DOCUMENT . . . 3

**2 ÉTAT DE L'ART** 5
2.1 PRÉSENTATION DES SYSTÈMES P2P . . . 5
2.1.1 Caractéristiques des systèmes P2P . . . 5
2.1.2 Les différents types de réseaux P2P . . . 6
2.1.3 Comparaison des réseaux P2P . . . 8
2.1.4 Bilan . . . 9
2.2 MAPPING DE SCHÉMAS . . . 9
2.2.1 Mapping de schémas dans les systèmes d'intégration de données . . . 9
2.2.2 Mapping de schémas dans les systèmes P2P . . . 10
2.3 DIFFÉRENTS TYPES DE REQUÊTES ÉVALUÉES EN ENVIRONNEMENT P2P 13
2.3.1 Les requêtes exactes . . . 13
2.3.2 Les requêtes par mots clés . . . 13
2.3.3 les requêtes complexes . . . 14
2.4 ROUTAGE DES REQUÊTES DANS LES SYSTÈMES P2P . . . 14
2.4.1 Routage des requêtes dans les systèmes non-structurés . . . 14
2.4.2 Routage des requêtes dans les DHTs . . . 18
2.4.3 Routage dans les systèmes de type super-pair . . . 22
2.4.4 Semantic Overlay Networks . . . 23
2.5 REPRÉSENTATION SYNTHÉTIQUE DES DONNÉES DANS LES SYSTÈMES P2P . . . 24
2.6 TRAITEMENT DES REQUÊTES COMPLEXES . . . 25
2.6.1 Les requêtes top-$k$ . . . 25
2.6.2 Les requêtes skyline . . . 32
2.6.3 Autres types de requêtes complexes . . . 34
2.7 CONCLUSION . . . 36

**3 MÉTRIQUES CENTRÉES SUR L'UTILISATEUR POUR MESURER LA PERFORMANCE DES ALGORITHMES TOP-$k$ 39**
3.1 Fondements . . . 40
3.2 Conclusion . . . 41

**4 UNE NOUVELLE APPROCHE DE TRAITEMENT DES REQUÊTES TOP-$k$ DANS LES SYSTÈMES P2P 43**
4.1 MODÈLE DU SYSTÈME . . . 43
4.1.1 Modèle de réseau P2P non-structuré . . . 43
4.1.2 Requêtes top-$k$ . . . 44
4.2 DÉFINITION DU PROBLÈME . . . 45
4.2.1 Énoncé du problème . . . 45
4.3 VUE D'ENSEMBLE SUR LE TRAITEMENT DES REQUÊTES TOP-$k$ PAR L'APPROCHE ASAP . . . 46
4.3.1 Propagation et exécution locale des requêtes . . . 46
4.3.2 Remontée des réponses . . . 46
4.4 APPROCHES BASÉES SUR UN SEUIL DYNAMIQUE POUR LA REMONTÉE DES RÉPONSES . . . 48
4.4.1 Couverture locale des réponses d'un pair . . . 49
4.4.2 Estimation pessimiste de la couverture locale des réponses . . . 50
4.4.3 Fonction de seuil dynamique . . . 50
4.4.4 Réduction des coûts de communication . . . 51
4.4.5 Algorithmes basés sur un seuil dynamique . . . 51
4.4.6 Exemple . . . 52
4.5 ANALYSE THÉORIQUE DES COÛTS . . . 53
4.6 GESTION DES PANNES DES PAIRS . . . 53
4.6.1 Technique efficace d'amélioration de la précision du résultat top-$k$ en cas des pannes 54
4.6.2 Changement de parent . . . 55
4.7 MESURES DE FEEDBACK POUR LES RÉSULTATS INTERMÉDIAIRES . . . 55
4.7.1 Probabilité de stabilisation . . . 56
4.7.2 Proportion des pairs contributeurs . . . 57
4.7.3 Discussion . . . 58
4.8 ÉVALUATION EXPÉRIMENTALE . . . 58
4.8.1 Environnement de simulation . . . 58
4.8.2 Résultats et analyses des performances . . . 61
4.9 CONCLUSION . . . 66

**5 TRAITEMENT DES REQUÊTES TOP-$k$ DANS LES SYSTÈMES P2P SURCHARGÉS 69**
5.1 MODÈLE DU SYSTÈME . . . 69
5.1.1 Modèle de réseau P2P non-structuré . . . 70
5.1.2 Requêtes top-$k$ . . . 70
5.1.3 Description synthétique du contenu des pairs . . . 70
5.2 DÉFINITION DU PROBLÈME . . . 72
5.2.1 Préliminaires . . . 72
5.2.2 Énoncé du problème . . . 73

5.3 TRAITEMENT DES REQUÊTES TOP-$k$ AVEC QUAT . . . . . 73
5.3.1 Architecture d'un pair dans QUAT . . . . . 73
5.3.2 Propagation et exécution locale des requêtes . . . . . 74
5.3.3 Remontée des réponses . . . . . 77
5.3.4 Optimisation . . . . . 78
5.4 INDICES DE ROUTAGES DISTRIBUÉS . . . . . 79
5.4.1 Construction des indices de routages . . . . . 79
5.4.2 Maintenance des indices de routages . . . . . 80
5.5 ÉVALUATION EXPÉRIMENTALE . . . . . 80
5.5.1 Environnement de simulation . . . . . 81
5.5.2 Résultats et analyses des performances . . . . . 83
5.6 CONCLUSION . . . . . 87

**6 CONCLUSION** **89**
6.1 CONTRIBUTIONS . . . . . 89
6.1.1 Nouvelles métriques de mesure de performances des algorithmes top-$k$ . . . . . 89
6.1.2 ASAP . . . . . 89
6.1.3 QUAT . . . . . 90
6.2 TRAVAUX FUTURS . . . . . 90
6.2.1 Expérimentation plus approfondie de QUAT . . . . . 90
6.2.2 Les requêtes top-$k$ skyline . . . . . 90
6.2.3 Top-$k$ sur les données incertaines . . . . . 91

**Bibliographie** **93**

# Liste des tableaux

*— Corps du document —*

2.1 Comparaison des systèmes P2P . . . . . 9
2.2 Comparaison des Géometries de Routage des DHTs . . . . . 22

4.1 **Les paramètres de simulation** . . . . . 60

5.1 **Les paramètres de simulation** . . . . . 81

# Table des figures

*— Corps du document —*


2.1 **Mapping de schéma grâce à un schéma de médiation global** . . . . . 10
2.2 **Un Exemple de l'approche pairwise schema mapping dans Piazza** . . . . . 11
2.3 **Common agreement schema mapping dans APPA** . . . . . 12
2.4 **Approches de routage BFS et BFS modifié** . . . . . 15
2.5 **Random walks** . . . . . 16
2.6 **Géométrie de routage Hypercube et Anneau** . . . . . 19
2.7 **Exemple de géométrie de Routage Butterfly** . . . . . 21
2.8 **Architecture d'Edutella** . . . . . 23
2.9 **Architecture d'un modèle de résumé de données dans PEERSUM** . . . . 25

3.1 **Qualité des réponses top-$k$ sur le pair initiateur / Temps d'exécution** . . 40

4.1 **Exemple d'arbre de propagation de requête d'un système P2P non-structuré** . . . . . 52
4.2 **Impact du nombre de pairs sur la performance d'ASAP** . . . . . 61
4.3 **Impact de $k$ sur la performance d'ASAP** . . . . . 63
4.4 **Impact de la réplication des données sur la performance d'ASAP** . . . . . 63
4.5 **Précision des résultats vs. Taux de panne** . . . . . 64
4.6 **Efficacité de notre mécanisme de calcul de garanties probabilistes** . . . . . 65
4.7 **Impact de la distribution des données sur la performance d'ASAP** . . . . 66

5.1 **Exemple de description synthétique de données d'une table** . . . . . 72
5.2 **Architecture d'un pair** . . . . . 73
5.3 **Routing Indices Tables of Peers** . . . . . 79
5.4 **Logs HTTP : impact de la Charge du système sur la performance de QUAT** 84
5.5 **Données synthétiques : impact de la charge du système sur la performance de QUAT** . . . . . 85
5.6 **Logs HTTP : impact du nombre de pairs sur la performance de QUAT** . 86
5.7 **Logs HTTP : impact de $k$ sur la performance de QUAT** . . . . . 87
5.8 **Logs HTTP : précision des résultats vs. Taux de panne** . . . . . 87

# Liste des exemples

*— Corps du document —*

# Chapitre 1

# INTRODUCTION GÉNÉRALE

Dans ce chapitre, nous présentons contexte et les enjeux qui ont motivé notre travail. Puis, nous définissons la problématique de notre sujet de thèse. Enfin, après avoir résumé les principales contributions, nous présentons le plan du rapport.

## 1.1 CONTEXTE

Dans le cadre de cette thèse, nous proposons une contribution pour le partage de données favorisant les échanges et la constitution de communautés virtuelles par exemple entre différents acteurs. Plus précisément, le domaine d'application plus particulièrement visé est celui de la gestion des problèmes environnementaux. Les acteurs impliqués dans cette problématique sont des scientifiques et des décideurs : géologues, océanographes, biologistes, climatologues, sociologues, économistes, ingénieurs agricoles, urbanistes, politiques, industriels, etc. Le domaine qui nous intéresse fait donc intervenir des utilisateurs qui peuvent être très nombreux et intéressés par différentes formes de collaborations, par exemple le partage de connaissances, d'idées, d'expériences, de données. De plus, ces acteurs, bien qu'ils soient intéressés par diverses formes de collaborations, souhaitent conserver un fort contrôle sur les données qu'ils proposent à la communauté.

Dans un tel contexte, le partage de données ne peut pas être effectué grâce à des systèmes traditionnels de bases de données distribuées [ÖzsuV99], ni par les systèmes d'intégration de données [TRV98, TV01a] ou les bases de données parallèles [ÖzsuV99] car ces systèmes font usage d'un schéma global pour masquer l'hétérogénéité des sources de données. En effet, mettre en place un schéma global dans une communauté virtuelle où les participants peuvent être très nombreux n'est pas réaliste. Une approche pair à pair (P2P) apparaît alors comme une base de solution très intéressante pour concevoir des communautés virtuelles de partage de données efficaces et capables de passer à l'échelle pour un très grand nombre de participants, permettant à la fois l'autonomie, l'hétérogénéité des participants et un fort contrôle des participants sur les données qu'ils proposent à la communauté.

Un service majeur pour une communauté virtuelle P2P de partage de données est le traitement de requêtes de haut niveau, permettant aux participants d'exprimés des requêtes en exploitant la connaissance de leurs structures des données, c.-à-d. leurs schémas. On parle alors de requêtes basées sur le schéma. En effet, dans une communauté virtuelle environnementale par exemple, il est très important pour certains types de participants (à savoir les scientifiques et les décideurs) de pouvoir qualifier le résultat d'une requête. Ces participants ne peuvent pas se contenter d'un résultat qui n'est que partiel au sens où il y a peut être de meilleures réponses dans les pairs qui ne sont pas impliqués dans le calcul de la réponse finale. Une telle application

nécessite donc le support des requêtes complexes de type top-$k$ permettant de retourner les $k$ meilleurs résultats d'une requête répondant aux préférences de l'utilisateur [CG99, HKP01]. Ce type de requêtes permettra donc aux acteurs d'une communauté de pouvoir rechercher des données en fonction de leurs propres préférences au lieu d'une simple recherche par identifiant (exact match) qui ne pourra pas leur permettre d'évaluer la qualité d'une réponse par rapport à une autre. Les requêtes top-$k$ offrent la possibilité aux utilisateurs de qualifier les résultats d'une requête grâce à une fonction de score [CG99, HKP01]. Cette fonction fournie par l'utilisateur permet de trier les résultats par rapport à ses préférences individuelles. Les requêtes top-$k$ permettent également à l'utilisateur de limiter le nombre de résultats que le système doit lui retourner. Cela évite de submerger l'utilisateur par un grand nombre de résultats. Cela présente aussi l'avantage de réduire considérablement le trafic réseau. Ainsi, les résultats retourner à l'utilisateur sont les meilleurs par rapport à la fonction de score. Cependant concevoir un tel service dans un système P2P est difficile en raison des caractéristiques spécifiques du P2P, par ex. la nature dynamique et autonome des pairs. La plupart des techniques conçues pour les systèmes de bases de données distribuées, qui exploitent statiquement le schéma et l'information sur le réseau, sont inapplicables dans les systèmes P2P. Il est donc nécessaire de concevoir un ensemble de nouvelles techniques d'accès aux données qui soient décentralisées, dynamiques et auto-adaptatives. Grâce à ces techniques, le service de traitement de requêtes doit pouvoir faire le routage des requêtes aux pairs pertinents sans avoir besoin d'un catalogue centralisée, et exécuter les requêtes, en particulier les requêtes complexes de type top-$k$ de façon complètement distribuée tout en supportant le comportement dynamique des pairs.

Cette thèse a été effectuée en particulier dans le contexte du projet ANR Dataring [Dat09], dont le principal objectif est d'adresser les problèmes de partage de données P2P dans les communautés en ligne.

## 1.2 PROBLÉMATIQUE

La problématique de cette thèse est le développement de nouvelles approches de traitement des requêtes top-$k$ de manière complètement distribuée (c.-à-d. dont l'exécution ne doit pas dépendre d'un pair ou d'un groupe de pair spécifique) permettant aux utilisateurs de pouvoir rechercher des données selon leurs préférences dans une communauté virtuelle P2P de partage de données. Ces approches doivent permettre aux utilisateurs d'avoir les bonnes réponses à leurs requêtes le plus tôt possible avec un coût de communication raisonnable et ceci malgré l'autonomie des participants de la communauté. Dans notre contexte l'autonomie des participants se décline en deux points principaux. Le premier point est la capacité d'un participant à joindre ou à quitter le système à tout moment. Le deuxième point est qu'un participant doit pouvoir avoir un contrôle sur les données qu'il propose à la communauté. Ces approches doivent aussi pouvoir s'adapter à la charge des pairs étant donné qu'une communauté virtuelle P2P de partage de données peut être exposée à un très grand nombre de requêtes utilisateurs dans un court laps de temps et peut donc être facilement surchargé. Enfin, les approches proposées doivent permettre de passer à l'échelle pour un grand nombre d'utilisateurs.

## 1.3 RÉSUMÉ DES CONTRIBUTIONS

L'objectif principal de cette thèse est de fournir de nouvelles approches de traitement de requêtes top-$k$ complètement distribuées et permettant aux participants d'une communauté virtuelle P2P de partage de données d'avoir les meilleurs résultats à leurs requêtes le plus tôt possible et ceci malgré l'autonomie des participants et la répartition des données. Nos contributions principales sont les suivantes.

Premièrement, nous avons étudié de façon générale les techniques de traitement des requêtes dans les systèmes P2P tout en montrant les insuffisances de ces techniques par rapport à notre problématique.

Deuxièmement, nous avons proposé de nouvelles métriques centrées sur l'utilisateur permettant de mesurer la performance d'un algorithme de traitement de requêtes top-$k$ [DLAV10a, DLAV10b].

Troisièmement, nous avons proposé une solution efficace pour le traitement des requêtes top-$k$ appelé ASAP (As Soon As Possible) [DLAV10a, DLAV10b] fonctionnant sur un modèle de réseau P2P non-structuré et permettant aux utilisateurs d'avoir les résultats de bonne qualité le plus tôt possible. ASAP adresse par ailleurs le problème de pannes des pairs où des pairs peuvent quitter le système au cours de traitement des requêtes. Nous avons évalué la performance de notre solution à l'aide de la simulation, en utilisant PeerSim [JMJV]. Les résultats montrent que notre approche permet à l'utilisateur d'avoir le résultat top-$k$ final nettement plus tôt comparativement aux approches classiques et avec un coût raisonnable.

Quatrièmement, nous avons proposé une extension d'ASAP appelée QUAT [DLAV11] dont l'objectif est de réduire le temps d'attente des utilisateurs dans le traitement des requêtes top-$k$ dans les systèmes P2P surchargés (c.-à-d., les systèmes où les pairs peuvent recevoir un très grand nombre de requêtes dans un très court laps de temps). QUAT fait usage des descriptions synthétiques des données des pairs pour permettre aux pairs en cas de forte charge de requêtes de traiter en premier des requêtes auxquelles ils peuvent fournir des résultats de bonne qualité. Ces descriptions sont également utilisées par des pairs en cas de forte charge de requêtes pour router en premier vers les voisins les requêtes auxquelles ces derniers peuvent retourner des résultats de bonne qualité. Nous avons également évalué la performance de QUAT en utilisant PeerSim [JMJV]. Les résultats montrent que QUAT s'adapte mieux aux systèmes surchargés comparativement aux approches classiques de traitement de requêtes top-$k$.

## 1.4 ORGANISATION DU DOCUMENT

Le reste de notre document est organisé comme suit.

Dans le Chapitre 2, nous présentons l'état de l'art. Premièrement, nous présentons les systèmes P2P de façon générale tout en montrant les caractéristiques des différentes types de réseaux P2P qui existent. Deuxièmement, nous nous intéressons aux différentes approches proposées pour le mapping de schéma dans les systèmes P2P. Puis, nous présentons les différentes techniques de routages dans les systèmes P2P. Ensuite, nous donnons une vue d'ensemble sur les techniques de résumés de données dans les systèmes P2P. Enfin, nous nous intéressons aux approches proposées pour le traitement des requêtes complexes en particulier les requêtes de type top-$k$ dans les systèmes P2P.

Dans le Chapitre 3, nous présentons deux nouvelles métriques centrées sur l'utilisateur per-

mettant d'évaluer l'efficacité d'un algorithme top-$k$.

Dans le Chapitre 4, nous présentons notre nouvelle approche de traitement de requête top-$k$ dans un système P2P. En nous appuyant sur les métriques proposées dans Chapitre 3, nous définissons formellement le problème du traitement d'une requête top-$k$ permettant à l'utilisateur d'avoir les réponses de bonne qualité le plus tôt possible. Ensuite, nous proposons une famille d'algorithmes de traitement de requête permettant de résoudre ce problème. Puis, nous évaluons analytiquement le coût de notre approche en termes de nombre de messages et de volume de données transféré. Enfin, nous évaluons expérimentalement les performances de notre solution ASAP par rapport aux approches phares présentées dans l'état de l'art.

Dans le Chapitre 5, nous présentons une extension de notre solution ASAP appelé QUAT dont le but est de permettre aux utilisateurs d'avoir les résultats de bonne qualité le plus tôt possible même si le système est surchargé. Notre proposition consiste à utiliser des descriptions synthétiques des pairs pour construire des indices de routage permettant de router et de traiter rapidement les requêtes auxquelles les pairs peuvent fournir des résultats de bonne qualité. Ensuite, nous évaluons les performances de cette nouvelle solution par rapport à ASAP en prêtant une attention particulière au contexte des réseaux chargés très problématique pour les solutions précédentes.

Enfin, Dans le Chapitre 6, après avoir rappelé nos contributions principales, nous proposons quelques directions de recherche futures.

# CHAPITRE 2

# ÉTAT DE L'ART

Dans ce chapitre, nous passons en revue les techniques qui ont été proposées pour le traitement des requêtes dans les systèmes P2P. En particulier, nous nous concentrons sur les requêtes basées sur les schémas qui sont essentielles pour les applications avancées de gestion de données. Nous donnons d'abord une vue d'ensemble des réseaux P2P existants, et une comparaison des propriétés des différents types de réseaux P2P dans le cadre de la gestion des données. Puis, nous discutons des approches qui sont utilisées dans les systèmes P2P pour le mapping de schémas. Ensuite, nous présentons les algorithmes qui ont été proposés pour le routage des requêtes. Nous nous concentrons en particulier sur le routage des requêtes dans les réseaux non-structurés et dans les DHT. Ensuite, nous donnons une vue d'ensemble sur les techniques de résumés de données dans les systèmes P2P. Enfin, nous présentons les techniques qui ont été proposées pour le traitement des requêtes complexes dans les systèmes de P2P.

## 2.1 PRÉSENTATION DES SYSTÈMES P2P

Les systèmes P2P sont des systèmes distribués conçus pour le partage des ressources (contenu, stockage, cycles de cpu). Ce paradigme permet de concevoir des systèmes de très grande taille à forte disponibilité et à faible coût sans recourir à des serveurs centraux [VP04]. Tous les systèmes P2P s'appuient sur un réseau P2P pour fonctionner. Les réseaux P2P sont des réseaux overlays, décentralisés, où tous les nœuds, appelés pairs, jouent à la fois le rôle de serveur et de client. L'organisation dans un tel réseau repose sur l'ensemble des pairs, donc il n'y a pas d'entité chargé d'administrer le réseau. En distribuant les données et les traitements sur tous les pairs du réseau, les systèmes P2P peuvent passer à très grande échelle sans recourir à des serveurs très puissants.

Dans cette section, nous présentons d'abord les différentes caractéristiques des systèmes P2P. Ensuite, nous présentons les différents types de réseaux P2P. Nous dressons ensuite un tableau comparatif de ces différents types de réseaux en se basant sur les critères mis en avant dans les systèmes P2P dans un contexte de gestion de données de données dans une communauté virtuelle.

### 2.1.1 Caractéristiques des systèmes P2P

Les caractéristiques des systèmes P2P sont les suivantes :

1. dans un système P2P, les pairs sont très dynamiques et peuvent rejoindre ou quitter le système à tout moment.
2. habituellement dans un système P2P, il n'y a pas de schéma global prédéfini pour décrire les données qui sont partagées par les pairs.

3. dans un système P2P, les réponses aux requêtes sont généralement incomplètes. Cela est dû au fait que certains pairs peuvent être absents au moment de l'exécution d'une requête. En plus de cela, compte tenu de la très grande taille du réseau, la transmission d'une requête à tous les pairs du réseau peut être inefficace.
4. dans un système P2P, il n'y a pas un catalogue centralisé qui peut être utilisé pour déterminer les pairs qui détiennent des données pertinentes pour une requête donnée. Cependant, un tel catalogue est une composante essentielle des systèmes de bases de données distribuées.

### 2.1.2 Les différents types de réseaux P2P

Les réseaux P2P peuvent être classés en trois grandes catégories [MPV] : les réseaux non-structurés, les réseaux structurés et les réseaux super-pairs. Dans cette section, nous décrivons chacune de ces trois classes de réseaux.

#### 2.1.2.1 Les réseaux non-structurés

Dans les réseaux P2P non-structurés, le réseau overlay est créé de manière non-déterministe (ad hoc) et le placement des données est complètement indépendant de la topologie du réseau (c.-à-d. que chaque pair à la possibilité d'avoir le contrôle sur ses données). Dans ce type de système, chaque pair connaît ses voisins, mais il n'a aucune connaissance sur les ressources que possèdent ces derniers. Le routage des requêtes dans ce système est généralement effectué par une propagation des requêtes à tous les pairs qui se trouvent à une certaine distance du pair initiateur.

Dans les systèmes non-structurés, il n'y a aucune restriction sur la manière de décrire les données recherchées (expressivité de la requête), c.-à-d. que les requêtes peuvent être de type lookup (recherche par clé où la clé est par exemple le nom de la donnée recherchée), de type SQL, et d'autres types de requêtes peuvent être également utilisés. Bien que les réseaux non-structurés soient de gros consommateurs de bande passante, ils sont cependant très tolérants aux pannes et ont un coût de maintenance très faible [SW07]. De plus, dans ces types de réseaux, chaque pair est autonome de décider quelles données de ses voisins stockées en cas de réplication.

Nous pouvons citer comme exemples de systèmes P2P qui supportent les réseaux non-structurés : Gnutella [Jov00, JAB01, Gnu06], KaZaA [Kaz06], et FreeHaven [DFM00].

#### 2.1.2.2 Les réseaux structurés

Les réseaux structurés se sont émergés pour résoudre le problème de passage à l'échelle des réseaux non-structurés dû à leur forte consommation de la bande passante. Ils atteignent cet objectif en contrôlant la topologie du réseau overlay et le placement des données. Les données (ou des pointeurs vers les données) sont placées à des endroits spécifiés avec précision et les mappings (correspondances) entre les données et leur emplacement (par exemple un identificateur de fichier est mappé à une adresse d'un pair) sont fournis sous la forme d'une table de routage distribuée.

La table de hachage distribuée (DHT) est le principal représentant des réseaux P2P structurés. Une DHT fournit une interface de table de hachage avec des primitives $put(clé, valeur)$ et $get(clé)$, où clé est l'identifiant d'un objet. Dans une DHT, chaque pair est responsable du stockage des valeurs (contenus des objets) correspondant à un intervalle de clés donné. Chaque pair à la connaissance aussi d'un certain nombre d'autres pairs, appelé voisins, et possède une

table de routage qui associe les identifiants de ses voisins à des adresses correspondantes. Les opérations d'accès aux données dans la plupart des DHTs sont constituées de *lookup*, pour trouver l'adresse du pair $p$ qui contient la donnée recherchée, suivie par une communication direct avec $p$. Dans la phase de *lookup*, plusieurs sauts peuvent être effectués selon le voisinage des nœuds.

Dans les DHTs, les requêtes peuvent être acheminées de manière efficace, puisque le mécanisme de routage permet de trouver le pair responsable d'une clé en $O(logn)$ sauts de routage, où $n$ est le nombre de pairs dans le réseau. Cependant ces types de réseaux présentent trois inconvénients principaux. Premièrement, l'autonomie des pairs est très réduite car un pair ne peut stocker que les valeurs correspondantes à son intervalle de clé. Ce qui fait aussi que les pairs n'ont pas de contrôle sur leurs propres données. Deuxièmement, les requêtes sont généralement limitées à la recherche par clé. Troisièmement, le coût de maintenance dans ces types de réseaux est très élevé compte tenu de leur forte sensibilité aux pannes.

Ces dernières années ont connu une recherche très active pour étendre les capacités des DHT pour faire face à des requêtes plus complexes telles que les requêtes de type intervalles [HIMT], les requêtes de jointures [GS04], et les requêtes top-$k$ [APV07].

Nous pouvons citer comme exemples de systèmes P2P qui supportent les réseaux structurés : Chord [SMK+01], CAN [RFH+], Tapestry [ZHS+04], Pastry [SMK+01], Freenet [CMH+02], PIER [HHL+03], OceanStore [KBC+00], PAST [RD01], et P-Grid [ACMD+03, DHA03].

#### 2.1.2.3 Les super-pair

Les réseaux non-structurés et structurés sont considérés comme des "purs" systèmes P2P car dans ces types de réseaux P2P, tous leurs pairs offrent les mêmes fonctionnalités. En revanche, les réseaux super-pair sont des hybrides entre les systèmes client-serveur et les purs réseaux P2P. Comme les systèmes client-serveur, certains pairs, les super-pairs, peuvent servir de serveurs dédiés pour les autres pairs et peuvent assurer des fonctions complexes telles que l'indexation, le traitement des requêtes, le contrôle d'accès, et la gestion des méta-données. L'utilisation par exemple d'un seul super-pair dans un réseau super-pair réduit ce réseau super-pair au client-serveur avec tous les problèmes liés à l'utilisation d'un nœud central. Comme dans les purs réseaux P2P, les super-pairs peuvent être organisés de façon P2P et communiquer de manière efficace entre eux, et ainsi permettre la distribution ou la réplication de l'information globale à travers tous les super-pairs. Les super-pairs peuvent être élus dynamiquement (par exemple en fonction de la bande passante et de la capacité de traitement) et remplacés en cas de pannes.

Dans un réseau super-pair, un pair initiateur d'une requête envoie simplement la requête, qui peut être exprimée dans un langage de haut niveau, au super-pair auquel il est associé. Ce super-pair peut alors localiser les pairs pertinents, soit directement grâce à son index locale ou indirectement via ses super-pairs voisins.

Les principaux avantages des réseaux super-pairs sont l'efficacité et la qualité de service (l'efficacité est perçue par l'utilisateur en termes de complétude des réponses des requêtes, temps de réponse des requêtes, etc.). Le temps nécessaire pour trouver des données en accédant directement aux indexes des super-pairs est très faible par rapport aux inondations. D'autre part, les réseaux de type super-pair peuvent exploiter et tirer partie des différentes capacités des pairs en termes de puissance de cpu, bande passante, ou capacité de stockage puisque les super-pairs prennent sur eux une grande partie de la charge de l'ensemble du réseau. Par contre, dans les réseaux P2P purs, tous les nœuds sont équitablement chargés indépendamment de leurs capa-

cités. Le contrôle d'accès peut également être mieux appliqué car des informations de sécurité peuvent être maintenus sur les super-pairs. Cependant, l'autonomie des pairs est fortement restreinte car les pairs ne peuvent pas par exemple se connecter librement à tous les super-pairs ou au super-pair de leur choix. De plus, la tolérance aux pannes est généralement faible dans un réseau super-pair car les super-pairs constituent des points de défaillances pour les pairs dont ils sont responsables (toutefois un remplacement dynamique des super-pairs peut alléger ce problème).

Nous pouvons citer comme exemples de réseaux super-pair : Napster [Nap06], Edutella [NSS03, NWQ+02], et JXTA [Jxt06]. Notons aussi qu'une version de Gnutella s'appuie également sur les super-pairs [ATS04].

### 2.1.3 Comparaison des réseaux P2P

Du point de vue de la gestion des données, les principales exigences d'un réseau P2P sont [DGMY03] : l'autonomie, l'expressivité des requêtes, l'efficacité, la qualité de service, la tolérance aux pannes et sécurité. Dans un premier temps, nous décrivons ces exigences, ensuite nous nous basons sur ces exigences pour comparer les différents réseaux P2P dans un contexte de partage de données dans une communauté virtuelle.

Les différents exigences d'un système P2P du point de vue de la gestion des données sont les suivantes :

- **Autonomie :** un pair doit pouvoir être en mesure de joindre ou quitter un système à tout moment sans aucune restriction. Il doit pouvoir être en mesure de contrôler les données qu'il stocke et quels autres pairs sont en mesure de stocker ses données (par exemple ceux qu'il considère comme digne de confiance).
- **Expressivité des requêtes :** le langage de requête devrait permettre à l'utilisateur de décrire les données désirées à un niveau de détail approprié. La forme basique d'une requête est la recherche par identifiant (clé) qui est uniquement appropriée pour trouver des fichiers. Une recherche par mot clé avec le classement des résultats est appropriée pour la recherche de documents. Mais pour les données plus structurées, un langage de type requête SQL est nécessaire.
- **Efficacité :** l'utilisation efficace des ressources du réseau P2P (bande passante, puissance de calcul, de stockage) devraient entraîner une baisse des coûts (de traitement des requêtes) et donc un débit élevé de requêtes, c'est à dire qu'un plus grand nombre de requêtes peuvent être traitées par le système P2P en un temps donné.
- **Qualité de service :** la qualité de service fait référence à l'efficacité du réseau P2P perçue par l'utilisateur, par exemple la complétude des réponses, la cohérence des données, la disponibilité des données, le temps de réponse des requêtes, etc.
- **Tolérance aux pannes :** l'efficacité et la qualité des services doivent être fournies malgré des pannes de pairs.
- **Sécurité :** le caractère ouvert d'un réseau P2P fait de la sécurité un enjeu majeur car on ne peut pas compter sur les serveurs de confiance. Dans le cadre de la gestion des données, le principale challenge en matière de sécurité est le contrôle d'accès qui inclut le respect des droits de propriété intellectuelle sur le contenu des données.

Au vu des exigences des systèmes P2P pour la gestion des données, nous dressons dans le Tableau 2.1 la comparaison des principales catégories de réseaux P2P du point de vue de la gestion des données dans une communauté virtuelle de partage de données.

Tableau 2.1 Comparaison des systèmes P2P

| | **Non structurés** | **Structurés** | **Super-pair** |
|---|---|---|---|
| Autonomie | forte | faible | faible |
| Expressivité de la requête | forte | faible | forte |
| Efficacité | faible | forte | forte |
| QoS | faible | forte | forte |
| Tolérance aux pannes | forte | forte | faible |
| sécurité | moyenne | faible | moyenne |

### 2.1.4 Bilan

Au regard des exigences des systèmes P2P du point de vue de la gestion des données, les systèmes P2P non-structurés offrent une solution intéressante pour la mise en place d'une communauté virtuelle P2P de partage de données. En effet ce type de système permet une meilleure autonomie des nœuds et est très tolérant aux pannes. De plus les systèmes P2P non-structurés ont un coût de maintenance très faible en cas de forte dynamicité des nœuds comparativement aux autres types systèmes P2P. Enfin, ils offrent une meilleure expressivité des requêtes et donc capable de supporter des requêtes complexes.

## 2.2 MAPPING DE SCHÉMAS

L'hétérogénéité sémantique est un problème clé dans tous les systèmes de partage de données à grande échelle, que ce soit un système d'intégration de données ou un système de gestion de données P2P [MBDH05]. Les sources de données présentes dans ces systèmes sont généralement conçues de manière indépendante, et donc utilisent des schémas différents. Pour être en mesure de faire inter-opérer des données de deux sources, le système a besoin d'une correspondance de schéma (*schema mapping*), c.-à-d. un ensemble d'expressions qui indique comment les données d'une source correspondent aux données d'une autre source. Dans les systèmes de base de données relationnelles, le *schema mapping* signifie le processus de définition des équivalences entre les relations et les attributs de deux schémas [BAH+06].

Dans cette section, nous donnons une brève description du *schema mapping* dans les systèmes d'intégration de données. Ensuite, nous discutons des approches qui sont utilisées dans les systèmes P2P pour le mapping de schéma.

### 2.2.1 Mapping de schémas dans les systèmes d'intégration de données

Pour être en mesure d'interroger les sources de données hétérogènes, les systèmes d'intégration de données [TV01b, TRV98] s'appuient généralement sur la définition d'un schéma de médiation globale (voir Figure 2.2). Les schémas locaux des sources de données sont mappés sur le schéma médiateur, et les correspondances entre schémas (c.-à-d. les mappings) sont maintenues sur le médiateur. Les utilisateurs émettent leurs requêtes en termes du schéma de médiation, et le médiateur reformule la requête en sous-requêtes qui sont exprimées sur les schémas locaux et peuvent être exécutées sur des sources de données. Il y a *wrapper* au niveau de chaque sources de données qui fournit des services de traduction entre le schéma de médiation et le langage de requête locale [Ull97].

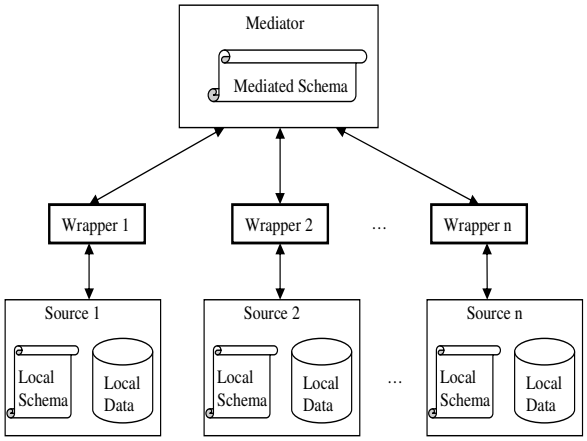



Figure 2.1 **Mapping de schéma grâce à un schéma de médiation global**

Dans les systèmes d'intégration de données, il existe deux principales approches pour définir les mappings : l'approche *Global-As-View* (GAV) et l'approche *Local-As-View* (LAV). Dans l'approche GAV, le schéma de médiation est défini comme une vue des schémas locaux. Dans celle de LAV, les schémas locaux sont définis comme une vue du schéma de médiation [Len02]. Comparativement à l'approche LAV, l'approche GAV offre une autonomie plus élevée aux sources de de données, car elles peuvent définir leurs schémas locaux comme elles le désirent. Cependant, si une nouvelle source est ajoutée a un système qui utilise l'approche GAV, un effort considérable peut être nécessaire pour mettre à jour le code du médiateur. Pour cela, l'approche GAV doit être privilégiée lorsque les sources de données ne sont pas susceptibles de changer. L'avantage de l'approche LAV est que de nouvelles sources de données peuvent être ajoutées au système avec moins d'effort que dans l'approche GAV. Pour cela, l'approche LAV doit donc être privilégiée si le schéma de médiation n'est pas susceptible de changer, c.-à-d. que le schéma médiateur est assez complète pour que tous les schémas locaux puissent être décrits comme une vue celui-ci.

### 2.2.2 Mapping de schémas dans les systèmes P2P

En raison des caractéristiques spécifiques des systèmes de P2P, par exemple la nature dynamique et autonome des pairs, les approches de mapping de schémas qui s'appuient sur un schéma global centralisé ne sont plus applicables dans les systèmes P2P. Ainsi, le problème principal est de fournir une approche de mapping de schémas décentralisée afin qu'une requête formulée sur le schéma d'un pair puisse être reformulée sur le schema d'un autre pair. Les approches qui sont utilisées par les systèmes P2P pour définir et créer les correspondances entre les schémas des pairs peuvent être classées comme suit : le *pairwise schema mapping*, le mapping basé sur les techniques d'apprentissage automatique, le *common agreement mapping*, et le mapping de schéma utilisant les techniques de RI (Recherche d'Information).

#### 2.2.2.1 Pairwise schema mapping

Dans cette approche, les utilisateurs définissent des correspondances entre leurs schémas locaux et tout autre schéma qui apparaît être intéressant pour eux. Le système essaie d'extraire des correspondances entres les schémas qui n'ont pas de correspondances prédéfinies en s'appuyant

sur la transitivité des correspondances qui ont été définies.

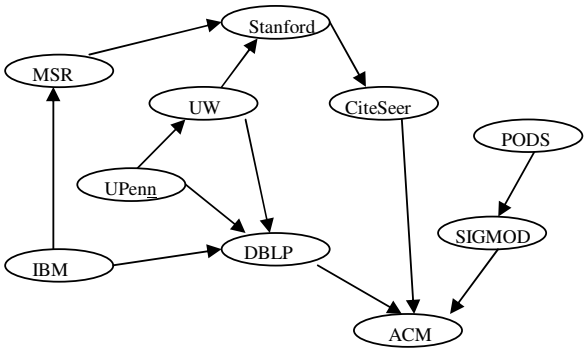



Figure 2.2 **Un Exemple de l'approche pairwise schema mapping dans Piazza**

Piazza [HIMT] suit cette approche (voir Figure 2.2.2.1). Dans Piazza, les données sont partagées en tant que documents XML, et chaque pair a un schéma, exprimé en XMLSchema, qui définit la terminologie et les contraintes structurelles du pair. Quand un nouveau pair (avec un nouveau schéma) entre dans le système pour la première fois, il essai d'établir des correspondances entre son schéma et les schémas de certains pairs. Chaque définition de correspondance commence par un modèle XML qui correspond à un chemin ou un sous-arbre d'une instance du schéma cible, c.-à-d. un préfixe de la chaîne légale dans la grammaire du DTD cible. Les éléments du modèle peuvent être annotés avec des expressions de requête (dans un sous-ensemble de XQuery) qui lient les variables aux nœuds XML dans la source.

Un autre exemple de l'approche *pairwise schema mapping* est le Local Relational Model (LRM) [BGK$^+$02]. Le LRM suppose que les pairs disposent des bases de données relationnelles, et chaque pair $p$ connait un ensemble de pairs avec lesquels il peut échanger des données et des services. Cet ensemble de pairs est appelé connaissances de $p$. Chaque pair doit définir des dépendances sémantiques et des règles de traduction entre ses données et les données partagées par chacun de ses connaissances. Les correspondances définies forment un réseau sémantique, qui est utilisé pour la reformulation des requêtes dans le système P2P.

PGrid suppose aussi l'existence des correspondances entre pairs. Ces correspondances sont initialement construites par des experts qualifiés [ACMH03]. En s'appuyant sur la transitivité de ces correspondances et en utilisant un algorithme de propagation de rumeur (gossiping), PGrid extrait de nouvelles correspondances entre les schémas des pairs entre lesquelles il n'y avait pas de correspondances prédéfinies.

#### 2.2.2.2 Mapping basé sur les techniques d'apprentissage automatique

Cette approche est généralement utilisée lorsque les données partagées sont décrites à partir des ontologies et des taxonomies comme proposé dans le Web sémantique [W3C01]. Cette approche utilise des techniques d'apprentissage automatique pour extraire automatiquement les correspondances entre les schémas partagés. Les correspondances extraites sont stockées sur le réseau, afin d'être utilisées pour le traitement des requêtes futures.

GLUE [DMD$^+$03] est un exemple qui utilise cette approche. Étant donné deux ontologies, pour chaque concept dans l'une, GLUE trouve le concept le plus similaire dans l'autre. GLUE donne des définitions probabilistes bien fondées à différentes mesures de similarités pratiques.

Cette technique, utilise plusieurs stratégies d'apprentissage, dont chacune exploite différents type d'informations, soit sur les données elles mêmes ou sur la structure taxonomique des ontologies. Pour améliorer encore la précision des correspondances, GLUE intègre les connaissances de sens commun (commonsense knowledge) et les contraintes de domaine dans le processus de mapping de schéma. L'idée de base est de fournir des classificateurs pour les concepts. Pour décider de la similarité entre deux concepts $A$ et $B$, les données du concept $B$ sont classées à l'aide du classificateur $A$ et vice versa. Le nombre total de valeurs qui peut être classé avec succès dans A et B représentent la similarité entre $A$ et $B$.

#### 2.2.2.3 Common agreement mapping

Dans cette approche, les pairs qui ont un intérêt commun s'accordent sur une description commune du schéma pour le partage des données. Le schéma commun est généralement mise en place et maintenu par des utilisateurs experts. Par exemple APPA [AM07, AMPV06b, AMPV06c] fait l'hypothèse que les pairs qui souhaitent coopérer, par exemple pour la durée d'une expérience, s'accordent sur une description commune de schéma (Common Schema Description : CSD). Étant donné un CSD, le schéma d'un pair peut être spécifié en utilisant des vues. Ceci est similaire à l'approche LAV utilisée dans les systèmes d'intégration de données, sauf que, dans APPA, les requêtes qui sont posées sur les pairs sont exprimées en termes des vues locales mais pas en termes du CSD. Une autre différence entre cette approche et l'approche LAV est que le CSD n'est pas un schéma global, c'est à dire qu'il est commun à un ensemble limité de pairs ayant des intérêts communs ((voir Figure 2.2.2.3). Ainsi, le CSD ne pose pas de problème de passage à l'échelle du système. Quand un pair décide de partager des données, il a besoin de mapper son schéma local sur le CSD. Étant donné $r_1$ et $r_2$ la définition de deux relations CSD, un exemple de mapping de pair sur le pair $p$ est : $p : r(A, B, C) \subseteq csd : r_1(A, B, C), csd : r_2(C, D, E)$. Dans cet exemple, la relation $r(A, B, C)$ qui est partagée par le pair $p$ est mappée sur les relations $r_1(A, B, C)$ et $r_2(C, D, E)$ qui sont impliquées dans le CSD. Dans APPA, les mappings ou les correspondances entre le CSD et le schéma local d'un pair sont stockés localement sur ce pair. Étant donnée une requête $q$ sur le schéma local, le pair reformule $q$ en une requête sur le CSD en utilisant les correspondances stockées localement.

Dans [MP03], les auteurs présentent AutoMed, un autre système qui repose sur le principe des accords communs pour le mapping de schéma. AutoMed, défini les mappings à l'aide des primitives de transformations bidirectionnelles définies en termes d'un modèle de données de bas niveau.

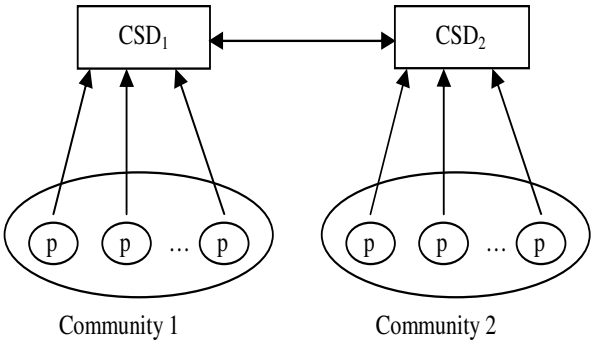



Figure 2.3 **Common agreement schema mapping dans APPA**

#### 2.2.2.4 Mapping de schéma à l'aide techniques de RI

Cette approche extrait des correspondances de schémas au moment de l'exécution de la requête en utilisant des techniques de RI en explorant les descriptions de schémas fournies par les utilisateurs. PeerDB [OST03] suit cette approche pour le traitement des requêtes dans les réseaux P2P non-structurés. Pour chaque relation qui est partagée par un pair, la description de la relation et de ses attributs sont maintenues sur ce pair. Les descriptions sont fournies par les utilisateurs lors de la création des relations, et servent comme une sorte de synonymes des noms des relations et des attributs. Quand une requête est soumise, certains agents sont propagés aux pairs pour découvrir les correspondances possibles et ramener les méta-données correspondantes. En appariant des mots-clés à partir des méta-données des relations, PeerDB est capable de trouver les relations qui sont potentiellement similaires aux relations de la requête. Les relations trouvées sont présentées à l'utilisateur qui a émis la requête, et elle décide ou non de procéder à l'exécution de la requête sur le pair distant qui possède ces relations.

Edutella [NSS03] suit également cette approche pour le mapping de schéma dans les réseaux super-pair. Dans Edutella, les ressources sont décrites en utilisant le modèle de méta-données RDF, et les descriptions sont stockées sur les super-pairs. Lorsqu'un utilisateur émet une requête sur un pair $p$, la requête est envoyée au super-pair de $p$ où les schémas des descriptions stockés sont explorés et les adresses des pairs pertinents sont retournées à l'utilisateur. Si le super-pair ne trouve pas de pairs pertinents, il envoie la requête à d'autres super-pairs de sorte qu'ils recherchent des pairs pertinents en explorant leurs schémas de descriptions. Afin d'explorer les schémas stockés, les super-pairs utilisent le langage de requête RDF-QEL. Le langage RDF-QEL est basée sur la sémantique de Datalog et est donc compatible avec tous les langages de requêtes existantes, et supportant des fonctionnalités de requêtes qui étendent les langages de requête relationnelle usuelle.

## 2.3 DIFFÉRENTS TYPES DE REQUÊTES ÉVALUÉES EN ENVIRONNEMENT P2P

Il existe trois grandes classes de requêtes qui sont évaluées dans un environnement P2P qui sont : les requêtes exactes, les requêtes par mots clés et les requêtes complexes.

### 2.3.1 Les requêtes exactes

Les requêtes exactes (en anglais exact-match queries) permettent de rechercher une donnée à partir d'un identifiant ou d'une clé. La plupart des systèmes P2P proposés au début utilisent de ce type de requêtes pour la localisation des ressources c'est le cas par exemple des systèmes de partage de fichiers [JAB01].

### 2.3.2 Les requêtes par mots clés

Les requêtes par mots clés sont généralement utilisées pour rechercher dans un système P2P des documents contenant un certain mots clés.

### 2.3.3 les requêtes complexes

Les requêtes complexes sont des requêtes de haut niveau, exprimées en exploitant une connaissance de la structure des données des pairs, c.-à-d. leurs schémas. Ils existe plusieurs types de requêtes complexes, nous pouvons citer : les requêtes top-$k$, les requêtes skyline, les requêtes de jointures, les requêtes de type intervalles et les requêtes multi-attributs.

## 2.4 ROUTAGE DES REQUÊTES DANS LES SYSTÈMES P2P

Le problème principal du traitement des requêtes dans les systèmes P2P est comment router les requêtes vers les pairs pertinents c.-à-d. comment localiser les pairs qui possèdent des données relatives à une requête donnée [LW06]. Une fois que la requête est envoyée aux pairs pertinents, elle est exécutée sur ces pairs et les réponses sont renvoyées au pair initiateur de cette requête. Dans cette section, nous décrivons les approches de routage de requêtes dans les systèmes non-structurés, les DHTs, et dans les systèmes de type super-pair.

### 2.4.1 Routage des requêtes dans les systèmes non-structurés

Les techniques de routage des requêtes dans les systèmes P2P non-structurés peuvent être classifiées suivant deux taxonomies.

1. **Taxonomie déterministe ou probabiliste** [LW08, Sch08]. Dans une approche de type déterministe, les requêtes sont propagées de manière déterministe (par exemple, chaque pair qui reçoit une requête l'envoie à tous ses voisins). Par contre, dans une approche de type probabiliste, les requêtes sont propagées de façon probabiliste, aléatoire, ou en se basant sur un classement (*ranking*).

2. **Taxonomie aveugle ou non aveugle** [TR03a, TR03b, TR03c, ZZSL07]. Dans une recherche aveugle les pairs ne gardent pas des informations sur la localisation d'une donnée. Une requête initiée par un utilisateur, est propagée à plus de pairs possibles dans le but de satisfaire cette requête. Contrairement à la recherche aveugle, dans la recherche non aveugle, les pairs maintiennent des méta-données qui permettent de guider la recherche.

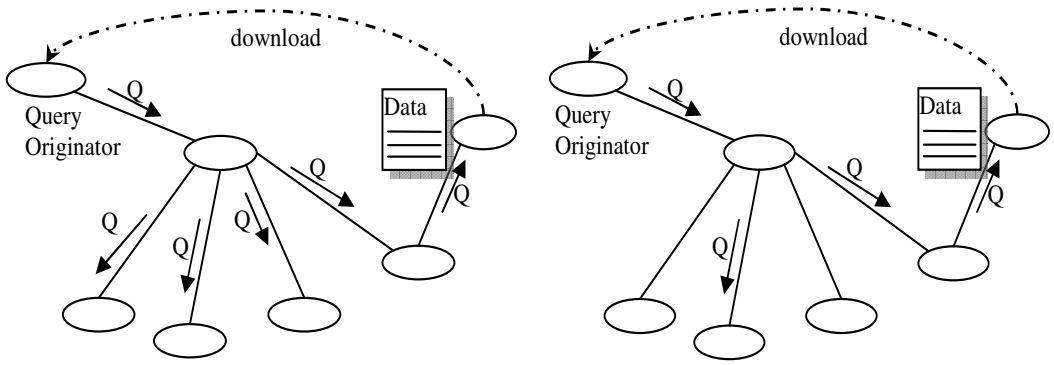
   Dans la suite de ce rapport nous utilisons la taxonomie aveugle ou non aveugle pour classifier les différentes techniques de routage des requêtes dans les systèmes P2P non-structurés.

#### 2.4.1.1 Les algorithmes de recherche aveugles

– **BFS (Breaft-first Search)** : Cette approche est utilisée dans la version de base de Gnutella [JAB01, Jov00] pour la localisation des données. Dans cette approche chaque initiateur d'une requête donnée envoie la requête avec une certaine *ttl* (*Time To Live*) à tous ses voisins (voir Figure 2.4(a)). Chaque voisin recevant la requête décroit la valeur du *ttl* de 1, exécute la requête et la propage à son tour à tous ses voisins si le *ttl* n'est pas nul. Cette technique utilise des requêtes exactes. Dans BFS, chaque pair ayant reçu une requête, l'exécute localement et renvoie les résultats directement au pair initiateur de

cette requête. Cette approche de routage de requête génère un nombre considérable de messages et une consommation importante de la bande passante. Cependant elle permet d'obtenir toutes les réponses recherchées si elles sont présentes dans l'horizon de la requête.

– **Modified-BFS** [KGZY02] : Cette approche est une variation de BFS. Dans ce cas-ci au lieu que les pairs envoient la requête à tous les voisins, celle-ci est uniquement propagée à un sous-ensemble de voisins (voir Figure 2.4(a)). Malgré que cette technique permette de réduire le nombre de messages générés par une requête par rapport à BFS, cela peut toutefois engendrer la perte réponses qu'on pourrait trouver en utilisant le BFS classique. Ce qui fait que l'approche Modified-BFS ne garantit pas d'obtenir toutes les réponses recherchées même si elles sont présentes dans l'horizon de la requête.




(a) BFS (b) BFS Modifié

Figure 2.4 **Approches de routage BFS et BFS modifié**

– **Iterative Deepening** [YGM02] : Dans cette approche de routage, chaque pair initiateur d'une requête effectue des recherches BFS successives en utilisant un ensemble de *ttl* ($ttl_1 < ttl_2 < \ldots < ttl_{max}$). La recherche prenne fin lorsque la réponse à la requête est satisfaite ou lorsque le $ttl_{max}$ est atteint. Cette approche présente une meilleure performance si la requête est satisfaite avec un petit *ttl*. Dans le cas contraire, la charge de réseau engendrée est plus importante que dans le cas BFS standard et le temps de réponse est également très long.

– **Random Walks** [LCC+02] : Cette approche de routage utilise des requêtes exactes. Dans cette approche, pour chaque requête donnée, l'initiateur de la requête envoie $k$ messages de requête (marcheurs à $k$ de ses voisins choisis aléatoirement (voir Figure 2.4.1.1). Les voisins ayant reçus le "marcheur" l'envoient de la même façon à leur tour à leur $k$ voisins. Chaque marcheur contacte périodiquement l'initiateur de la requête pour savoir si la condition de terminaison de la requête est remplie ou non. L'avantage principal de cette approche est qu'il produit au pire cas $k * ttl$ nombre de messages pour router les requêtes, nombre qui est indépendant de la topologie du réseau. Les résultats des simulations présentés dans [LCC+02, TR03b] montrent qu'il y a une réduction considérable du nombre de message par rapport au BFS standard. Cependant, son véritable inconvénient est que sa performance en terme de nombre réponses dépend de la topologie du réseau et des choix de routages

aléatoires. Un autre inconvénient de cette approche est qu'il ne permet aucun apprentissage du réseau. Il faut aussi noter que cette approche ne garantit pas d'obtenir toutes les réponses recherchées même si elles sont présentes dans l'horizon de la requête.

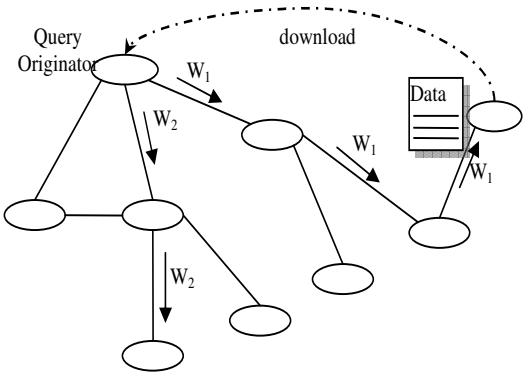



Figure 2.5 **Random walks**

#### 2.4.1.2 Les algorithmes de recherche non aveugles

– **BFS direct** [YGM02] : L'idée principale de cette approche est que l'initiateur d'une requête envoie la requête à ses voisins (directs) qui sont susceptibles de lui retourner plusieurs réponses en se basant sur certaines heuristiques : nombre de réponses retournée par le passé, voisins stables, etc. Ces voisins envoient ensuite la requête à tous leurs voisins comme dans le cas BFS. L'inconvénient de cette approche est que, seul l'initiateur de la requête choisisse intelligemment ses voisins. Du coup, cette approche ne garantit pas d'obtenir toutes les réponses recherchées même si elles sont présentes dans l'horizon de la requête.

– **Intelligent-BFS ou Intelligent Search** [KGZY02] : Cette approche est la version non aveugle de *modified-BFS*. Dans cette approche chaque pair maintient une trace de pertinence des réponses renvoyées pour chacune des requêtes traitées. Lorsqu'un pair reçoit une nouvelle requête, il la route vers ses voisins qui ont été les plus pertinents pour une requête similaire. Comparativement à BFS modifié, BFS intelligent permet d'obtenir plus de réponses mais engendre cependant un grand nombre de messages de routages car les pairs doivent obligatoirement renvoyer les réponses par le chemin où ils ont reçu cette requête afin de mettre à jour les informations de traces. Cette approche se concentre sur la localisation d'une donnée répondant à une requête plutôt qu'à une réduction des messages générés. Cependant elle ne garantit pas d'obtenir toutes les réponses recherchées même si elles sont présentes dans l'horizon de la requête.

– **Adptative probabibilistic search (APS)** [TR03c] : Dans cette approche, pour chaque donnée pertinente à une requête initiée récemment, les pairs maintiennent l'identifiant de cette donnée ainsi que la probabilité d'accès à cette donnée via chacun de leurs voisins. L'initiateur d'une requête envoie $k$ marcheurs à $k$ de ses voisins choisis aléatoirement. Les voisins ayant reçus un marcheur l'envoient seulement à leur voisin ayant la plus grande

probabilité de retourner la donnée recherchée. Cette approche présente une bonne performance comparée au Random walks car le nombre de messages engendrés par une requête est à peu près identique dans les deux cas et que la probabilité de localisation des données par APS est très élevée par rapport au Random walks. Cependant l'inconvénient principale de l'APS est qu'une forte dynamicité du réseau peut réduire de façon significative son habilité à localiser les données pertinentes. Ce qui fait que cette approche ne garantit pas d'obtenir toutes les réponses recherchées même si elles sont présentes dans l'horizon de la requête.

- **Local indices** [YGM02, CGM02] : Dans cette approche, chaque pair indexe les données partagées de ses voisins se trouvant dans un rayon $r$ c.-à-d. tous les voisins situés à une distance inférieure ou égale à $r$. Le routage d'une requête est faite de la même manière que dans l'approche BFS, sauf que la requête est traitée uniquement sur les pairs qui sont à une certaine distance du pair initiateur de la requête. Afin de minimiser les overheads, le nombre de sauts consécutives entre deux pairs qui exécutent la requête doit être $2 * r + 1$. En d'autre termes, la requête doit être exécutée par les pairs se trouvant à une distance $m * (2 * r + 1)$ de l'initiateur de la requête avec $m = 1, 2, \cdots$. L'inconvénient de cette approche est que le coût de maintenance des indexes des pairs est assez importante dans un environnement très dynamique. En effet, une inondation du réseau avec un $ttl = r$ est nécessaire pour la mise à jour des indexes des pairs du réseau quand un pair rentre, quitte le réseau ou met à jour ses données partagées.

- **Distributed Resource Location Protocol (DRLP)** [MK02] : Dans cette approche, les pairs indexent l'emplacement de toutes les données issues de la réponse à une requête. cette indexation est effectuée de manière graduelle. Les pairs ne disposant d'aucune information sur une donnée recherchée, envoie la requête à un ensemble de voisins choisis aléatoirement. Si la donnée est trouvée sur un pair, un message contenant l'emplacement de la donnée est envoyée par le chemin inverse de la requête au pair initiateur. Par contre, si l'initiateur dispose des informations sur la donnée recherchée, il envoie directement la requête au pairs possédant cette donnée. Cette technique engendre initialement un nombre important de messages de routages. Par contre, pour les requêtes suivantes, un seul message pourrait suffir pour localiser la donnée. Cette approche est donc efficiente si les mêmes requêtes sont soumises fréquemment. L'inconvénient principal de DRLP est que dans un environnement très dynamique, les pairs effectuent beaucoup de recherches aveugles. Cette approche ne garantit pas d'obtenir toutes les réponses recherchées même si elles sont présentes dans l'horizon de la requête.

- **Routing Indices (RI)** : Cette approche est proposée dans [CGM02]. Il en existe trois variantes de cette approche : *compound RI*, *hop-count RI* et *exponentially aggregated RI*. Le principe général de l'approche RI est que chaque pair garde des informations sur les thématiques et le nombre de documents que possèdent chacun de ses voisins directs ainsi que les pairs accessibles par ces voisins. Chaque pair initiateur d'une requête envoie uniquement la requête à ses voisins par lesquels il peut accéder à plus de documents. Cette technique utilise des requêtes par mots clés basées sur le contenu au lieu et place des requêtes basées sur l'identifiant des fichiers ou des noms des fichiers contrairement aux approches précédentes. Bien qu'une recherche par RI consomme moins de bande passante, son

principale inconvénient est que la création et la mise à jour des Routing indices nécessite une inondation du réseau ce qui est très coûteux dans un environnement de forte dynamicité. De plus, cette approche ne garantit pas d'obtenir toutes les réponses pertinentes même si elles sont présentes dans l'horizon de la requête.

### 2.4.2 Routage des requêtes dans les DHTs

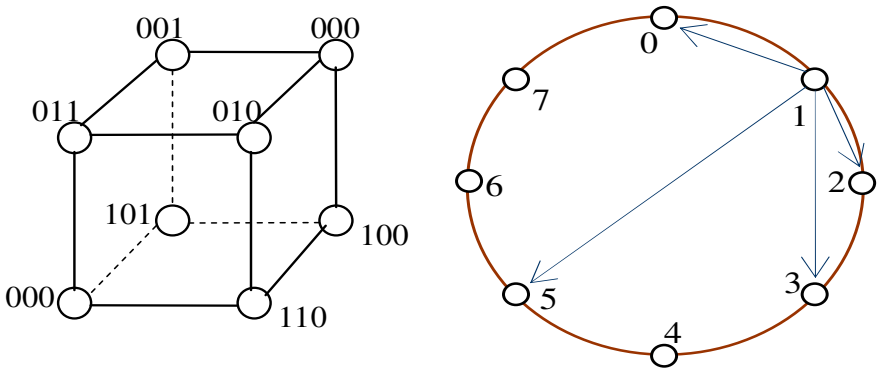
Les tables de hachages distribuées (DHT) fournissent une solution efficace et qui passe à l'échelle pour la localisation des données dans les systèmes P2P. Bien qu'il existe des différences significatives entre la plupart des implémentations des DHTs, elles associent tous cependant une clé donnée à un pair $p$, appelé le responsable de cette clé, en utilisant une fonction de hachage, et peuvent localiser efficacement $p$, généralement en $O(log(n))$ sauts de routage où $n$ est le nombre des pairs [HHH+02]. Les DHTs fournissent généralement une opération $put(key, data)$ qui permet de stocker la donnée $data$ sur le pair qui est responsable de la clé $key$. Pour demander une donnée, il existe une opération $get(key)$ qui permet de router la clé $key$ au pair qui responsable de $key$, et récupérer ensuite la donnée demandée.

La manière dont une DHT route les clés vers leurs responsables dépend de la géométrie de routage du DHT, c'est à dire la topologie utilisée par la DHT pour l'organisation des pairs et le routage des requêtes. Les géométries de routage dans les DHTs peuvent être en arbre, hypercube, papillon, XOR et hybrides [GGG+03]. Dans la suite de cette section, nous décrivons ces diverses géométries et ensuite nous discutons du routage des requêtes.

#### 2.4.2.1 Géométrie de type arbre

L'arbre est la première géométrie qui est utilisée pour l'organisation des pairs dans une DHT et le routage des requêtes entre ses pairs. Dans cette approche, les identifiants des pairs constituent des feuilles d'un arbre binaire de profondeur $log(n)$ où $n$ est le nombre de nœuds de l'arbre. Le responsable d'une clé donnée est le pair dont l'identifiant à le plus grand nombre de bits de préfixes en commun avec la clé. Soit $h(p, q)$ la hauteur du plus petit sous-arbre commun entre deux pairs $p$ et $q$. Pour chaque $i$ avec $1 \leq i \leq log(n)$, chaque $p$ connaît l'adresse d'un pair $q$ tel que $h(p, q) = i$. Cela signifie que, pour chaque $i$ $(1 \leq i \leq log(n))$, le pair $p$ connaît un pair $q$ tel que le nombre de bits de préfixes en commun entre les identificateurs de $p$ et $q$ est $i$. Le routage d'une clé est effectué en utilisant le principe de *"longest prefix matching"* sur chaque pair intermédiaire jusqu'à atteindre le pair qui a le plus de bits de préfixes en commun avec la clé. Les algorithmes de routage de base dans Tapestry [ZHS+04] sont assez similaires à cet algorithme. Dans Tapestry, chaque identifiant est associé à un nœud qui est la racine d'un arbre couvrant, qui est utilisé pour acheminer les messages d'un identifiant donné. La géométrie de routage de Tapestry est étroitement associée à une structure arborescente et dans ce rapport nous la classifions comme telle.

La géométrie de type arbre offre beaucoup de liberté aux pairs dans le choix de leurs voisins : chaque pair à $(2^i - 1)$ options pour le choix de ses voisins dans un sous-arbre de hauteur $i$. Ainsi, chaque pair à au total environ $2^{log(n)*(log(n)-1)/2}$ options pour sélectionner tous ses voisins. Par conséquent, la géométrie en arbre a une bonne flexibilité pour la sélection du nœud voisin. Cependant, la géométrie de type arbre n'est pas flexible pour le routage des messages : il n'y a qu'un voisin à qui un pair peut router un message de requête donnée, c'est le voisin qui à le plus de bits de préfixes en commun avec la clé recherchée.




(a) Exemple de la géométrie de routage Hypercube

(b) Exemple de la géométrie de routage Anneau : les flèches indiquent les voisins du pair 1

Figure 2.6 **Géométrie de routage Hypercube et Anneau**

#### 2.4.2.2 Géométrie de type hypercube

La géométrie de type hypercube est basée sur de partitionnement d'un espace $d$-dimensionnel en un ensemble de zones distinctes et chaque zone est attribuée à un pair. Les pairs ont des identifiants uniques de $log(n)$ bits, où $n$ est le nombre de pairs de l'hypercube. Chaque pair $p$ a $log(n)$ voisins tel que l'identifiant du $i$-ème voisin diffère de celui de $p$ que par le $i$-ème bit. Ainsi, il n'y a qu'un seul bit qui diffère l'identifiant de $p$ de celui de chacun de ses voisins (voir Figure 2.6(a)). La distance entre deux pairs est le nombre de bits sur lequel leurs identifiants diffèrent. Le routage des requêtes s'effectue par une propagation de la clé recherchée via les pairs intermédiaires au pair qui à le minimum de bits différents avec cette clé. Ce type de routage est similaire à celui utilisé dans géométrie de type arbre. Cependant, la différence est que la géométrie de type hypercube permet une réduction des différences entre les bits dans n'importe quel ordre alors que dans le cas de la géométrie de type arbre cette réduction ne peut être effectuée que de gauche à droite.

Le nombre de possibilités de sélection d'une route entre deux pairs dont $k$ bits diffèrent est $log(n) * (log(n) - 1) * \cdots * (log(n) - k)$, c'est à dire que le premier pair sur le chemin à $log(n)$ choix, et que chaque pair suivant à un choix de moins que son prédécesseur. La géométrie de type hypercube offre donc une grande flexibilité pour la sélection des routes. Cependant, pour la sélection de ses voisins, un pair n'a qu'un seul choix. Ainsi, la forme géométrique hypercube n'offre pas de flexibilité dans la sélection des nœuds voisins. A l'opposé de la géométrie de type hypercube, la géométrie de type arbre offre beaucoup de souplesse pour la sélection du nœud voisin, mais pas de flexibilité pour la sélection de la route.

La géométrie de routage utilisée dans CAN [RFH+] ressemble à une géométrie de type hypercube. En effet, CAN utilise un espace de $d$ dimensions qui est divisé en $n$ zones et chaque zone est occupée par un pair. Lorsque $d = log(n)$, les ensembles des voisins dans CAN sont similaires a ceux d'un hypercube de dimension $log(n)$.

#### 2.4.2.3 Géométrie de type anneau

La géométrie de type anneau est basée sur un espace circulaire à une dimension de telle sorte que les pairs sont ordonnés sur un cercle dans le sens des aiguilles d'une montre et ceci selon leurs identifiants (voir Figure 2.6(b)). Chord [SMK+01] est un exemple de protocole DHT qui repose sur cette géométrie pour le routage des requêtes. Dans Chord, chaque pair a identifiant de $m$ bits, et le responsable d'une clé est le premier pair dont l'identifiant est égal ou suit cette clé sur l'anneau. Chaque pair maintient les adresses de $log(n)$ autres pairs de l'anneau de telle sorte que $i$-ème voisin soit le pair dont la distance de $p$ sur l'anneau (dans le sens des aiguilles d'une montre) est $2^{i-1}mod(n)$. Par conséquent, tout pair peut router une clé vers le responsable de cette clé en $log(n)$ sauts puisque chaque saut divise la distance vers la destination de moitié.

Bien que la version originale du protocole Chord définie un ensemble spécifique de voisins pour chaque pair, la géométrie anneau n'a pas besoin nécessairement d'une telle rigidité dans la sélection du nœud voisin. Dans la géométrie anneau, chaque pair $p$ peut choisir son $i$-ème voisin à partir des pairs dont la distance de $p$ dans le sens des aiguilles d'une montre sur le cercle est dans l'intervalle $[2^{i-1}mod(n), 2^{i}mod(n)]$. Cela implique que dans la géométrie anneau, chaque pair a $2^{i-1}$ options de sélection de son $i$-ème voisin. Ainsi, en terme de flexibilité de la sélection du nœud, il y a un total d'environ $2^{(log(n)-1))*((log(n)-1)/2}$ options pour chaque pair.

#### 2.4.2.4 Butterfly

La géométrie butterfly est une extension du réseau butterfly traditionnel avec des propriétés de passage à l'échelle des DHTs. Viceroy [MNR02] est un exemple de DHT qui utilise cette géométrie pour l'organisation des pairs et pour la localisation efficace des données. Les pairs d'un butterfly de taille $n$ sont partitionnés en $log(n)$ de niveaux et $\frac{n}{log(n)}$ lignes (voir Figure 2.4.2.4). Les pairs de chaque ligne sont reliés les uns aux autres par les liens successeur/prédécesseur. Le nombre de pairs sur chaque ligne est $log(n)$, donc une recherche séquentielle sur chaque ligne est effectuée en $O(log(n))$. En plus des liens successeur/prédécesseur, chaque pair à des liens vers les pairs d'autres lignes. Les liens entre les lignes sont disposés de telle manière que la distance entre un pair se trouvant au niveau 1 d'une ligne à tout autre ligne soit égale à $log(n)$. Le routage d'une requête dans la géométrie butterfly est effectué en trois étapes comme suit. Premièrement, la requête est transmise de façon séquentielle au pair se trouvant au niveau 1 de la même ligne que l'initiateur de la requête. Cela s'effectue en $O(log(n))$ sauts de routage. Deuxièmement, du niveau 1, la requête est routée en $O(log(n))$ de sauts de routage vers la ligne à laquelle se trouve le pair destinataire. Troisièmement, à partir de cette ligne (la ligne où se trouve le destinataire), la requête est routée séquentiellement vers le pair destinataire. Notons que chacune de ces étapes se fait en $O(log(n))$, donc le temps total de recherche de routage est de l'ordre de $O(log(n))$.

L'avantage principal de la géométrie butterfly est que la taille de la table de routage par pair, c'est à dire le nombre de voisins de chaque pair, est à la fois petit et constant, alors que dans la plupart des autres géométries, cette taille est de $O(log(n))$. Cependant, la géométrie butterfly n'est pas flexible en matière de sélection de nœuds voisins et de sélection de routes car un pair n'a qu'un seul choix de sélection de ses voisins ou d'une route.

#### 2.4.2.5 XOR

L'approche XOR utilise comme topologie un arbre symétrique unidirectionnel pour structurer les pairs du réseau P2P. Kademlia [MM02] est un exemple de DHT qui utilise la géométrie XOR

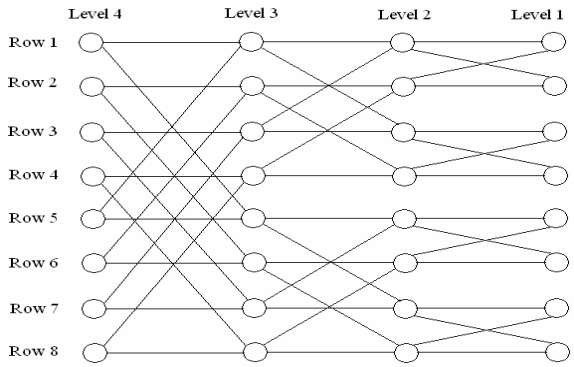



Figure 2.7 **Exemple de géométrie de Routage Butterfly**

pour le routage des requêtes. Dans Kademlia, la distance entre deux pairs est calculée comme la valeur numérique du OU exclusif (XOR) de leur identifiant. Chaque pair $p$ a $log(n)$ voisins, où le $i$-ème voisin est un pair dont la distance XOR vers $p$ est dans l'intervalle $[2i, 2i+1)$. La géométrie XOR offre la même flexibilité en matière de sélection des nœuds voisins comme la géométrie de type arbre. En outre, en termes de sélection de routes, la géométrie XOR peut réduire les différences entre les bits dans n'importe quel ordre, et ne nécessite donc pas la fixation stricte des bits de gauche à droite comme dans le cas de la géométrie en arbre. Ainsi, XOR possède une grande flexibilité pour la sélection des routes comparable à celle de la géométrie en anneau.

#### 2.4.2.6 La géométrie hybride

Les géométries hybrides utilisent une combinaison de formes géométriques. Par exemple Pastry [RD01] combine les géométries de type arbre et anneau afin d'offrir une plus grande efficacité et flexibilité. Les identifiants des pairs sont maintenus à la fois comme les feuilles d'un arbre binaire et comme des points d'un cercle cercle à une dimension. Dans Pastry, la distance entre une paire de nœuds est calculée de deux manières différentes : distance en utilisant l'arbre et distance en utilisant l'anneau. Les pairs ont une grande flexibilité en terme de sélection de leurs voisins. Pour sélectionner leurs voisins, les pairs prennent en compte les propriétés de proximité, c'est à dire qu'ils choisissent les voisins qui sont proches d'eux dans le réseau physique sous-jacent. La sélection des routes est également flexible car, pour acheminer un message, les pairs ont la possibilité de choisir une route qui faire progresser le message dans l'arbre ou une route qui faire progresser le message sur l'anneau.

#### 2.4.2.7 Comparaison des différentes géométries de routages des DHTs

En fonction de leurs géométries de routage, les DHTs disposent de différentes propriétés de routage. Le tableau 2.2 donne une comparaison des géométries de routage des DHTs du point de vue des propriétés de routage suivantes :

- **flexibilité de sélection des routes**. Elle est définie comme le nombre de possibilités que les pairs ont pour router une requête au prochain saut. Plus la flexibilité est élevée, plus le mécanisme de routage est résistant au comportement dynamique des pairs.
- **flexibilité de sélection des voisins**. Elle est définie comme le nombre d'options qu'un pair possède dans la sélection de ses voisins. Plus la flexibilité est élevée, plus les pairs ont

Tableau 2.2 Comparaison des Géometries de Routage des DHTs

| **Géométrie de routage** | **Flexibilité de sélection des voisins** | **Flexibilité de sélection des routes** | **Taille de table de routage** |
|---|---|---|---|
| Arbre | $O(2^{log(n)*log(n)})$ | $O(1)$ | $O(log(n))$ |
| Hypercube | $O(1)$ | $O(log(n)!)$ | $O(log(n))$ |
| Anneau | $O(2^{log(n)*log(n)})$ | $O(log(n)!)$ | $O(log(n))$ |
| Butterfly | $O(1)$ | $O(1)$ | $O(l1$ |
| XOR | $O(2^{log(n)*log(n)})$ | $O(log(n)!)$ | $O(log(n))$ |

plus de possibilités dans le choix de leurs voisins.

– **taille de la table de routage**. C'est le nombre de pairs dont un pair peut maintenir des informations de routage, par exemple leur adresse. Plus la taille de la table de routage est large, beaucoup plus sont les overheads pour maintenir le réseau overlay.

### 2.4.3 Routage dans les systèmes de type super-pair

Les réseaux de type super-pair reposent généralement sur un ensemble de pairs à forte capacités et hautement disponibles appelés super-pairs. Ces super-pairs indexent les données partagées par les pairs qui sont connectés au système. Edutella est l'un des réseaux super-pair les plus connus. Dans Edutella, les super-pairs sont organisés en topologie HyperCup [SSDN02] (voir Figure 2.4.3), afin que les messages puissent être échangés entre deux super-pairs en $O(log(m))$ de sauts de routage, où $m$ est le nombre de super-pairs. Le processus d'adhésion d'un super-pair au réseau se compose de deux parties : prise de la position appropriée dans la topologie HyperCuP et annonce de sa présence à ses voisins. Chaque pair ordinaire rejoint le système en connectant à un super-pair.

Pour permettre un routage efficace des requêtes, deux types d'indices de routage sont maintenues sur chaque super-pair : les indices super-pair / pair (SP/P) et les indices super-pair / super-pair (SP/SP). Les requêtes sont routées vers les super-pairs en utilisant les indices SP/SP, et vers les pairs ordinaires grâce au indices SP/P.

Dans les indices SP/P, chaque super-pair stocke des informations sur les caractéristiques des données qui sont partagées par les pairs qui lui sont connectés. Ces indices sont utilisés pour acheminer les requêtes du super-pair vers les pairs qui lui sont connectés. Pendant l'entrée des pairs dans le système, ils fournissent des informations sur leurs méta-données en publiant une annonce (*advertisement*). Pour indexer les méta-données fournies, Edutella utilise les approches basées sur des schémas qui ont été utilisées avec succès dans le contexte des systèmes d'information à base de médiateur. Pour assurer que les indices soient toujours à jour, les pairs informent leurs super-pairs quand leurs données changent. Quand un pair quitte le réseau, toutes les références à ce pair sont supprimées des indices. Quand un super-pair tombe en panne, les pairs qui lui étaient connectés doivent se connecter à un autre super-pair choisi de manière aléatoire, ces pairs doivent fournir des informations sur leurs méta-données à leurs nouveaux super-pairs.

Le deuxième type d'indices est les indices SP/SP qui sont essentiellement des résumés des indices SP/P. La mise à jour des indices SP/SP est déclenchée après toute modification des indices SP/P. Cette mise à jour s'effectue de la façon suivante. Quand un super-pair change son index SP/P, par exemple en raison d'une entrée ou d'une sortie d'un pair, le super-pair diffuse une annonce de mise à jour du réseau super-pair en utilisant la topologie HyperCuP. Les autres

super-pairs mettent à jour leurs indices SP/SP en conséquence. Bien que cette diffusion n'est pas optimale, il n'est pas non plus trop couteux, parce-que le nombre de super-pairs est beaucoup plus petit que celui de tous les pairs. En outre, si les pairs entrent ou sortent fréquemment, le super-pair peut envoyer périodiquement un résumé d'annonces au lieu d'envoyer des annonces séparées pour chaque entrée ou sortie d'un pair.

Le routage des requêtes dans Edutella est effectué de la manière suivante. Lorsqu'un pair reçoit une requête émise par l'utilisateur, il envoie cette requête à son super-pair. Quand la requête arrive au super-pair, les méta-données utilisées dans la requête sont comparées à des indices SP/P afin de déterminer les pairs locaux (les pairs qui sont connectés à ce super-pair) qui sont en mesure de répondre à cette requête. Si la requête ne peut être satisfaite par les pairs locaux, elle est transmise à d'autres super-pairs à l'aide des indices SP/SP.

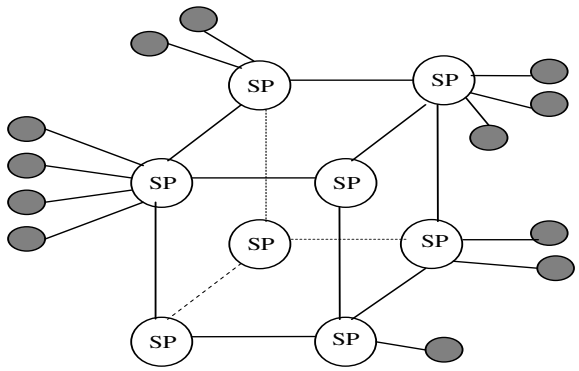



Figure 2.8 **Architecture d'Edutella**

### 2.4.4 Semantic Overlay Networks

Ces dernières années de nombreux travaux ont visé à améliorer la fonction de recherche dans les systèmes non-structurés. L'idée de base est de remplacer le routage aléatoire par un routage guidé par la sémantique. Pour ce faire, le problème est analysé selon les dimensions suivantes :

- quelle sémantique : il s'agit de définir le type d'information à utiliser dans le processus de routage. Cela peut être l'information sur le contenu des pairs (les données/documents stockés), sur l'intérêt du pair (les requêtes déjà émises), sur les utilisateurs (profil d'un utilisateur ou des communautés d'utilisateurs) ;
- quelle représentation de la sémantique : cela va d'une simple information temporelle (les pairs ayant répondus récemment) à des modèles non structurés (simple liste plate de concepts) à des modèles structurés (ontologies) ;
- comment construire la sémantique : le processus de construction peut être manuel (par intervention de l'utilisateur) ou bien automatique (algorithmes d'apprentissage). On peut également trouver des approches mixtes où l'utilisateur intervient via des formes de feed-backs ;
- qu'est ce qui est partagé entre les pairs : a minima les pairs doivent partager des structures de données communes (représentation de la sémantique par exemple), mais cela peut aller au partage de fonctions (algorithmes d'apprentissage par exemple), voire au partage de connaissances (ontologie commune partagée) ;

– comment utiliser la sémantique : généralement la sémantique va être utilisée pour sélectionner le sous ensemble des pairs les plus "pertinents" pour une requête donnée. Cela peut aussi servir à organiser le réseau des pairs (classification des pairs selon leur contenu par exemple) ou bien a modifier les requêtes ;
– comment diffuser la sémantique : la connaissance construite localement sur un pair doit être diffusée aux autres pairs pour qu'ils puissent améliorer leur connaissance du réseau. Cette diffusion peut être globale (à tous les pairs) ou partielle (a quelques uns) et la fraîcheur est également importante (diffusion lors de chaque modification ou bien périodique). Le coût de la diffusion en nombre de messages est par ailleurs crucial.

L'objectif principal des *Semantic Overlays Networks* (SONs) [CGM04b, RPT09] est de regrouper les pairs qui sont sémantiquement similaires et de ce fait de transmettre les requêtes aux SONs dont l'intérêt est proches de celui de la requête, ce qui fait diminuer le coût de routage et augmente l'efficacité du système.

## 2.5 REPRÉSENTATION SYNTHÉTIQUE DES DONNÉES DANS LES SYSTÈMES P2P

Le partage de données massivement distribué dans un environnent P2P est par nature difficile. Étant donnée que la quantité de données et le nombre de pairs augmentent, les techniques de localisation de données deviennent de plus en plus coûteuses. Une approche pratique consiste à s'appuyer sur des résumés compactes des données au lieu des données brutes, dont l'accès est très couteux dans un système P2P à très grande échelle. Dans cette section nous présentons deux techniques proposées pour la représentation synthétique des données dans un système P2P.

Dans [Hay, HRVM07], les auteurs présentent PEERSUM, un service de résumés de données dans les systèmes P2P proposé dans le contexte d'APPA. Dans cette approche, le processus de résumé est intégré au DBMS de chaque pair afin de leur permettre de construire leur résumé local. Cette approche est basée sur SaintEtiQ [RM02], une technique linguistique pour la compression sémantique d'une base de données relationnelle. SaintEtiQ traite de la caractérisation intentionnelle des groupes de tuples en utilisant des variables linguistiques [Zad75]. Cette compression sémantique respecte les schémas initiaux (d'origines) des ensembles de données et peut être directement interrogée ou utilisée comme ensemble de données alternatif pour toute opération nécessitant une vue de la base de données (par exemple interrogation, fouille de données, navigation, etc.). L'approche PEERSUM consiste à organiser le réseau P2P en domaines. Chaque pair maintient un résumé local sur ses propres données, et les pairs qui sont dans un domaine construisent un résumé global des données qu'ils partagent dans le domaine (voir Figure 2.5). L'ensemble des résumés globaux matérialisés et les liens entre les nœuds de leurs domaines, fournissent un résumé virtuel complet, qui décrit idéalement le contenu de toutes les données partagés dans le réseau. Dans PEERSUM, les résumés de données peuvent être interroger ou utiliser directement pour répondre approximativement à une requête sans une exploration des données d'origine. De plus, ces résumés tout comme des indexes sémantiques, peuvent permettre de localiser les pairs pertinents et ceci en se basant sur leurs contenus.

Dans [VLCV09a], Les auteurs proposent une technique de représentation synthétique permettant de décrire un ensemble de documents, le contenu d'un pair, mais aussi un groupe de pairs de manière optimiste. Cette représentation synthétique est obtenue à partir d'une fonction d'agrégation qui possède des propriétés d'incrémentalité et de composabilité permettant ainsi une

gestion simplifiée des regroupements (par exemple des contenus des pairs) dans un système distribué. La fonction d'agrégation possède aussi la propriété d'optimisme. Cette propriété présente l'avantage qu'aucune information ne peut avoir une mesure de pertinence supérieure à celle obtenue par une agrégation dans laquelle elle apparaît. Les auteurs ont montré l'intérêt que peut avoir cette propriété d'optimisme dans la recherche d'une information dans un système distribué, en permettant en particulier d'éviter de visiter certains pairs. Cette propriété peut aussi être utilisée localement par un pair pour éviter de consulter systématiquement l'ensemble de sa collection d'information.

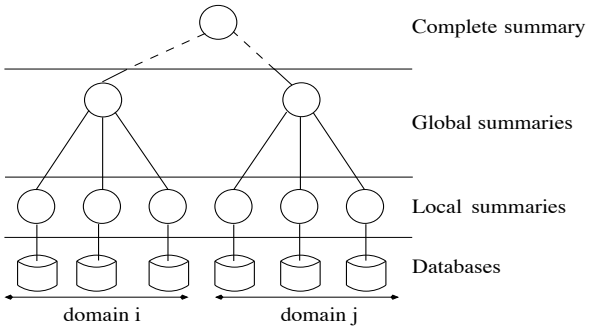



Figure 2.9 **Architecture d'un modèle de résumé de données dans PEERSUM**

## 2.6 TRAITEMENT DES REQUÊTES COMPLEXES

Dans la section 2.4 nous avons présenté des techniques de routage des requêtes aux pairs pertinents c.-à-d. les pairs qui possèdent des données intéressantes à ces requêtes. Ces techniques permettent aux systèmes P2P de fournir un bon support des requêtes exactes (*exact-match queries*). Cependant, le support des requêtes plus complexes ou des requêtes basées sur des schémas par les systèmes P2P n'est pas du tout aisé et ceci est encore plus difficile dans les DHTs. Comme exemples de requêtes complexes, nous pouvons citer : les requêtes top-$k$, les requêtes skyline, les requêtes de jointures, les requêtes de type intervalles et les requêtes multi-attributs. Dans cette section, nous présentons les approches qui ont été proposées pour le traitement de ces différents types de requêtes complexes dans les systèmes P2P.

### 2.6.1 Les requêtes top-$k$

Les requêtes top-$k$ ont attiré beaucoup d'attention dans plusieurs domaines de la technologie de l'information tels les réseaux et les systèmes de surveillance [BO03, KOT04, CW04], la recherche d'information [MTW05, BNST05, PZSD96], les bases de données multimédias [CGM04a, NR99, dVMNK02], l'analyse des données spatiales [BBK01, CP02, HS03], etc. Grâce à une requête top-$k$, l'utilisateur peut spécifier le nombre $k$ de réponses les plus pertinentes que le système doit lui retourner. Le degré de pertinence (score) des réponses par rapport à la requête est déterminé par une fonction de score. Les requêtes top-$k$ sont très utiles pour la gestion des données dans les systèmes P2P [APV06], car elles offrent deux avantages principaux. Premièrement, elle permet d'éviter de submerger l'utilisateur par un grand de nombre en lui

retournant que le nombre de réponses les plus pertinentes qu'il souhaite. Deuxièmement, elle permet une réduction significative du trafic réseau.

Pour illustrer l'importance des requêtes top-$k$ pour la gestion des données P2P, considérons un système P2P où des médecins veulent partager certaines données (accès restreint) des patients pour une étude épidémiologique. Supposons donc que tous les médecins utilisent la même description sous forme relationnelle de l'entité *Patient*. Un médecin peut donc vouloir soumettre la requête suivante pour obtenir les 10 meilleurs résultats classés par une fonction de score sur la taille (*height*) et le poids (*weight*) des patients :

```
SELECT * FROM Patient P WHERE (P.disease = "diabetes") AND (P.height < 170) AND
(P.weight > 70) ORDER BY scoring-function(height, weight) STOP AFTER
```

La fonction de score spécifie à quel point chaque donnée correspond aux conditions de la requête. Par exemple, dans la requête ci-dessus, la fonction de score pourrait être $(weight - (height - 100))$ qui calcule la quantité de sur-poids d'un patient.

L'exécution des requêtes top-$k$ dans les systèmes P2P à grande échelle est difficile. Dans cette section, nous examinons d'abord les techniques les plus efficaces pour le traitement des requêtes top-$k$ dans les systèmes distribuées. Ensuite, nous présentons les techniques proposées pour les systèmes P2P.

#### 2.6.1.1 Les requêtes top-$k$ dans les Systèmes distribués

Les approches les plus efficaces pour le traitement des requêtes top-$k$ dans les systèmes distribués sont basées sur le *Threshold Algorithm* (TA) [FLN01, GBK00, NR99]. Ces approches supposent une distribution verticale des données. L'approche TA est applicable pour les requêtes lorsque la fonction de score est monotone, c'est à dire que toute augmentation de la valeur d'entrée de la fonction de score ne diminue pas la valeur de la sortie. La plupart des fonctions d'agrégations à savoir Min, Max, Moyenne, sont monotones. Soit $m$ listes contenant chacun $n$ éléments (données) tels que chaque élément à un score locale dans chaque liste et que les listes sont triées en fonction des scores de leurs éléments. En supposant que le score global d'un élément est calculé sur la base des scores locaux de cet élément dans toutes les listes en utilisant une fonction de score. TA trouve les $k$ éléments ayant les scores globaux les plus élevés de la façon suivant. TA parcours de façon descendante les listes triées en parallèle, une position à un moment, et récupère pour chaque élément, ses scores dans toutes listes, et calcul son score global. Ce processus se poursuit jusqu'à ce que soit trouvé les $k$ éléments dont les scores globaux sont supérieurs à un seuil qui est le score global d'une donnée virtuelle dont les scores locaux sont les scores locaux se trouvant à la position courante dans les listes.

L'algorithme TA peut être appliqué pour le traitement des requêtes top-$k$ sur des données relationnelles en considérant chaque colonne d'une table comme une liste (distribution verticale des données), ainsi en triant les colonnes en fonction de leurs valeurs, et en utilisant algorithm TA pour trouver les $k$ tuples dont les scores globaux sont les plus élevés.

Plusieurs extensions de l'algorithme TA ont été proposées pour le traitement des requêtes top-$k$ dans un environnement distribué. Du fait que ces approches sont des extensions de TA, elles utilisent donc comme hypothèse une distribution verticale des données. Parmi les approches proposées nous pouvons citer : KLEE [MTW05] et *Three Phase Uniform Threshold* (TPUT) [CW04].

Le framework KLEE [MTW05] est une extension de l'algorithme TA qui vise à réduire le coût de communication du traitement des requêtes top-$k$ dans les systèmes distribués. cet algorithme offre un meilleur gain de performance au frais d'une légère réduction de la complétude des réponses. Dans KLEE, les $m$ listes triées sont reparties sur $m$ nœuds. Chaque nœud divise sa liste en $c$ cellules et maintient des informations statistiques décrivant ces cellules, par exemple le plus petit, le plus grand, la moyenne et la fréquence des scores qui appartiennent à la cellule . Pour chaque cellule, KLEE utilise des filtres de Bloom KLEE [Blo70] pour représenter de manière compacte, l'ensemble des données dont les scores locaux appartiennent à cette cellule. Cette information sur les cellules avec les filtres de Bloom permet de réduire le nombre de scores locaux, qui doit être communiqué sur le réseau, réduisant ainsi le coût de la communication.

L'algorithme TPUT [CW04] est une extension de l'algorithme de TA qui exécute les requêtes top-$k$ en trois *round trips* (allers-retours). TPUT suppose que chaque liste est détenue par un nœud, que nous appelons détenteur de liste. TPUT s'effectue en trois phases comme suit.

1. Chaque détenteur de liste envoie à l'initiateur de la requête ses $k$ meilleurs données, c.-à-d. les $k$ données ayant les scores locaux les plus élevés dans sa liste. Soit $f$ la fonction de score, $d$ une donnée reçue par l'initiateur de la requête, et $s_i(d)$ le score local de $d$ dans la liste $i$, alors la somme partielle de $d$ est définie comme suit $p_{sum}(d) = s'_1(d) + s'_2(d) + \cdots + s'_m(d)$ où $s'_i(d) = s_i(d)$ si le détenteur de la liste $i$ a envoyé $d$ à l'initiateur de la requête, sinon $s'_i(d) = 0$. L'initiateur de la requête calcule les sommes partielles de toutes données reçues et identifie les éléments qui ont les $k$ sommes partielles les plus élevées. La somme partielle du $k$-ème donnée est appelée *phase-1 bottom* et est notée $\lambda_1$.
2. L'initiateur de la requête envoie une valeur de seuil $\tau = \frac{\lambda_1}{m}$ à chaque détenteur de liste. Ensuite, chaque détenteur de liste renvoie à l'initiateur de la requête toutes ses données dont les scores locaux ne sont pas inférieur à $\tau$. L'intuition derrière ce mécanisme est que, si à cette phase, si une donnée n'a pas été retournée par aucun nœud, c'est que son score doit être inférieur à $\tau$, donc il ne peut être dans le top-$k$. Soit alors $D$ l'ensemble des données reçues de la part des détenteurs de listes, l'initiateur de la requête calcule la nouvelle somme partielle des données contenues dans l'ensemble $D$, et identifie les éléments qui ont les $k$ sommes partielles les plus élevées. La somme partielle du $k$-ème donnée est appelée *phase-2 bottom* et est notée $\lambda_2$. Soit $u_{score}(d) = u_1(d) + u_2(d) + \cdots + u_m(d)$ le score limite supérieure d'une donnée $d$ où $u_i(d) = s_i(d)$ si $d$ a été déjà reçue de la part du détenteur de la liste $i$, sinon $u_i(d) = \tau$. Pour chaque donnée $d \in D$, si le score limite supérieure de $d$ est inférieure $\lambda_2$ alors $d$ est retirée de $D$. Les données restantes dans l'ensemble $D$ sont appelées des données candidates au top-$k$.
3. L'initiateur de la requête envoie l'ensemble des données candidates au top-$k$ à chaque détenteur de liste qui lui renvoie les scores de ces données. Ensuite, l'initiateur de la requête calcule les scores globaux, extrait les $k$ données ayant les scores les plus élevés, et retourne le résultat à l'utilisateur.

Notons que quand le nombre $m$ de listes est grand, le temps de réponse de TPUT est bien meilleur que celui de l'algorithme TA de base comme ça été montré dans [CW04].

#### 2.6.1.2 Les requêtes top-$k$ dans les Systèmes P2P

Dans cette section, nous présentons les approches proposées pour le traitement des requêtes top-$k$ dans chacune des trois classes de systèmes P2P que nous avons présenté dans la section 2.1.

(i) **Top-$k$ dans les systèmes non-structurés**.
Les approches de traitement des requêtes top-$k$ dans les systèmes P2P non-structurés s'appuie sur l'hypothèse d'une distribution horizontale des données. Dans ce contexte, une approche simple pour le traitement des requêtes top-$k$ dans les systèmes non-structurés consiste à router la requête à tous les pairs, récupérer toutes les réponses disponibles, ensuite calculer les scores de ces réponses en utilisant la fonction de score, et retourner à l'utilisateur les $k$ réponses ayant les scores les plus élevés. Cependant, cette approche n'est pas efficace en termes de temps de réponse et de coût de communication comme montré dans [AMPV06c]. Les principales approches de traitement des requêtes top-$k$ dans les systèmes non-structurés sont : l'approche présentée dans PlanetP [CAPMN03], l'approche FD (*Fully Distributed*) [APV06], et BRANCA [ZTZ07].
PlanetP [CAPMN03] est un système non-structuré qui supporte les requêtes top-$k$. PlanetP utilise un protocole de propagation par rumeur (*gossip*) pour répliquer un index global et compacte des résumés des contenus (correspondances entre terme/pair) qui sont partagés par chaque pair. Le traitement d'une requête top-$k$ se déroule comme suit. Étant donnée une requête top-$k$ notée $q$, l'initiateur de la requête évalue les scores de pertinence des pairs par rapport à $q$ en utilisant le résumé global compacte. Ensuite, l'initiateur de la requête contact un à un chacun des pairs dans l'ordre décroissant de leurs scores de pertinence et leur demande de lui retourner l'ensemble des noms de leurs documents ayant des scores les plus élevés accompagnés de leurs scores. Bien que cette approche offre une bonne performance dans les systèmes de tailles moyennes, elle n'est pas applicable pour un système P2P de grande taille, car maintenir à jour un index global répliqué est un problème majeur qui limite le passage à l'échelle du système.
Dans [APV06], Akbarinia et al. présentent FD une famille d'algorithmes totalement distribués pour le traitement des requêtes top-$k$ dans des réseaux non-structurés. L'objectif principal de FD est de réduire le trafic réseau dans les systèmes P2P non-structurés. L'algorithme de base de FD se déroule de la façon suivante. L'algorithme débute sur le pair $p_i$ : le pair initiateur de la requête $q$. Celui-ci commence par :

1. Initialiser la valeur du *ttl*.
2. Donner un identifiant unique *qid* à la requête $q$. *qid* est crée à partir de l'identifiant du pair $p_i$ et d'un compteur local (au pair à $p_i$). cet identifiant permet aux paires de distinguer les nouvelles requêtes de celles qu'ils ont déjà reçu.

Le pair $p_i$, exécute les quatre phase suivantes : transmission de la requête, exécution locale de la requête, fusion et renvoie des résultats et récupération des données.

**Transmission de la requête** Lorsqu'un pair $p$ reçoit un message incluant la requête $q$, les actions suivantes sont effectuées :

1. si la requête a déjà été reçue, le message est ignoré ; sinon l'adresse de l'expéditeur est conservée : l'expéditeur devient le père de $p$.
2. la valeur du *ttl* est diminuée de 1 : si elle reste strictement positive, un nouveau message est créé et envoyé à tous les voisins de $p$ (sauf à son père). Le message créé contient la requête $q$, son identifiant *qid*, la nouvelle valeur du *ttl* et l'adresse de l'initiateur de la requête $p_i$.

**Exécution locale de la requête** Après avoir procédé à la transmission de la requête, chaque pair exécute $q$ localement. L'exécution de la requête se traduit par le parcours de toutes les données du pair et par l'attribution d'une valeur de pertinence à chaque donnée. La pertinence d'une donnée $d$ par rapport à une requête $q$ est mesurée grâce à une fonction de score (*scoring function*). Une fois que les $k$ meilleurs données locales sont sélectionnées, le pair doit attendre les réponses de tous ses fils avant de commencer la phase suivante.

**Fusion et renvoi des résultats** Une fois qu'un pair reçoit toutes les réponses de ses fils, il procède à la fusion de ces réponses avec ses réponses locales, récupère ensuite les $k$ meilleurs qu'il retourne à son père. Pour réduire le trafic réseau, les pairs ne remontent pas directement les réponses mais des *scores-lists*. Un score-list est un ensemble de couples $(p', sc)$ où $p'$ est l'adresse du pair contenant la donnée et $sc$ le score de la donnée.

**Récupération des données** Une fois que l'initiateur de la requête reçoit tous les scores-lists de tous ses voisins, et récupérer les $k$ meilleurs éléments, ce dernier procède à la récupération des $k$ meilleurs données de la façon suivante :

1. détermination du nombre de fois que chaque pair $p'$ apparaît dans le scores-list final. Soit alors $m$ ce nombre.
2. demande à chaque pair $p'$ de renvoyer ses $m$ meilleurs données.

Le pair obtient ainsi les $k$ données les plus pertinentes, dans un rayon $R$ (où $R$ est la valeur du *ttl* fourni par le pair initiateur).
Bien que l'approche FD permet de réduire considérablement le trafic réseau, son principal inconvénient est que le temps d'attente de l'utilisateur pour obtenir les $k$ meilleurs réponses est très long. Ceci est dû au fait que tous les pairs doivent traitées la requête avant que le résultat de la requête top-$k$ ne soit retourner à l'utilisateur. Le temps d'attente de l'utilisateur est donc dépendant du pair le plus lent du système.
Dans [ZTZ07] les auteurs présentent une technique de traitement des requêtes top-$k$ appelée BRANCA. Dans BRANCA, chaque pair stocke dans un cache les réponses reçues de ses voisins lors du traitement des requêtes top-$k$ précédentes. Étant donné une nouvelle requête, le cache permet au pair de transmettre la requête uniquement à la partie du réseau susceptible de contenir des données pouvant être dans le top-$k$ final. Cette approche permet de fournir rapidement des résultats à une requête si cette requête peut être résolue à partir des informations stockées dans le cache. Si cette requête ne peut pas être résolue totalement par le cache, l'exécution de la requête est effectuée comme dans le cas de FD. Ce qui signifie quand une requête ne peut pas être résolue via le cache, le temps d'attente de l'utilisateur peut être très long (car celui ci va dépendre du pair le plus lent). Notons aussi que le fait que les pairs stockent résultats des requêtes passées dans des caches pour répondre à des requêtes ultérieures fait que la performance de cette approche dépend de la distribution des requêtes, de la taille des caches et de la technique de gestion de ces caches. En plus il est à noter que s'appuyer des caches pour répondre à une requête top-$k$ dans un environnement à grande échelle et très dynamique peux avoir un impact négative sur la précision des résultats retournés à l'utilisateur.

(ii) **Top-$k$ dans les systèmes super-pair**.
Les principales approches de traitement des requêtes top-$k$ dans les systèmes super-pair

s'appuie sur l'hypothèse d'une distribution horizontale des données.
Dans [BNST05], les auteurs proposent un algorithme de traitement de requête top-$k$ pour Edutella, un réseau de super-pair. L'approche proposée par les auteurs s'appuie sur l'hypothèse d'une distribution horizontale des données. Dans Edutella, un petit pourcentage de nœuds sont des super-pairs et sont censés être hautement disponibles avec de très bonne capacité de calcul. Les super-pairs sont responsables du traitement de la requête top-$k$ et les autres pairs exécutent uniquement les requêtes localement et assignent des scores à leurs ressources. Le traitement d'une requête top-$k$ s'effectue de la façon suivante. Étant donnée une requête top-$k$ notée $q$, l'initiateur de la requête envoie $q$ à son super-pair, et son super-pair envoie $q$ aux autres super-pairs. Les super-pairs routent ensuite la requête $q$ vers les pairs pertinents qui leurs sont connectés. Chaque pair qui possède des données pertinentes à $q$ assigne des scores à ces données et envoie la donnée ayant le score maximal à son super-pair. Chaque super-pair choisi la donnée ayant le score maximal parmi toutes les données reçues. Pour déterminer la deuxième donnée la plus pertinente, le super-pair demande au pair qui lui avait retournée la donnée ayant le plus grand score de lui retourner sa donnée ayant le deuxième plus grand score. Ensuite, le super-pair sélectionne la donnée ayant le plus grand score parmi les données reçues précédemment de ses pairs et de la nouvelle donnée reçue (du pair ayant fourni la donnée ayant le plus grand score). Ce mécanisme est effectué par le super pair jusqu'à ce que les $k$ données ayant les scores les plus élevés soient sélectionnées. Enfin, les super-pairs envoient leurs $k$ meilleurs données au super-pair de l'initiateur de la requête qui extrait $k$ meilleurs données parmi l'ensemble de ces données et les envoie à l'initiateur de la requête. L'inconvénient de cette approche est que le temps d'attente pour obtenir tous les $k$ meilleurs réponses peut être très long.
Dans [VDNV08], les auteurs proposent SPEERTO, un framework qui supporte le traitement des requêtes top-$k$ dans les réseaux super-pair basé sur l'utilisation de l'opérateur de skyline [BKS01]. SPEERTO s'appuie sur une distribution horizontale des données. Dans SPEERTO, pour un $K$ désignant la limite supérieure du nombre de résultats souhaité par l'utilisateur pour les requêtes top-$k$ ($k \leq K$), chaque pair calcule son $K$-skyband comme une étape de pré-traitement. Chaque super-pair maintient et agrège l'ensemble des $K$-skyband de ses pairs. Une fois que les $K$-skyband sont agrégés, ils sont échangés entre tous les autres super-pairs. Cela permet à tous les super-pairs d'avoir une connaissance globale des données du réseau. Quand un super-pair reçoit une requête top-$k$, le super-pair se base sur cette connaissance globale pour router uniquement la requête vers les pairs ou les super-pairs dont les résultats seront dans le top-$k$. L'inconvénient de cette approche est l'utilisation d'une connaissance globale qui peut limiter le passage à l'échelle du système dans un environnement très dynamique. En effet chaque entrée ou sortie ou chaque mis à jour des données d'un pair nécessite un recalcule des $K$-skyband sur les super-pairs.

(iii) **Top-$k$ dans les DHTs**.
La fonctionnalité principale d'une DHT est de mapper un ensemble de clés aux pairs du système P2P et de localiser de manière efficace le responsable de n'importe clé se trouvant dans cet ensemble. Cela permet un support efficace (et pouvant passer à l'échelle) des requêtes de type exact match. Cependant, il est difficile de supporter les requêtes top-$k$ dans les DHTs. Une solution simple de traitement d'une requête top-$k$ dans une DHT consiste à récupérer tous les tuples des relations impliquées dans la requête, calculer le

score de chaque tuple récupéré, et enfin retourner les $k$ tuples dont les scores sont les plus élevés. Cependant, cette solution ne peut pas passer à l'échelle quand le nombre de tuples stockés est très grand. Une autre solution consiste à stocker tous les tuples de chaque relation en utilisant la même clé (par exemple le nom de la relation), de telle sorte que tous les tuples soient stockés sur le même pair. Ensuite, le traitement d'une requête top-$k$ peut alors être effectué sur ce pair central en utilisant les algorithmes de traitement de requête top-$k$ proposés dans les systèmes centralisés. Cependant, le pair central devient un goulot d'étranglement et le point de défaillance du système.
Dans le cadre d'APPA (un réseau indépendant du type du système P2P) [AMPV06b], Akbarenia et al. ont proposé une solution efficace pour le traitement des requêtes top-$k$ dans les DHTs [AMPV06c]. La solution proposée est basée sur l'algorithme TA [FLN01, GBK00, NR99]. cette solution proposée peut passer à l'échelle pour un grand nombre de pairs et évite tout stockage centralisé des données. La solution est basée sur un mécanisme de stockage de données qui stocke les données partagées dans la DHT de manière totalement distribuée. Nous décrivons cette solution ci-dessous.

**Mécanisme de stockage de données** Dans le mécanisme de stockage proposé dans APPA, les pairs stockent leurs données relationnelles dans la DHT grâce à deux méthodes complémentaires : la méthode de stockage de tuples et la méthode de stockage de paires attribut-valeur. Avec la méthode de stockage de tuples, chaque tuple d'une relation est stocké dans la DHT en utilisant l'identifiant du tuple (par exemple, sa clé primaire) comme clé de stockage. Ceci permet aux pairs de rechercher un tuple à partir de son identifiant.
Le stockage attribut-valeur stocke individuellement dans la DHT les attributs qui peuvent apparaître dans la fonction de score des requêtes top-$k$. Comme par exemple pour les index secondaires dans les bases de données, il permet de rechercher les tuples en utilisant leurs valeurs d'attribut. La méthode de stockage attribut-valeur a deux propriétés importantes : 1) suite à la recherche d'une valeur d'attribut, les pairs peuvent rechercher facilement le tuple correspondant ; 2) les valeurs d'attribut qui sont relativement "proches" sont stockées sur le même pair. Pour offrir la première propriété, la clé, qui est utilisée pour stocker le tuple entier, est stockée avec la valeur d'attribut dans la DHT. La deuxième propriété est fournie en employant le concept de partitionnement de domaine comme suit. Soit $a$ un attribut, et $D_a$ le domaine des valeurs de $a$. $D_a$ est divisé en $n$ sous-domaines non vides $d_1, d_2, \cdots, d_n$ tels que leur union est égale à $D_a$, l'intersection de deux sous-domaines différents soit vide, et pour chaque $v_1 \in d_i$ et $v_2 \in d_j$, si $i < j$ alors nous avons $v_1 < v_2$. Pour faire le stockage attribut-valeur, la fonction de hachage est appliquée sur le sous-domaine de la valeur d'attribut. Ainsi, pour les valeurs qui se trouvent dans le même sous-domaine, la clé de stockage est identique ; elles sont donc stockées sur le même pair. Pour éviter les distributions de données biaisées qui peuvent conduire à un stockage déséquilibré entre des pairs différents, le partitionnement de domaine est fait de telle façon que les valeurs d'attributs soient uniformément distribuées dans les sous-domaines. Cette technique emploie des informations basées sur des histogrammes qui décrivent la distribution des valeurs de l'attribut.

**Algorithme de Traitement de Requêtes Top-$k$** Soit $q$ une requête top-$k$, $f$ est une fonction de score, et $p_{int}$ le pair initiateur de la requête $q$. En supposant que les attributs de scoring sont ceux qui sont passés à la fonction de score comme arguments. L'algorithme de

base de traitement des requêtes top-$k$ proposés par les auteurs, appelé DHTop1, commence sur le pair $p_{int}$ et se déroule en deux phases : (1) préparation des listes de sous- domaines ; (2) recherche des valeurs d'attribut et de leurs tuples jusqu'à trouver les $k$ meilleurs tuples.

1. Pour chaque attribut de scoring, $p_{int}$ prépare une liste de sous-domaines et les trie selon leur impact positif sur la fonction de scoring, c.-à-d. les sous-domaines pour lesquels la valeur de la fonction de scoring est plus élevée sont au début de la liste. Pour chaque liste, $p_{int}$ enlève de la liste les sous-domaines dont aucun membre ne peut satisfaire les conditions de $q$. Par exemple, s'il y a une condition qui impose que l'attribut doit être égal à une constante, par ex. $\alpha = u$, $p_{int}$ enlève de la liste tous ses sous-domaines, excepté le sous-domaine auquel la constante appartient. Dénotons par $CDL_\alpha$ la liste préparée pour l'attribut de scoring $\alpha$ dans cette phase.
2. Pour chaque attribut de scoring $\alpha$, $p_{int}$ procède comme suit (en parallèle). Il envoie $q$ et $\alpha$ au pair $p$ qui est responsable pour maintenir les valeurs du premier sous-domaine de $CDL_\alpha$, et lui demande de renvoyer les valeurs de $\alpha$ qui sont stockées. Les valeurs sont renvoyées à $p_{int}$ par ordre de leur impact positif sur la fonction de scoring. Après la réception de chaque valeur, $p_{int}$ recherche le tuple correspondant, calcule le score du tuple, et le garde si son score est plus grand que les $k$ meilleurs scores déjà calculés. Ce processus continue jusqu'à avoir $k$ tuples qui ont des scores plus élevés que le seuil qui est calculé basé sur les dernières valeurs récupérées pour chaque attribut de scoring. Si les valeurs que $p$ renvoie a $p_{int}$ ne sont pas suffisantes pour déterminer les $k$ meilleurs tuples, $p_{int}$ envoie $q$ et $\alpha$ au pair qui est responsable du deuxième sous-domaine inclus dans $CDL_\alpha$. Le seuil est calculé comme suit. Supposons que $\alpha_1, \alpha_2, \cdots, \alpha_m$ sont les attributs de scoring. Supposons que $v_1, v_2, ..., v_m$ sont les dernières valeurs récupérées pour les attributs $\alpha_1, \alpha_2, \cdots, \alpha_m$ respectivement. Le seuil est calculé par $\gamma = f(v_1, v_2, ..., v_m)$. Une caractéristique importante de $\gamma$ est de diminuer après la réception de chaque nouvelle valeur d'attribut. Donc, après la récupération d'un certain nombre de valeurs d'attributs et de leurs tuples, le seuil devient inférieur au score de tous les $k$ meilleurs tuples déjà récupérés.

### 2.6.2 Les requêtes skyline

Les requêtes top-$k$ sont parfois difficiles à définir par l'utilisateur, spécialement si plusieurs attributs doivent être optimisés (difficulté de définition de la fonction de score). Les requêtes skyline [BKS01] apparaissent donc une solution pour résoudre ce problème. Les requêtes skyline aident l'utilisateur à prendre des décisions intelligentes sur des données qui présentent plusieurs critères contradictoires. Le skyline $S$ d'un ensemble de points $d$-dimensionnel est l'ensemble des points qui ne sont pas dominés par tout autre point de $S$. Un point $p$ est dominée par un autre point $q$ si et seulement si $q$ n'est pas pire que $p$ sur aucune des dimensions et que $q$ est meilleur que $p$ sur au moins une dimension.
Le traitement des requêtes skyline a récemment suscité une attention considérable à la fois dans les systèmes centralisées [BKS01] et distribués [BGZ04, ZTZ09]. L'un des premiers algorithmes dans les distribués, proposé par Balke et al. [BGZ04], se concentre sur le traitement des requêtes skyline impliquant de multiples sources de données distribués, avec chaque source de stockant uniquement un sous-ensemble d'attributs (distribution verticale des données). Plus tard, la plupart des travaux se sont concentrés sur les environnements fortement distribués et P2P, en supposant que toutes les sources de données stockent les

mêmes attributs (distribution horizontale des données). Dans les environnement fortement distribués, dans [CLX+08] les auteurs proposent une technique appelé PaDSkyline pour traitement des requêtes skyline. Dans cette approche, il existe un coordonnateur qui peut communiquer directement avec tous les pairs (serveurs). L'approche se base aussi sur l'utilisation d'un mécanisme appelé MBRs (*Minimum Bounding Regions*) pour résumer les données stockées sur chaque serveur. Le traitement d'une requête skyline s'effectue en deux étapes. Dans la première étape, les MBRs de tous les serveurs sont collectés et affectés à des groupes incomparables, qui peuvent être interrogés en parallèle, tandis que des plans spécifiques sont utilisés au sein de chaque groupe. Par la suite, les serveurs sont interrogés et les réponses sont retournés au coordonnateur. Récemment, dans [ZTZ09], un algorithme de skyline distribué basé sur des feeddbacks (*FDS : Feedback-based distributed skyline*) a été proposé. Malgré que cet algorithme soit efficace en termes de consommation de bande passante, il nécessite cependant plusieurs allers-retours pour calculer le skyline, et donc susceptible d'engendrer un temps de réponse très long.

Dans le contexte P2P, la majorité des approches de traitement de requêtes skyline se basent sur une distribution horizontale des données. Les approches qui se basent sur la distribution horizontale des données peuvent être classées en deux catégories principales. Dans la première catégorie, les méthodes proposées supposent un partitionnement de l'espace entre les pairs, ainsi chaque pair est responsable d'une partition disjointe de l'espace de données. Pour atteindre cet objectif, un P2P structuré plus précisément une DHT est utilisée. Le système contrôle l'emplacement de chaque point de données et divise les données d'une manière que le système puisse visiter premièrement les pairs ayant une grande probabilité d'avoir les points du skyline. Par exemple dans [WZF+06], les auteurs proposent la première approche de traitement des requêtes skyline dans les systèmes P2P de type CAN [RFH+]. Les auteurs présentent une méthode de partitionnement récursive de régions et d'encodage dynamique de régions pour faire respecter un ordre partiel sur les zones de la CAN. Ainsi tous les pairs participants au système, peuvent être correctement en pipeline pour l'exécution de la requête. Au cours de la la propagation de la requête, les espaces de données dont les données ne peuvent pas faire partie du skyline sont dynamiquement éliminés et les résultats de la requête sont générés progressivement. Par conséquent, les utilisateurs n'ont pas besoin d'attendre la fin d'une requête pour pouvoir obtenir des résultats partiels, ce qui réduit considérablement le temps d'attente. Cependant, ce travail s'est focalisé uniquement sur les requêtes skyline à contraintes [PTFS03] où les utilisateurs ne sont intéressés que par les points de l'ensemble skyline qui doivent vérifier de multiples contraintes (qui sont souvent des contraintes dures).

Dans la seconde catégorie d'approches, chaque pair stocke de manière autonome ses propres données. Par exemple dans SKYPEER [VDKV07], chaque super-pair calcule et stocke le skyline étendu des pairs qui lui sont connectés. Ensuite, quand un super-pair traite une requête skyline, la requête est forwardée aux super-pairs voisins. Dans cette approche, les requêtes skyline sont traitées localement par les super-pairs en se basant sur les skylines étendus, suivi ensuite d'une fusion des réponses via le réseau. Fotiadou et al. [FP08] proposent BITPEER une approche de traitement des requêtes skyline dans les réseaux super-pair qui utilise une représentation bitmap, afin d'améliorer les performances du traitement des requêtes. Dans [HLS06], les auteurs utilisent des résumés de données distribuées (QTree) des données stockées par les pairs dans le mécanisme de traitement des requêtes skyline. L'approche proposée par les auteurs s'appuie sur un réseau non-structuré. Durant

la phase de traitement d'une requête skyline, le QTree est utilisé comme mécanisme de routage pour déterminer les pairs, qui ont besoin de traiter cette requête, afin de trouver les points du skyline. Bien que cette approche fournit des garanties quant à la complétude des réponses, elle supporte aussi des réponses approximatives afin de réduire le coût de traitement des requêtes.
Les requêtes skyline sont très intéressantes dans un cadre de gestion de données P2P. Cependant, les approches proposées dans les systèmes P2P présentent deux inconvénients principaux. Le premier inconvénient est que le temps d'attente de l'utilisateur pour obtenir le résultat de la requête skyline peut être très long. Le deuxième inconvénient est que le nombre des réponses (cardinalité des réponses) retourné à l'utilisateur peut devenir très important quand le nombre d'attributs à optimiser augmente. Ce qui peut donc entrainer une très forte charge sur le réseau et sur le pair initiateur de la requête. Pour pallier au problème de la cardinalité des réponses des requêtes skyline une nouvelle forme de requête skyline a été proposée très récemment : le *top-k skyline* ou le *top-k dominating queries* [YM07, GV09]. L'idée est retournée uniquement à l'utilisateur les $k$ points les plus importantes de l'ensemble skyline. Ces $k$ points sont déterminer grâce à leur puissance de domination (domination power). La puissance de domination d'un point $p$ dans un ensemble $S$ est le nombre de points que $p$ domine dans $S$.

### 2.6.3 Autres types de requêtes complexes

Dans cette section, nous présentons trois autres types de requêtes complexes : les requêtes de type intervalle, les requêtes multi-attributs, et les requêtes de jointure. Le traitement de ces types de requêtes dans les réseaux non-structurés et super-pair ne soulève aucune difficulté. En effet, les requêtes de type intervalle et les requêtes multi-attributs peuvent être traitées comme de simples requêtes de sélection, et les requêtes de jointures peuvent être traitées comme dans les systèmes distribués. Cependant, le traitement de ces requêtes dans les DHTs est beaucoup plus difficile. Pour cela dans la suite de cette section, nous nous focalisons sur le traitement de ces trois types de requêtes dans les DHTs.

#### 2.6.3.1 Les requêtes de type intervalle

Les requêtes de type intervalle sont difficiles à traiter dans les DHTs. Une requête de type intervalle permet à l'utilisateur de récupérer des données dont les valeurs des attributs se situe dans un intervalle de valeurs spécifique. Un exemple de requête type intervalle est : `les patients dont le poids est compris entre 50 et 60 kilogrammes`. Le problème principal pour la mise en œuvre des requêtes de type intervalle dans les DHTs est que la fonction de hachage, utilisée par une DHT pour stocker les données sur des pairs, ne permet pas de maintenir la proximité des données. En effet, le *hash* de deux données très proches peuvent être des nombres éloignés. Il y a eu plusieurs propositions pour le support des requêtes de type intervalle dans les DHTs. Dans [GAA03], les auteurs s'appuient sur un hachage sensible à la localité pour permettre que, des valeurs proches puissent être stockées sur le même pair avec une forte probabilité. Ils proposent une famille de fonctions de hachage sensible à la localité, appelée *min-wise independent permutations*. Les résultats des simulations ont montré que cette solution offre une bonne performance. Cependant pour les réseaux de grandes tailles, il se pose le problème de la non équilibrage de charge.

SkipNet [HJS+03] est une DHT lexicographique qui préserve l'ordre, cela permet donc aux données ayant des valeurs similaires d'être placées sur les pairs contigües. Elle utilise les noms au lieu des identifiants hachés pour ordonner les pairs dans le réseau overlay, et chaque pair est responsable d'un intervalle de chaînes de caractères. Bien que cela facilite l'exécution de requêtes de type intervalle, cette approche n'est pas efficace car le nombre de pairs à visiter dans l'intervalle de la requête est linéaire.
Dans [GS04], les auteurs proposent un mécanisme adaptative pour le traitement des requêtes de type intervalle dans les DHTs. Dans [GS04], un arbre logique de recherche par intervalle (*Range Search Tree* : RST) est crée et stocke les valeurs des attributs. Quand un utilisateur pose une requête de type intervalle, cette requête est décomposée en un petit nombre de sous-requêtes qui sont ensuite envoyées à un petit nombre de pairs de la RST. Le résultat de la requête initiale est l'union des résultats des sous-requêtes. Cependant, le problème est que la charge n'est pas uniformément répartie sur les pairs. En effet, plus un pair est proche de la racine du RST, plus sa charge est élevée.

#### 2.6.3.2 Les requêtes multi-attributs

Il y a eu quelques travaux sur le support les requêtes multi-attributs dans les DHT [BAH+06]. Dans [CFCS03], les auteurs proposent *Multi-Attribute Addressable Network* (MAAN), un système P2P qui supporte le traitement des requêtes multi-attributs et des requêtes de type intervalle dans les DHTs. Ce système est construit au-dessus de Chord et est le premier système qui supporte à la fois les deux types de requêtes dans les DHTs. MAAN exécute les requêtes multi-attributs et les requêtes de type intervalle en $O(n * log(n) + n * s_{min})$ sauts de routage, où $n$ est le nombre de pairs de la DHT et $s_{min}$ l'intervalle minimale de sélectivité entre tous les attributs. L'intervalle de sélectivité est défini comme le ratio de l'intervalle de la requête par l'intervalle du domaine de l'attribut. Cependant, les auteurs ont fait remarquer qu'il y a un point d'arrêt de sélectivité à partir duquel une inondation complète du système offre de meilleures performances que leur technique. Un autre point faible de ce système est qu'il nécessite un schéma global fixe qui est connu à l'avance par tous les pairs.
Dans RDFPeers [CF04], les auteurs étendent la solution proposée dans MAAN pour permettre l'hétérogénéité des schémas des pairs. Chaque pair contient des données RDF décrites par des triplets de type $< sujet, prédicat, objet >$. Ces triplets sont hashés sur des pairs de MAAN. Les résultats expérimentaux ont montré une amélioration dans la répartition de charge.

#### 2.6.3.3 Les requêtes de jointures

Compte tenu de la méthode de distribution des données utilisée par les DHTs, à savoir l'utilisation des fonctions de hachages pour choisir l'endroit où les données doivent être stockées, l'exécution des requêtes de jointure dans ces types de systèmes est un véritable challenge.
Dans [HHL+03], les auteurs ont abordé le problème de traitement des requêtes de jointures dans PIER. Dans l'approche proposée, une clé est construite à partir d'un espace de noms (*namespace*) et de l'identifiant de la ressource (*resource ID*). Il y à un espace de nom pour chaque relation et chaque identifiant de ressource est la clé primaire des tuples de base de cette relation. Les requêtes sont envoyées de manière multicast à tous les

pairs dans les deux espaces de noms (relations) à joindre. Deux algorithmes sont utilisés pour le traitement des requêtes équi-jointures. Le premier algorithme est une version de l'algorithme de jointure de hachage symétrique (*symmetric hash join*) [WA91]. Dans cet algorithme, chaque pair qui se trouve dans l'un de ces deux espaces de noms localise les tuples pertinents, ensuite effectue le hachage de ces tuples dans l'espace de nom d'une nouvelle requête. Le second algorithme, appelé *"fetch matches"*, s'appuie sur l'hypothèse que l'une des relations est déjà hashée sur les attributs de jointure. Chaque pair dans le second espace de nom localise les tuples correspondant à la requête. Ensuite le pair récupère les tuples correspondants à partir de la première relation. Pour réduire la forte consommation de la bande passante induit par le *symmetric hash join*, les auteurs ont également proposé deux autres techniques, à savoir *symmetric semi-join rewrite* et *Bloom filter rewrite*. Ils ont évalué par simulations la performance de leurs algorithmes sur un réseau de 10000 pairs. Les résultats de ces simulations ont montré une bonne performance de ces algorithmes. Cependant, pour les cas où la relation de jointure à un grand nombre de tuples, cette solution n'est pas efficace en terme de coût de communication.
Pour éviter le multicast de la requête à un grand nombre de pairs, les auteurs de [TP03] proposent d'allouer un nombre limité de pairs spéciaux puissants, appelé *range guards*, pour le traitement des requêtes de jointures. Dans l'approche proposée, le domaine des attributs de jointures est divisé, et chaque partition est dédiée à un *range guard*. Les requêtes de jointures sont uniquement envoyées aux *range guards*, où les requêtes sont exécutées. La sélection efficace des *range guards* et l'évaluation de l'approche proposée sont considérées par les auteurs comme des travaux futurs. Cependant, l'inconvénient de cette solution est que l'existence nécessaire des pairs spéciaux n'est pas évidente dans de nombreux réseaux P2P.

## 2.7 CONCLUSION

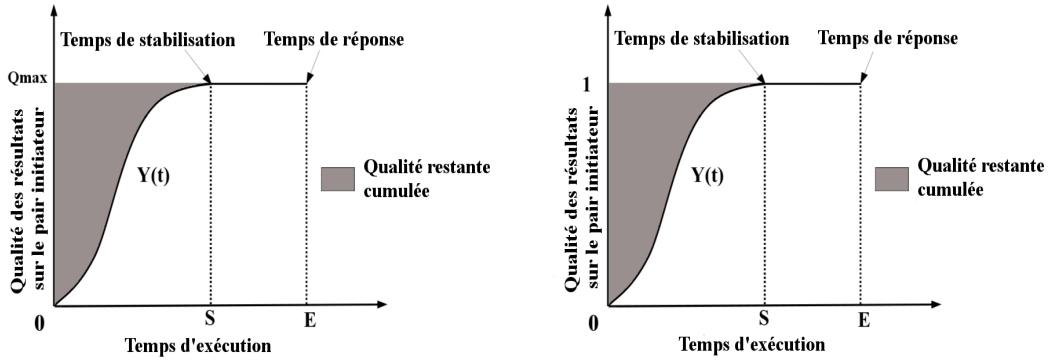
Dans ce chapitre nous avons discuté du traitement des requêtes basées sur les schémas dans les systèmes P2P. Premièrement nous avons présenté les trois principaux types de réseaux P2P : les non-structurés, les structurés et les super-pair. Nous avons brièvement décrit chacun de ces réseaux P2P, ensuite nous les avons comparés sur la base des principales exigences pour la gestion des données à savoir : l'autonomie , l'expressivité des requêtes, l'efficacité, la qualité de service, de tolérance aux pannes et la sécurité. Chacun de ces systèmes P2P offre partiellement toutes ces exigences, il n'y a pas donc de réseau P2P qui supporte toutes les exigences de la gestion des données. Par exemple dans les DHT, l'expressivité des requêtes est faible, les réseaux super-pair ne sont pas tolérants aux pannes, et les réseaux non-structurés sont souvent inefficaces.
Deuxièmement, nous avons présenté les techniques proposées pour résoudre le problème de l'hétérogénéité sémantique dans les systèmes P2P. En raison de leurs caractéristiques spécifiques, par exemple, la nature dynamique et autonome de leurs pairs, les systèmes P2P ne peuvent pas faire usage d'un schéma global centralisé comme les bases de données distribuées pour résoudre le problème de l'hétérogénéité. Nous avons donc présenté les techniques utilisées dans les systèmes P2P pour résoudre ce problème. Bien qu'il y ait eu des progrès significatifs, il y a encore beaucoup de limitations pour les approches de mappings P2P proposées. En particulier, ces techniques ont généralement besoin d'une

d'intervention humaine et la qualité des mappings n'est pas souvent très élevé.
Troisièmement, nous avons présenté les techniques de routage des requêtes à des pairs pertinents. Nous avons d'abord décrit les algorithmes de routage dans les systèmes non-structurés. La principale préoccupation dans les systèmes non-structurés est de savoir comment acheminer la requête aux autres pairs pour obtenir des réponses rapidement tout en minimisant les coûts de communication. Habituellement, les algorithmes où les pairs maintiennent un certain de nombre de statistiques sont plus performants que les autres. Toutefois, pour des systèmes très dynamiques, ces algorithmes peuvent engendrer un coût de communication très élevé. Nous avons également discuté du problème de routage des requêtes dans les systèmes structurés, en particulier dans les DHTs. Nous avons présenté les les principales géométries de routage qui sont utilisés dans les DHTs. Nous avons analysé les propriétés de routage de ces géométries et nous les avons comparés du point de vue de ces propriétés.
Enfin, nous avons décrit les techniques pour le traitement des requêtes complexes dans les systèmes de P2P. Nous nous sommes concentrés sur les requêtes top-$k$, les requêtes skyline, les requêtes de jointure, recherche par intervalle, et les requêtes multi-attributs.

# Chapitre 3

# MÉTRIQUES CENTRÉES SUR L'UTILISATEUR POUR MESURER LA PERFORMANCE DES ALGORITHMES TOP-$k$

Les requêtes de type top-$k$, en ne renvoyant à l'utilisateur que les meilleures réponses en fonction du critère qu'il a lui même défini présentent plusieurs avantages. Au bénéfice de l'utilisateur, elles évitent de le submerger avec un grand nombre de réponses pas présentées dans un ordre qu'il ne maîtrise pas. Au bénéfice du système, elles offrent des possibilités d'optimisation simplement liées au fait que l'on ne cherche pas toutes les réponses, mais seulement les $k$ meilleures. Cependant, les solutions existantes présentent un inconvénient majeur qui limite leur usage, en particulier dans les systèmes P2P. Cet inconvénient est que le temps d'attente des résultats est très long et l'utilisateur doit se montrer patient, ce qui n'est pas toujours le cas. Ce temps d'attente est encore plus important quand le système est surchargé c.-à-d. en présence de forte charge de requêtes. Pour résoudre ce problème, toutes les solutions se sont pour l'instant concentrées sur le temps de réponse, c.-à-d. le temps pour que l'algorithme termine et présente les solutions à l'utilisateur. Pour éviter les problèmes d'attente trop important nous proposons de présenter des résultats intermédiaires à l'utilisateur. Dans ce chapitre, nous présentons donc de nouvelles métriques adaptées à la mesure de la qualité des résultats intermédiaires dans le cadre des requêtes trop-$k$.




(a) Évolution de la qualité (b) Évolution de la qualité normalisée

Figure 3.1 **Qualité des réponses top-$k$ sur le pair initiateur / Temps d'exécution**

## 3.1 Fondements

Pour éviter un temps d'attente trop importante à l'utilisateur avant d'obtenir les résultats du traitement d'une requête top-$k$, nous proposons de lui fournir des résultats intermédiaires au fur et à mesure de leur disponibilité. Ce mécanisme permet à l'utilisateur de voir l'évolution de l'exécution de sa requête et il peut même être satisfait des résultats intermédiaires avant même la fin de l'exécution. Notons qu'à un certain moment de l'exécution de la requête, l'ensemble des résultats top-$k$ intermédiaire reçu par l'utilisateur ne change plus jusqu'à la fin de l'exécution de la requête. Nous notons donc l'instant à partir duquel l'ensemble des résultats top-$k$ intermédiaire ne change plus le *temps de stabilisation* (voir Figure 3.1(a)).

**Définition 1.** ***Temps de stabilisation.*** *Étant donnée une requête top-k $q$. Nous appelons* temps de stabilisation *de $q$, l'instant à partir duquel l'utilisateur reçoit l'ensemble des réponses top-k final de $q$.*

Une suite temporelle de réponses intermédiaires est donc présentée à l'utilisateur. Le temps de réponse et le temps de stabilisation ne donnent qu'une analyse très partielle de cette suite : sa durée et la durée avant stabilisation. Or si l'on admet que l'utilisateur souhaite obtenir les meilleurs (top-$k$) résultats le plus vite possible, il est naturel qu'il soit plus généralement intéressé par obtenir des résultats de la meilleur qualité possible le plus vite possible. C'est un critère permettant de qualifier la réponse apporter à ce besoin que nous cherchons maintenant. L'étude de la qualité d'une réponse intermédiaire pourrait se baser sur une mesure de précision : quelles sont les réponses finales présentes dans la réponse finale. Cette solution est possible, mais elle ne tient pas compte de la qualité intrinsèque de chaque réponse pour se focaliser uniquement sur les meilleures. Or, dans le cadre des réponses top-$k$, nous pouvons tirer partie de ce que l'utilisateur fournit une fonction de score dont le rôle est spécifiquement de qualifier la qualité des réponses possibles. Nous proposons d'exploiter cette information bien plus précise et de mesurer la qualité d'un ensemble de réponses par la somme des scores de ces réponses. Soit alors $Y(t)$ le score des réponses top-$k$ obtenues au temps $t$. Cette qualité évolue au cours du temps en fonction de l'arrivée de nouvelles réponses (voir La Figure 3.1(a)). Pour pouvoir nous affranchir

des phénomènes d'échelles produits par des fonctions de scores très différentes les unes des autres et pouvant perturber les mesures lors d'expérimentations, nous proposons de normaliser $Y(t)$. Dans cet esprit, nous divisons $Y(t)$ par la somme des scores des résultats de l'ensemble top-$k$ final. Ainsi, les valeurs de $Y(t)$ sont dans l'intervalle $[0, 1]$ et la qualité des résultats de l'ensemble top-$k$ final est égale à 1 (voir Figure 3.1(b)). Pour étudier la dynamique de l'évolution des résultats fournis à l'utilisateur, nous introduisons la *qualité restante* cumulée au cours de l'exécution de la requête. Nous définissons la qualité restante cumulée comme étant la somme des écarts de qualités entre les ensembles des résultats top-$k$ intermédiaires reçus jusqu'au temps de stabilisation et l'ensemble des résultats top-$k$ final (voir Figure 3.1(b)). Nous la définissons formellement dans la Définition 2.

**Définition 2.** ***Qualité restante cumulée.*** *Étant donnée une requête top-k $q$, soit $Y(t)$ la qualité des résultats intermédiaires de $q$ au temps $t$ sur le pair initiateur de $q$, et soit $S$ le temps de stabilisation de $q$. La qualité restante cumulée de la requête $q$, notée $C_{qg}$ est :*

$$C_{qg} = \int_0^S (1 - Y(t))\,\mathrm{d}t = S - \int_0^S Y(t)\,\mathrm{d}t \tag{3.1}$$

## 3.2 Conclusion

Dans ce chapitre, nous avons présenté de nouvelles métriques permettant de mesurer la qualité des réponses intermédiaires dans le cadre des requêtes top-$k$. Ces mesures sont : le *temps de stabilisation des réponses* et la *qualité restante cumulée au cours de l'exécution de la requête*. Ces mesures viennent en complément du temps de réponse pour mesurer le temps d'attente de l'utilisateur pour le traitement des requêtes top-$k$.

# CHAPITRE 4
# UNE NOUVELLE APPROCHE DE TRAITEMENT DES REQUÊTES TOP-$k$ DANS LES SYSTÈMES P2P

Dans le chapitre précédent, nous avons présenté deux nouvelles métriques adaptées à la mesure de la qualité de résultats intermédiaires dans le cadre des requêtes trop-$k$ : *le temps de stabilisation* des réponses et la *qualité restante cumulée* au cours de l'exécution de la requête. Ces deux nouvelles métriques viennent en complément du temps de réponse pour qualifier le temps d'attente de l'utilisateur dans le traitement des requêtes top-$k$ dans les systèmes P2P. Dans ce chapitre, nous prenons en compte ces nouvelles métriques c.-à-d. *temps de stabilisation et la qualité restante cumulée* pour redéfinir formellement le problème du traitement d'une requête top-$k$ le plus tôt possible (ASAP [1] top-$k$). Ensuite, nous proposons une famille d'algorithmes appelée ASAP. Cette nouvelle proposition est évaluée expérimentalement, et les résultats obtenus montrent une amélioration notoire, notre approche retournant les réponses des requêtes top-$k$ nettement plus tôt aux utilisateurs par rapport aux approches classiques avec un coût raisonnable.

## 4.1 MODÈLE DU SYSTÈME

Sont présentés ici, le modèle général des systèmes P2P non-structurés utilisé pour décrire notre solution ainsi que des définitions relatives aux requêtes top-$k$.

### 4.1.1 Modèle de réseau P2P non-structuré

Nous modélisons un réseau P2P non-structuré de $n$ pairs par un graphe non orienté $G = (P, E)$, où $P = \{p_0, p_1, \cdots, p_{n-1}\}$ est un ensemble de pairs et $E$ l'ensemble des connexions entre les pairs. Pour $p_i, p_j \in P, (p_i, p_j) \in E$ signifie que $p_i$ et $p_j$ sont des voisins. Nous

1. As Soon As Possible

notons par $N(p_i)$, l'ensemble des pairs auquel $p_i$ est directement connecté, donc $N(p_i) = \{p_j | (p_i, p_j) \in E\}$. La valeur $\|N(p_i)\|$ est appelée le degré $p_i$ (c.-à-d. le nombre de voisins de $p_i$). Le degré moyen des pairs dans $G$ est appelé le degré moyen de $G$ et est noté $\varphi$. Le *r-voisinage* $N^r(p)$ $(r \in \mathbb{N})$ d'un pair $p \in P$ est défini comme étant l'ensemble des pairs qui sont au plus à $r$ pas (ou sauts) de $p$, donc

$$N^r(p) = \begin{cases} \{p\} & si\ r = 0 \\ \{p\} \bigcup\limits_{p' \in N(p)} N^{r-1}(p') & si\ r \geq 1 \end{cases}$$

Chaque pair $p \in P$ détient et maintient un ensemble $D(p)$ d'éléments de données tels des images, des documents ou des données relationnelles (c.-à-d. des tuples). Nous notons par $D^r(p)(r \in \mathbb{N})$, l'ensemble des éléments de données qui sont dans $N^r(p)$, ainsi

$$D^r(p) = \bigcup_{p' \in N^r(p)} D(p')$$

Dans notre modèle, quand l'utilisateur soumet une requête sur un pair $p_0 \in P$ (le pair initiateur), cette requête est propagée du pair initiateur vers les pairs voisins jusqu'à ce que la valeur du *time-to-live* (ttl) de la requête atteigne 0 ou qu'il n'y ait plus de pair à qui transmettre cette requête. Ainsi, le flux de traitement d'une requête peut être représenté comme un arbre, appelé l'arbre de propagation de la requête. Cet arbre a pour racine le pair initiateur $p_0$ et inclut tous les pairs appartenant à $N^{ttl}(p_0)$. L'ensemble des fils d'un pair $p \in N^{ttl}(p_0)$ dans l'arbre de propagation de la requête $q$ est noté $\psi(p, q)$.

### 4.1.2 Requêtes top-$k$

Nous caractérisons chaque requête top-$k$ $q$ par un triplet $< qid, c, ttl, k, f, p_0 >$ tels que :
- $qid$ est l'identifiant de la requête, $c$ est la requête proprement dite (par exemple une requête de type SQL)
- $ttl \in \mathbb{N}$ est le nombre de sauts maximal que peut atteindre $q$ (ce nombre est défini par l'utilisateur)
- $k \in \mathbb{N}^*$ est le nombre de résultats demandé par l'utilisateur
- $f : \mathcal{D} \times \mathcal{Q} \rightarrow [0,1]$ est une fonction de score qui évalue le score de pertinence (c.-à-d. la qualité) d'un élément de donnée par rapport à une requête ($\mathcal{D}$ est l'ensemble des données et $\mathcal{Q}$ est l'ensemble des requêtes)
- $p_0 \in P$ est le pair initiateur de la requête $q$

L'ensemble des résultats top-$k$ d'une requête $q$ donnée est constitué des $k$ meilleurs résultats parmi tous les éléments de données que détiennent tous les pairs ayant reçu $q$. Nous le définissons formellement comme suit.

**Définition 3.** ***Ensemble des résultats top-$k$.*** *Étant donnée une requête top-k $q$, soit $D' = D^{q.ttl}(q.p_0)$. L'ensemble des résultats top-k de $q$, noté $Top^k(D', q)$, est un ensemble trié par ordre décroissant des scores tel que :*

1. $Top^k(D', q) \subseteq D'$ ;
2. *Si* $\|D'\| < q.k$, $Top^k(D', q) = D'$, *sinon* $\|Top^k(D', q)\| = q.k$ ;
3. $\forall d \in Top^k(D', q),\ \forall d' \in D' \setminus Top^k(D', q), \quad q.f(d, q.c) \geq q.f(d', q.c)$

**Définition 4.** ***Rang d'un résultat.*** *Étant donnée une requête top-k ayant pour ensemble des résultats $I$, nous définissons le rang d'un résultat $d \in I$, noté par $rank(d, I)$, comme étant la position de $d$ dans l'ensemble des résultats $I$.*

Notons que le rang d'un élément top-$k$ est dans l'intervalle $[1; k]$.

Dans les systèmes P2P non-structurés à grande échelle, les pairs peuvent avoir des différentes capacités de traitement et peuvent stocker des volumes de données différents. En outre, les pairs sont autonomes du point de vue des ressources qu'ils peuvent allouer pour traiter une requête donnée. Ainsi, certains pairs peuvent donc traiter plus rapidement une requête donnée plus que d'autres. Intuitivement, l'ensemble des résultats top-$k$ intermédiaire d'un pair $p$ donné est constitué des $k$ meilleurs résultats qu'il a déjà reçu de ses fils et de ses résultats locaux (si la requête a été déjà traitée localement par $p$). Formellement, nous définissons l'ensemble des résultats top-$k$ intermédiaire d'une requête top-$k$ comme suit.

**Définition 5.** ***Ensemble des résultats top-$k$ intermédiaire.*** *Étant donné une requête top-k $q$, et $p \in N^{q.ttl}(q.p_0)$. Soit $D_1$ l'ensemble des réponses de $q$ qui a été déjà reçu par $p$ de la part des pairs qui sont dans $\psi(p, q)$ et $D_2 = D_1 \cup D(p)$. L'ensemble des résultats top-k intermédiaire de $q$ sur le pair $p$, noté par $I_q(p)$ est défini comme suit :*

$$I_q(p) = \begin{cases} Top^k(D_2, q) & \textit{si } p \textit{ a déjà traité localement } q \\ Top^k(D_1, q) & \textit{sinon} \end{cases}$$

## 4.2 DÉFINITION DU PROBLÈME

Pour pouvoir définir clairement le problème auquel nous intéressons dans ce chapitre, nous appuyons sur quelques idéalisations concernant la gestion des schémas et l'architecture P2P non-structuré. Nous supposons que les pairs sont en mesure d'exprimer des requêtes sur leur propre schéma sans s'appuyer sur un schéma global centralisé comme dans les systèmes d'intégration de données [TIM+03]. Plusieurs solutions ont été proposées dans l'état de l'art pour le mapping des schémas de façon décentralisée. Cependant, cette question sort du cadre de cette thèse et nous supposons que la gestion des schémas de façon décentralisée est effectuée avec l'une des techniques existantes, par exemple [OST03], [TIM+03] et [AMPV06a]. Nous supposons également que tous les pairs du système que nous considérons sont dignes de confiance et qu'ils sont aussi coopératifs. Dans la suite de cette partie, nous allons définir formellement le problème du traitement ASAP d'une requête top-$k$.

### 4.2.1 Énoncé du problème

Pour traduire le besoin de l'utilisateur d'avoir des réponses de bonne qualité le plus vite possible, nous considérons que l'objectif est de minimiser le temps de stabilisation et la qualité restante cumulée. Formellement, nous définissons le problème du traitement ASAP d'une requête top-$k$ comme suit. Étant donné une requête top-$k$ $q$, soit $S$ le temps de stabilisation de $q$ et soit $C_{qg}$ la qualité restante cumulée au cours de l'exécution de $q$. Le problème est de minimiser $C_{qg}$ et $S$ tout en évitant un coût de communication important.

Notons que cette approche est relativement différente de l'énoncé classique qui consiste à se concentrer sur la réduction du temps de réponse. Nous considérons ici qu'il est prioritaire de fournir à l'utilisateur des résultats de bonne qualité et les top-$k$ le plus vite possible, la preuve que les résultats fournis sont bien les meilleurs pouvant être plus longue que le temps nécessaire à les trouver.

## 4.3 VUE D'ENSEMBLE SUR LE TRAITEMENT DES REQUÊTES TOP-$k$ PAR L'APPROCHE ASAP

Le traitement ASAP d'une requête top-$k$ se déroule en deux phases principales. La première phase est la propagation et l'exécution locale de la requête. La deuxième phase est la remontée des résultats des pairs à la requête au pair initiateur via l'arbre de propagation de cette requête.

### 4.3.1 Propagation et exécution locale des requêtes

Le traitement d'une requête débute sur le pair initiateur, c.-à-d. le pair qui a émis la requête top-$k$ $q$. L'initiateur de la requête effectue plusieurs initialisations. Premièrement, il initialise la valeur du *ttl* qui est soit spécifiée par l'utilisateur ou fixée par défaut par le système et le nombre $k$ de résultats souhaité par l'utilisateur. Deuxièmement, il crée un identifiant *qid* unique pour la requête $q$ qui est utile pour distinguer les nouvelles requêtes de celles qui ont été déjà reçues. Notons que *qid* est crée à partir de l'identifiant du pair et d'un compteur local géré par le pair. Ensuite, $q$ est inclus dans un message qui est diffusé par l'initiateur à ses voisins accessibles. **Algorithme 1** montre le pseudo-code de propagation des requêtes. Chaque pair qui reçoit le message contenant $q$ vérifie le *qid* de $q$ (voir ligne 2, Algorithme 1). Si c'est la première fois que le pair a reçu $q$, il enregistre la requête dans la liste des requêtes ayant été déjà reçues. Il enregistre aussi l'adresse de l'expéditeur comme l'adresse de son parent, et décrémente le *ttl* de la requête de 1 (voir lignes 3 à 4, Algorithme 1). Si le *ttl* est supérieur à 0, alors le pair envoie le message de la requête à tous voisins excepté son parent (voir lignes 5 à 7, Algorithme 1). Ensuite, il exécute $q$ localement. Si $q$ a été déjà reçue, alors si l'ancien *ttl* de $q$ est plus petit que son nouveau *ttl*, le pair procède de façon identique comme dans le cas où $q$ a été reçue la première fois sans cependant exécuter localement la requête (voir lignes 10 à 18, Algorithme 1), sinon le pair envoie un message signalant qu'il a déjà reçu cette requête au pair qui lui a envoyé $q$.

### 4.3.2 Remontée des réponses

Rappelons que, quand un pair soumet une requête top-$k$ $q$, les résultats locaux des pairs qui ont reçu $q$ sont remontés vers le pair initiateur en utilisant l'arbre de propagation de la requête $q$. Dans ASAP, la décision d'un pair d'envoyer des résultats intermédiaires est basée sur l'impact d'amélioration calculé en utilisant le ratio entre l'ensemble des résultats top-$k$ intermédiaire courant du pair et l'ensemble des résultats top-$k$ intermédiaire qu'il a déjà envoyé à son parent. Cet impact d'amélioration peut être calculé de deux façons : en utilisant le score ou le rang des résultats de l'ensemble top-$k$. Par conséquent, nous

**Algorithme 1** : *receive_Query(msg)*

```
   input  : msg, a query message.
 1 begin
 2   if (!already_Received(msg.getID()) then
 3     memorize(msg);
 4     msg.decreaseTTL();
 5     if (msg.getTTL() > 0) then
 6       forwardToNeighbors(msg);
 7     end
 8     executeLocally(msg.getQuery());
 9   else
10     qid = msg.getID();
11     oldMsg = SeenQuery(qid).;
12     if (msg.getTTL() > oldMsg.TTL()) then
13       memorize(msg);
14       msg.decreaseTTL();
15       if (msg.getTTL() > 0) then
16         forwardToNeighbors(msg);
17       end
18       sendDuplicateSignal(qid, oldMsg.getSender());
19     else
20       sendDuplicateSignal(qid, msg.getSender());
21     end
22   end
23 end
```

introduisons deux types d'impact d'amélioration : *impact d'amélioration basé sur le score* et *impact d'amélioration basé sur le rang.*
Intuitivement, l'impact d'amélioration basé sur le score sur un pair donné et pour une requête top-$k$ donnée est le gain de score de l'ensemble des résultats top-$k$ intermédiaire courant de ce pair par rapport à l'ensemble des résultats top-$k$ intermédiaires qu'il a déjà envoyé à son parent.

**Définition 6.** ***Impact d'amélioration basé sur le score.*** *Étant donnée une requête top-k $q$, et un pair $p \in N^{q.ttl}(q.p_0)$, soit $T_{cur}$ l'ensemble des résultats top-k intermédiaire courant de la requête $q$ sur le pair $p$ et soit $T_{old}$ l'ensemble des résultats top-k intermédiaire de $q$ déjà envoyé par $p$. L'impact d'amélioration basé sur le score de $q$ sur le pair $p$, noté $IScore(T_{cur}, T_{old)}$ est calculé comme suit :*

$$IScore(T_{cur}, T_{old}) = \frac{\sum_{d \in T_{cur}} q.f(d, q.c) - \sum_{d' \in T_{old}} q.f(d', q.c)}{k} \tag{4.1}$$

Notons que dans la formule 4.1, nous divisons par $k$ au lieu de $\|T_{cur} - T_{old}\|$ parce-que nous ne voulons pas que $IScore(T_{cur}, T_{old})$ soit une moyenne qui ne sera donc pas très sensible aux valeurs des scores (c.-à-d. qu'il est difficile de distinguer l'apparition des résultats de meilleurs scores). Les valeurs de l'impact d'amélioration basé sur le score sont dans l'intervalle $[0, 1]$.
Intuitivement, l'impact d'amélioration basé sur le rang sur un pair donné et pour une requête top-$k$ est la perte de rang ou de position des résultats de l'ensemble top-$k$ intermédiaire déjà envoyé par ce pair dû à la réception de nouveaux résultats intermédiaires.

**Définition 7.** ***Impact d'amélioration basé sur le rang.*** *Étant donné une requête*

*top-k q, et un pair $p \in N^{q.ttl}(q.p_0)$, soit $T_{cur}$ l'ensemble des résultats top-k intermédiaire courant de la requête q sur le pair p et soit $T_{old}$ l'ensemble des résultats top-k intermédiaire de q déjà envoyé par p. L'impact d'amélioration basé sur le rang de q sur le pair p, noté $IRank(T_{cur}, T_{old})$ est calculé comme suit :*

$$IRank(T_{cur}, T_{old}) = \frac{\sum_{d \in T_{cur} \setminus T_{old}} (k - rank(d, T_{cur}) + 1)}{\frac{k * (k+1)}{2}} \tag{4.2}$$

Notons que dans la Formule 4.2, nous divisons par $\frac{k*(k+1)}{2}$ qui est la somme des rangs d'un ensemble contenant $k$ éléments. Les valeurs de l'impact d'amélioration basé sur le rang sont dans l'intervalle $[0, 1]$.
Notons aussi que, dans le but de minimiser le trafic réseau, ASAP ne remonte pas les résultats eux-mêmes (qui pourraient constituer une charge importante sur le réseau), mais uniquement leurs scores et leurs adresses appelés *scores-listes*. Un score-liste est tout simplement une liste de couples $(ad, s)$, telle que $ad$ est l'adresse du pair possédant la donnée et $s$ le score de cette donnée.
Une façon simple de décider quand un pair doit remonter vers son parent les nouveaux résultats intermédiaires qu'il a reçu est de fixer une valeur minimale (seuil) que doit atteindre l'impact d'amélioration. Cette valeur est fixée initialement par l'application et elle est la même pour tous les pairs du système. Notons également que ce seuil ne change pas durant l'exécution de la requête. En utilisant les deux types d'impact d'amélioration que nous avons introduit précédemment, nous avons donc deux types d'approches basées sur un seuil statique. La première approche utilise l'impact d'amélioration basé sur le score et la seconde utilise l'impact d'amélioration basé sur le rang.
Un algorithme générique pour nos approches basées sur un seuil statique est présenté dans **Algorithme 2**. Dans ces approches, chaque pair maintient pour chaque requête un ensemble $T_{old}$ des résultats top-$k$ intermédiaire qu'il a déjà envoyé à son parent et un ensemble $T_{cur}$ des résultats top-$k$ intermédiaire courant. Quand un pair reçoit de ses fils un nouvel ensemble de résultats $N$ ou son propre ensemble $N$ de résultats après un traitement local d'une requête, il met à jour l'ensemble $T_{cur}$ avec des résultats de l'ensemble $N$ (voir ligne 2, Algorithme 2). Ensuite, il calcule l'impact d'amélioration *imp* de $T_{cur}$ par rapport à $T_{old}$ (voir ligne 3, Algorithme 2). Si *imp* est supérieur ou égal au seuil *delta* défini par l'application ou si le pair n'a plus de résultats à atteindre de ses fils, le pair envoie l'ensemble $T_{tosend} = T_{cur} \setminus T_{old}$ à son parent puis affecte $T_{cur}$ à $T_{old}$ (voir lignes 4 à 7, Algorithme 2).

## 4.4 APPROCHES BASÉES SUR UN SEUIL DYNAMIQUE POUR LA REMONTÉE DES RÉPONSES

Bien que les approches basées sur seuil statique sont intéressantes pour fournir des résultats de bonne qualité le plus rapidement à l'utilisateur, elles peuvent être bloquantes si les résultats de scores élevés sont remontés avant les résultats de faibles scores. En d'autres termes, l'envoi en premier des résultats de scores élevés va entrainer une diminution de l'impact d'amélioration des résultats suivants. Cela est dû au fait que l'impact d'amélioration

**Algorithme 2** : $Streat(k, T_{cur}, T_{old}, N, delta, Func)$

```
input : k, nombre de résultats ; T_cur, top-k courant ; T_old, top-k déjà envoyé ; N, nouvel ensemble de résultats ;
        delta, seuil de l'impact ; Func, type de l'impact d'amélioration.
1 begin
2     T_cur = mergingSort_Topk(k, T_cur, N);
3     imp = Func(T_cur, T_old);
4     if ((imp ≥ delta) or all_Results()) then
5         T_tosend = T_cur \ T_old;
6         send_Parent(T_tosend, all_Results());
7         T_old = T_cur;
8     end
9 end
```

prend en compte les résultats qui ont été déjà envoyés par un pair. Ainsi, les résultats de faibles scores, même s'ils sont dans l'ensemble des résultats top-$k$ final peuvent être retournés à la fin de l'exécution de la requête. Pour faire face à ce problème, une façon intéressante serait d'avoir un seuil dynamique, c.-à-d. un seuil qui décroit au fur et à mesure que l'exécution de la requête progresse. Cependant, il faudrait trouver le bon paramètre dont dépend ce seuil. Nous avons identifié deux solutions possibles pour mettre en place un seuil dynamique. La première solution est d'utiliser une estimation du temps d'exécution des requêtes. Cependant, l'estimation du temps d'exécution des requêtes dans un système P2P à grande échelle est très difficile car ce temps dépend des dynamiques du réseau, telles que la connectivité, la densité, la contention d'accès au médium, etc., et du pair le plus lent. La seconde solution, qui apparaît la plus pratique est d'utiliser pour chaque pair la proportion de pairs dans son sous-arbre, y compris lui-même (c.-à-d. tous ses descendants dans l'arbre de la propagation de la requête et lui-même) qui ont déjà traité localement la requête pour diminuer le seuil.

### 4.4.1 Couverture locale des réponses d'un pair

**Définition 8.** ***Couverture locale des réponses d'un pair****. Étant donnée une requête top-k $q$, et $p \in N^{q.ttl}(q.p_0)$, soit $\mathcal{A}$ l'ensemble des pairs du sous-arbre dont la racine est $p$ dans l'arbre de propagation de la requête $q$. Soit $\mathcal{E}$ l'ensemble des pairs appartenant à $\mathcal{A}$ qui ont déjà traité localement $q$. La couverture locale des réponses du pair $p$ pour la requête $q$, notée par $Cov(\mathcal{E}, \mathcal{A})$, est calculée en utilisant la formule suivante :*

$$Cov(\mathcal{E}, \mathcal{A}) = \frac{\|\mathcal{E}\|}{\|\mathcal{A}\|}$$

Les valeurs de la couverture locale des réponses d'un pair sont dans l'intervalle $[0, 1]$.
Il est à noter qu'il est très difficile de calculer la valeur exacte de la couverture locale des réponses d'un pair sans engendrer un nombre supplémentaire de messages sur le réseau. Cela est dû au fait que chaque pair doit envoyer un message à son parent à chaque fois que la valeur de sa couverture locale des réponses change. Ainsi, quand un pair se trouvant à $m$ sauts de l'initiateur de la requête met à jour sa couverture locale des réponses, $m$ messages seront envoyés sur le réseau. Pour faire face à ce problème, une solution intéressante est d'avoir une estimation de cette valeur au lieu de la valeur exacte.
L'estimation de la couverture locale des réponses d'un pair peut être effectuée en utilisant deux stratégies différentes : la stratégie optimiste et la stratégie pessimiste. Dans

la stratégie optimiste, chaque pair $p$ calcule la valeur initiale de sa couverture locale des réponses en se basant uniquement sur ses nœuds fils. Cette valeur est ensuite mis à jour progressivement au fur et à mesure que les pairs se trouvant dans le sous-arbre de $p$ font remonter leurs résultats à la requête. En effet, chaque pair inclut dans chaque message de réponse envoyé à son parent le nombre de pairs dans son sous-arbre (y compris lui-même) qui ont déjà traité localement la requête et le nombre total de pairs dans son sous-arbre y compris lui-même. Contrairement à la stratégie optimiste, dans la stratégie pessimiste, l'estimation de la couverture locale des réponses est calculé au début de l'exécution de la requête par chaque pair en se basant sur le *ttl* avec lequel il a reçu la requête et le degré moyen des pairs du système. Comme dans le cas de la stratégie optimiste, cette valeur est mise à jour progressivement par chaque pair au fur et à mesure que les pairs se trouvant dans leurs sous-arbres font remonter leurs résultats à la requête.
Dans nos approches basées sur un seuil dynamique, nous estimons la couverture locale des réponses d'un pair en utilisant la stratégie pessimiste parce-que la valeur de cette estimation est plus stable que celle de la stratégie optimiste. Dans la section suivante, nous donnons plus de détails sur notre stratégie d'estimation pessimiste de la couverture locale des réponses d'un pair.

### 4.4.2 Estimation pessimiste de la couverture locale des réponses

Afin d'estimer la couverture locale des réponses, chaque pair $p_i$ maintient pour chaque top-$k$ $q$ et pour chaque fils $p_j$ un ensemble $\mathcal{C}_1$ de couples $(p_j, a)$ où $a \in \mathbb{N}$ est le nombre de pairs dans le sous-arbre du pair $p_j$ y compris lui-même. $p_i$ maintiens aussi un ensemble $\mathcal{C}_2$ de couples $(p_j, e)$ où $e \in \mathbb{N}$ est le nombre total de pairs dans le sous-arbre de $p_j$ y compris lui-même qui ont déjà traité localement la requête $q$. Soit alors $ttl'$ le time-to-live avec lequel $p_i$ a reçu la requête $q$ et soit $\varphi$ le degré moyen des pairs du système. Au début de l'exécution de la requête, pour tous les fils du pair $p_i$, $e = 0$ et $a = \sum_{u=0}^{ttl'-2} \varphi^u$. Durant l'exécution de la requête, quand un fils $p_j$ de $\psi(p_i, q)$ veut envoyer des résultats à $p_i$, il insère dans le message de réponse son couple de valeurs $(e, a)$. Une fois que $p_i$ reçoit ce message, il le décompresse, récupère ces valeurs (c.-à-d. $e$ et $a$) et met à jour les ensembles $\mathcal{C}_1$ et $\mathcal{C}_2$. La couverture locale des réponses d'un pair $p_i$ pour une requête $q$ est alors estimée en utilisant la Formule 4.3.

$$\widetilde{Cov}(\mathcal{C}_1, \mathcal{C}_2) = \frac{\sum_{(p_j, e) \in \mathcal{C}_1} e}{\sum_{(p_j, a) \in \mathcal{C}_2} a} \tag{4.3}$$

Notons que les valeurs de l'estimation de la couverture locale des réponses d'un pair sont dans l'intervalle $[0, 1]$.

### 4.4.3 Fonction de seuil dynamique

Dans les approches basées sur un seuil dynamique, le seuil d'impact d'amélioration utilisé par un pair à un instant $t$ donné de l'exécution de la requête dépend de sa couverture locale des réponses à cet instant. Ce seuil d'impact d'amélioration diminue au fur et à

mesure que la couverture locale des réponses augmente. Pour diminuer le seuil d'impact d'amélioration utilisé par un pair au fur à mesure que sa couverture locale des réponses augmente, nous utilisons une fonction linéaire qui permet de fixer leur seuil d'impact d'amélioration pour une valeur donnée de la couverture des réponses. Nous définissons formellement cette fonction de seuil comme suit.

**Définition 9.** ***Fonction de Seuil Dynamique.*** *Étant donnée une requête top-k $q$ et $p \in N^{q.ttl}(q.p_0)$, le seuil d'impact d'amélioration utilisé par $p$ durant l'exécution de $q$, est une fonction décroissante monotone $H$ telle que :*

$$H : \left| \begin{array}{lcl} [0,1] & \rightarrow & [0,1] \\ & & \\ x & \mapsto & -\alpha * x + \alpha \end{array} \right. \tag{4.4}$$

avec $\alpha \in [0,1[$. Notons que $x$ est la couverture locale des réponses d'un pair à un instant donné et $\alpha$ le seuil d'impact d'amélioration initial (c.-à-d. $H(0) = \alpha$).

### 4.4.4 Réduction des coûts de communication

L'utilisation d'un impact d'amélioration basé sur le rang a l'inconvénient de ne pas réduire autant que possible le trafic réseau. Cela est dû au fait que la valeur de l'impact d'amélioration basé sur le rang est égale à 1 (la valeur maximale qu'il peut atteindre) quand un pair reçoit le premier ensemble de résultats (de l'un de ses fils ou après une exécution locale de la requête). Ainsi, chaque pair envoie toujours un message sur le réseau quand il reçoit le premier ensemble de résultats. Pour faire face à ce problème et ainsi réduire le coût de communication, nous utilisons les couvertures locales des réponses des pairs pour les empêcher d'envoyer un message quand ils reçoivent leur premier ensemble de résultats. L'idée est de permettre aux pairs de commencer à envoyer un message si et seulement si leur couverture locale de réponses atteigne un certain seuil prédéfini. A partir de ce seuil de couverture locale des réponses, les pairs envoient des résultats intermédiaires en fonction du seuil d'impact d'amélioration obtenu à partir de la fonction de seuil dynamique $H$ définie ci-dessus.

### 4.4.5 Algorithmes basés sur un seuil dynamique

Les algorithmes de nos approches utilisant un seuil dynamique sont basés sur les mêmes principes que ceux utilisant un seuil statique. Un algorithme générique pour nos approches utilisant un seuil dynamique est donné dans **Algorithme 3**. Quand un pair reçoit un nouveau ensemble de résultats $N$ de ses fils (ou son propre ensemble de résultats après une exécution locale de la requête), il met à jour premièrement l'ensemble $T_{cur}$ de ses résultats top-$k$ intermédiaire courant avec les résultats de $N$ (voir ligne 2, Algorithme 3). Si la couverture locale courante des réponses *cov* du pair est supérieure à la valeur du seuil de couverture prédéfinie *cov'*, alors le pair calcule le seuil d'impact d'amélioration en utilisant la fonction de seuil dynamique $H$ et ensuite l'impact d'amélioration *imp* (voir lignes 3 à 5, Algorithme 3). Si *imp* est supérieur ou égal au seuil *delta* ou si le pair n'a plus de résultats à atteindre de ses fils, le pair envoie l'ensemble $T_{tosend} = T_{cur} \setminus T_{old}$ à son parent puis affecte $T_{cur}$ à $T_{old}$ (voir lignes 6 à 9, Algorithme 3). Rappelons que $T_{cur}$ est

**Algorithme 3** : $Dtreat(k, T_{cur}, T_{old}, N, Func, cov, c\bar{o}v, H)$

**input** : $k$ ; $T_{cur}$ ; $T_{old}$ ; $N$ ; $Func$ ; $cov$, couverture locale de réponses ; $c\bar{o}v$, seuil de la couverture de réponses ; $H$, la function de seuil dynamique.

```
1  begin
2      T_cur = mergingSort_Topk(k, T_cur, N);
3      if (cov > cōv) then
4          delta = H(cov);
5          imp = Func(T_cur, T_old);
6          if ((imp ≥ delta) or all_Results()) then
7              T_tosend = T_cur \ T_old;
8              send_Parent(T_tosend, all_Results());
9              T_old = T_cur;
10         end
11     end
12 end
```

l'ensemble des résultats top-$k$ intermédiaire courant et $T_{old}$ l'ensemble des résultats top-$k$ intermédiaire que le pair a déjà envoyé à son parent.

### 4.4.6 Exemple

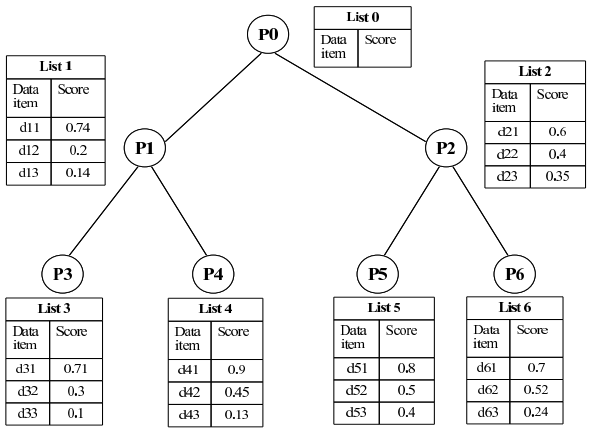



Figure 4.1 **Exemple d'arbre de propagation de requête d'un système P2P non-structuré**

Pour mieux illustrer le traitement ASAP d'une requête top-$k$, nous considérons un arbre de propagation d'une requête dans le graphe d'un réseau constitué de sept pairs $p_0, \cdots, p_6$ comme le montre la Figure 4.1. Supposons que $p_0$ émet une requête top-3 (c.-à-d. $k = 3$) $q$ , et que l'ordre de fin du traitement local de $q$ sur ces différents pairs le suivant $p_0$, $p_4$, $p_1$, $p_5$, $p_3$, $p_6$. Soient $list_0$, $\cdots$, $list_6$ les listes de résultats locaux respectifs des pairs $p_0$, $\cdots$, $p_6$ après le traitement local de la requête $q$. Nous illustrons uniquement dans cette section l'approche ASAP utilisant un impact d'amélioration basé sur le score et ceci sur la portion $(p_0, p_1, p_4)$ de l'arbre de propagation de la requête considérée. Notons aussi que nous n'illustrons que la version statique de cette approche. Dans notre exemple, nous supposons que le seuil d'impact d'amélioration basé sur le score est $delta = 0.2$. L'algorithme se déroule comme suit. Quand $p_4$ finit de traiter localement $q$, il envoie immédiatement sa liste de résultats $list_4$ à son parent $p_1$ (car $p_4$ n'a pas de résultats à attendre de ses fils).

Quand $p_1$ reçoit les résultats de $p_4$, il calcule l'impact d'amélioration basé sur le score de sa liste de résultats top-3 courant par rapport à la liste de résultats top-3 qu'il a déjà envoyé à son parent ce qui donne $\frac{(0.9+0.45+0.13)-(0)}{3} = 0.493$. Comme cette valeur est supérieur au seuil prédéfini qui est égal 0.2, $p_1$ envoie $list_4$ à $p_0$. Une fois que $p_1$ finit de traiter localement la requête $q$, il calcule l'impact d'amélioration basé sur le score de ses résultats top-3 courant par rapport aux résultats de la liste $list4$ (c.-à-d. les résultats top-3 qu'il a déjà envoyé à son parent), ce qui donne $\frac{(0.9+0.74+0.45)-(0.9+0.45+0.13)}{3} = 0.203$. Compte tenu du fait que la valeur de l'impact d'amélioration est supérieur au seuil prédéfini, $p_1$ envoie donc à $p_0$ l'élément dans la $list_1$ dont le score 0.74 car cet élément est le seul résultat à remonter vers le parent à cet instant.

## 4.5 ANALYSE THÉORIQUE DES COÛTS

Dans cette section, nous évaluons analytiquement le coût d'ASAP en termes de nombre de messages de réponse et de volume de données transféré sur le réseau (en nombre de bits) pour retourner le résultat top-$k$ final à l'utilisateur.
Avec ASAP, chaque pair envoie un message de réponses dans le meilleur cas. Ainsi, si $n$ est le nombre de pairs dans le réseau, alors le nombre de messages de réponse dans le meilleur cas est égal à $n-1$. Dans le pire cas, le nombre de messages de réponse envoyé par un pair dépend de sa profondeur dans l'arbre de propagation de la requête (c.-à-d. le $ttl$ avec lequel le pair a reçu la requête). Soit $P(i)$ le nombre de pairs qui se trouve à $i$ pas ou sauts de l'initiateur d'une requête $q$ (émise avec un time-to-live $ttl$) dans l'arbre de propagation de la requête $q$. Dans le pire cas, le nombre de messages de réponse noté $n_{asap}$ est :

$$\begin{aligned} n_{asap} &= 0*P(0)+1*P(1)+2*P(2)+\cdots+(ttl-2)*P(ttl-2)+ttl*P(ttl) \\ &\leq ttl*P(1)+ttl*P(2)+\cdots+ttl*P(ttl) \\ &\leq ttl*[P(1)+P(2)+\cdots+P(ttl)] \\ n_{asap} &\leq ttl*(n-1). \end{aligned}$$

Pour résumer, le nombre de messages de réponse envoyé par ASAP est tel que :

$$n-1 \leq n_{asap} \leq ttl*(n-1)$$

Soit alors $k$ le nombre de résultats souhaité par l'utilisateur et soit $z$ la taille en bits de chaque élément d'un ensemble de résultats. Dans le meilleur cas, le volume de données transféré sur le réseau est égal $k*z$. Dans le pire cas, puisque le nombre de messages de réponses est $ttl*(n-1)$, le volume de données transféré sur le réseau est égal à $k*z*ttl*(n-1)$. Ainsi le volume de données transféré sur le réseau dans le cas d'ASAP, noté $v_{asap}$ est :

$$z*k \leq v_{asap} \leq k*z*ttl*(n-1)$$

Pour résumer, le coût de communication d'ASAP en terme de nombre de messages de réponses est $O(n)$ et le volume de données transféré sur le réseau est $O(n*k)$.

## 4.6 GESTION DES PANNES DES PAIRS

Une des caractéristiques principale des systèmes P2P est le comportement dynamique des pairs. Pour cela, il peut arriver que certains pairs quittent le système (ou tombent

en panne) au cours du traitement d'une requête. Comme conséquence, des pairs peuvent devenir inaccessibles pendant la phase de remontée des résultats vers le pair initiateur. Dans cette section, nous proposons des extensions à nos algorithmes de traitement de requête top-$k$ pour faire donc face à ce problème.

### 4.6.1 Technique efficace d'amélioration de la précision du résultat top-$k$ en cas des pannes

Dans la phase de remontée des résultats vers le pair initiateur, chaque pair $p$ fait remonter les résultats des pairs de son sous-arbre vers son parent. Il peut arriver que le parent de $p$ soit inaccessible car il a quitté le système ou qu'il est tombé en panne. La question est de savoir quel est le chemin qu'on doit choisir pour faire parvenir les résultats intermédiaires du pair $p$ au pair initiateur de la requête. Pour répondre à cette question, une solution naïve consiste à ce que $p$ envoie directement ses résultats intermédiaires au pair initiateur quand le parent de $p$ est inaccessible. Rappelons que lors de la phase de propagation d'une requête l'adresse IP et le port du pair initiateur de la requête sont communiqués à tous les pairs qui ont reçu cette requête. Cependant, l'approche naïve présente quelques inconvénients. Premièrement, elle peut engendrer plusieurs fusions de résultats sur le pair initiateur qui peuvent être très couteuse en ressources. Deuxièmement, en envoyant les résultats intermédiaires des pairs dont les parents sont inaccessibles directement au pair initiateur, nous réduisons la capacité des pairs d'éliminer les résultats intermédiaires non intéressants et ceci peut augmenter de façon significative le volume de données transféré sur le réseau.
Notre solution pour régler donc le problème mentionné ci-dessus est le suivant. Chaque pair $p$ maintient localement pour chaque requête $q$ en cours d'exécution une liste $QPath$ contenant les adresses des pairs (adresses IP et ports) se trouvant sur le chemin partant de l'initiateur de la requête à $p$ dans l'arbre de propagation de la requête $q$. $QPath$ est une liste triée par ordre croissant des positions des pairs dans l'arbre de propagation de la requête $q$ par rapport à $p$ (le premier élément de cette liste est le parent de $p$, le second est le grand parent de $p$, etc.). Le mécanisme de construction de cette liste de pairs est la suivante. Chaque pair, y compris le pair initiateur, ajoute son adresse dans le message de requête avant de le transmettre à ses voisins. Ainsi, quand un message de requête arrive sur un pair $p$, il contient les adresses de tous les parents de $p$.
Dans la phase de remontée des résultats, quand un pair détecte que son parent (c.-à-d. le premier élément dans la liste $QPath$) est inaccessible, le pair envoie ses nouveaux résultats au pair suivant dans la liste $QPath$ qui est accessible. Un autre problème qui peut aussi arriver est qu'un pair peut quitter le système sans être en mesure d'envoyer à son parent les résultats qu'il a déjà reçu de ses fils, et cela peut avoir une conséquence grave sur la précision du résultat top-$k$ final. Pour surmonter ce problème nous adoptons la technique suivante. Durant la phase de remontée des réponses, quand un pair constate que son parent est inaccessible, il envoie son ensemble de résultats top-$k$ intermédiaire courant au pair suivant accessible dans la liste $QPath$. Bien que, cette technique soit susceptible d'augmenter le volume de données transféré dans un environnement très dynamique, il peut cependant améliorer de façon significative la précision des résultats top-$k$.

### 4.6.2 Changement de parent

Lors du calcul de la couverture locale des réponses d'un pair, nous devons tenir en compte du fait qu'un pair peut changer de parent lorsque son parent direct devient inaccessible. En effet, la non prise en compte de cette situation peut engendrer une surestimation de la valeur de la couverture locale des réponses des pairs qui peut affecter l'impact d'amélioration des réponses intermédiaires et par conséquent la réduction de l'habileté des pairs de remonter les résultats de bonne qualité le plus tôt possible. Dans cette section, nous présentons notre technique pour ajuster la valeur de la couverture locale des réponses des pairs en se basant sur la mise à jour des ensembles $\mathcal{C}_1$ et $\mathcal{C}_2$ que chaque pair $p_i$ maintient pour chaque requête top-$k$ $q$ et pour chaque fils $p_j$. Rappelons que $\mathcal{C}_1$ est l'ensemble des couples $(p_j, a)$ où $a \in \mathbb{N}$ est le nombre de pairs dans le sous-arbre du pair $p_j$ et $\mathcal{C}_2$ l'ensemble des couples $(p_j, e)$ où $e \in \mathbb{N}$ est le nombre de pairs dans le sous-arbre du pair $p_j$ qui ont déjà traité localement la requête.
Pour permettre aux pairs d'avoir une bonne estimation de leurs couvertures locales des réponses lorsque certains pairs deviennent inaccessibles, nous modifions notre approche de gestion de pannes des pairs présentée précédemment comme suit. Chaque pair $p_i$ maintient pour son parent $p_j$ et pour chaque requête active les dernières valeurs des estimations du nombre de pairs dans son sous-arbre qui ont déjà traité localement la requête et la valeur du nombre de pairs dans son sous-arbre qu'il a envoyé à son parent $p_j$. Pendant la phase de remontée des résultats, quand $p_j$ est inaccessible, $p_i$ insère dans le message de réponse qu'il va envoyer à son nouveau parent $p_k$ (le premier pair accessible dans $QPath$) les informations suivantes : (1) la nouvelle et la dernière (envoyé à $p_j$) valeurs du nombre de pairs et du nombre de pairs qui ont déjà traité localement la requête dans son sous-arbre ; (2) l'adresse du pair prédécesseur de $p_k$ dans la liste $QPath$ (ce pair est l'un des fils de $p_k$ dans l'arbre de propagation de la requête).
Quand un pair $p_k$ reçoit un message de réponse d'une requête $q$ d'un pair $p_j$ dont le parent est inaccessible, il met à jour les valeurs du nombre de pairs et du nombre de pairs qui ont déjà traité localement $q$ dans le sous-arbre de son fils direct $p_r$ qui a été déclaré "inaccessible" par $p_j$ (voir lignes 2 à 17, Algorithme 4). Ensuite $p_k$ active un déclencheur pour informer $p_r$ (quand il devient inaccessible) que $p_j$ n'est plus dans son sous-arbre (voir ligne 18, Algorithme 4).

## 4.7 MESURES DE FEEDBACK POUR LES RÉSULTATS INTERMÉDIAIRES

Bien qu'il soit important de fournir des résultats de bonne qualité le plutôt possible à l'utilisateur, il est également intéressant d'associer des "garanties probabilistes" aux résultats intermédiaires retournés à l'utilisateur. Cela permettra à l'utilisateur d'avoir une idée sur la distance qui sépare ces résultats intermédiaires des résultats finaux et ceci tout au long de l'exécution de la requête. Par exemple, on peut souhaiter être en mesure de fournir une garantie probabiliste, telle que : "l'ensemble des résultats top-$k$ courant est susceptible d'être l'ensemble des résultats top-$k$ final avec une probabilité $\gamma$". Notre objectif dans cette section est de fournir un mécanisme permettant de calculer de façon continue des garanties probabilistes au fur et à mesure que les résultats sont remontés au pair initiateur de la requête. Pour ce faire, nous proposons deux mesures de feedback qui sont retournées

**Algorithme 4** : *update_Estimation*($msg$)

```
   input  : msg,message de réponse ; C1 ; C2.
 1 begin
 2   if (change_Parent(msg) then
 3     qid = msg.getQueryID();
 4     sender = msg.getAnswerSender();
 5     a_l = getNbPeersSent(msg);
 6     e_l = getNbAnsPeersSent(msg);
 7     p = getLastPeerInacessible(msg);
 8     if (e_l > C2.get(p)) then
 9       x1 = C2.get(p);
10       y1 = C1.get(p);
11     else
12       x2 = C2.get(p) − e_l;
13       y2 = C1.get(p) − a_l;
14     end
15     C2.update(p, (x1+x2)/2);
16     y2 = C1.update(p, (y1+y2)/2);
17   end
18   propagateUpdate(queryId, Sender, p, a_l, e_l);
19 end
```

continuellement à l'utilisateur : (1) la probabilité que le top-$k$ courant soit le top-$k$ final (nous appelons cette mesure la probabilité de stabilisation), (2) la proportions des pairs dont les résultats locaux sont déjà prise en compte dans le calcul du top-$k$ courant (nous appelons cette mesure la proportion des pairs contributeurs). Dans cette section, nous présentons la manière dont ces mesures de feedback peuvent être calculées.

### 4.7.1 Probabilité de stabilisation

Pour être en mesure de calculer la probabilité de stabilisation d'une requête top-$k$ $q$ (c.-à-d. la probabilité que le top-$k$ courant de $q$ soit le top-$k$ final), il est nécessaire de connaître les paramètres suivants :
- le nombre total $L$ de pairs interrogés pour la requête $q$
- le nombre total $M$ de données partagées par les $L$ pairs
- le nombre $l$ de pairs dont les données ont été pris en compte dans le calcul du top-$k$ courant de $q$
- le nombre total $m$ de données partagées par les $l$ pairs

Dans la Section 4.4.2, nous avons présenté comment estimer le paramètre $L$ et comment calculer $l$. En ce qui concerne le calcul de $m$, il peut être effectué en incluant dans chaque message de réponse qu'un pair envoie à son parent le nombre de données qui ont été pris en compte pour le calcul de cette réponse. Cependant, pour être en mesure d'estimer $M$ il est nécessaire de connaître le nombre $l'$ de pairs que le pair initiateur a déjà pris connaissance à travers ses fils et le nombre $m'$ de données de ces $l$ pairs. Le mécanisme permettant de calculer $l'$ et $m'$ fonctionne comme suit. Chaque pair $p_i$ maintient pour chaque fils $P_j$ un ensemble $\mathcal{C}_3$ de couples $(p_j, c, d)$ où $c$ est le nombre de pairs dans le sous-arbre de $p_j$ que $p_i$ connaît à travers $p_j$ et $d$ le nombre de données partagées par les $c$ pairs. Au début de l'exécution d'une requête, chaque pair du système initialise $c = 0$ et $d = 0$. Pendant la phase de remontée des réponses, quand un pair $p_j \in \psi(p_i, q)$ veut envoyer des résultats à

$p_i$, il insère dans son message de réponses les couples de valeurs ( $\sum_{(p_j,c,d)\in\mathcal{C}_3} c$, $\sum_{(p_j,c,d)\in\mathcal{C}_3} d$). Quand $p_i$ reçoit ce message, il le décompresse, recupère ces valeurs et met à jour l'ensemble $\mathcal{C}_3$.
Le nombre total $M$ de données de tous les pairs interrogés peut être alors estimé en utilisant la Formule 4.5.

$$M = \|D(p_0)\| + m' + \frac{\|D(p_0)\| + m'}{l' + 1} * (L - l' - 1) \tag{4.5}$$

où $D(p_0)$ est le nombre de données partagées par le pair initiateur de la requête.

En supposant que la distribution des données sur les pairs est uniforme, la probabilité $P_k^m$ pour trouver les $k$-meilleurs données dans le top-$k$ courant est :

$$P_k^m = \frac{C_{M-k}^{m-k}}{C_M^m} \tag{4.6}$$

Si $l$ pairs parmi $L$ ont déjà remonté leurs résultats locaux au pair initiateur de la requête, la probabilité d'avoir $m$ données sur ces $l$ pairs est :

$$P_l^m = \begin{cases} 1 & si\ l = L \\ C_M^m \times (\frac{l}{L})^m \times (\frac{L-l}{L})^{M-m} & sinon \end{cases} \tag{4.7}$$

Sachant que $l$ pairs ont déjà remonté leurs résultats locaux vers le pair initiateur de la requête, la probabilité d'avoir au moins $k$ données est :

$$P_l^{\geq k} = \sum_{m=k}^{M} P_l^m \tag{4.8}$$

Pour trouver tous les $k$-meilleurs résultats dans ces $l$ pairs, il doit y avoir au moins $k$ données sur ces $l$ pairs et tous les meilleurs résultats (c.-à-d. les $k$-meilleurs) doivent être stockés sur ces $l$ pairs. Ainsi la probabilité d'avoir tous les résultats de l'ensemble top-$k$ final dans l'ensemble top-$k$ courant est égale :

$$P_l^{ktop} = \sum_{m=k}^{M} P_l^m \times P_k^m \tag{4.9}$$

Afin d'assurer une meilleure estimation de la probabilité que le top-$k$ courant soit le top-$k$ exact en cas de panne des pairs, nous adoptons la technique présentée à la Section 4.6 pour réajuster l'estimation de tous les paramètres utilisés dans le calcul de cette probabilité.

### 4.7.2 Proportion des pairs contributeurs

La proportion des pairs contributeurs d'un ensemble de résultats top-$k$ courant est le ratio entre le nombre de pairs interrogés dont les résultats locaux ont été déjà considéré dans le calcul du top-$k$ courant et le nombre total de pairs interrogés. La valeur de cette proportion est égale à l'estimation de la couverture locale des réponses du pair initiateur de la requête présentée dans la Section 4.4. Ainsi, nous retournons continuellement cette dernière à l'utilisateur comme proportion des pairs contributeurs.

### 4.7.3 Discussion

Dans certains cas, il peut arriver qu'un système P2P non-structuré soit configuré de sorte que les requêtes émises atteignent tous les pairs du système (par exemple en utilisant un *ttl* très élevé). Dans ce cas, un moyen efficace d'estimer le nombre de pairs interrogés (c.-à-d. la taille du réseau) est d'utilisé la méthode *gossip-based aggregation* [JM04] (aggrégation basée sur la propagation par rumeurs). Cette approche repose sur l'assertion suivante : si exactement un seul nœud du système est titulaire d'une valeur égale à 1, et toutes les autres valeurs sont égales à 0, la moyenne est $\frac{1}{N}$. Ainsi la taille du système peut donc être directement calculé. Pour exécuter cet algorithme, un initiateur doit prendre la valeur 1, et commencer le gossiping ; les nœuds qui sont atteints par ce gossiping participent au processus en mettant leur valeur à 0. A chaque cycle prédéfini, chaque nœud du réseau choisit au hasard un de ses voisins avec lequel il échange ses estimations (la taille du réseau). Le nœud contacté fait la même chose. Les deux nœuds calculent leurs nouvelles estimations alors comme suit :

$$Estimation = \frac{Estimation + Neighbor's\_Estimation}{2}$$

En s'appuyant sur l'approche *gossip-based aggregation* on peut aussi donc estimer le nombre total de données partagées par tous les pairs du système.
Notons que pour fournir des estimations correctes, cet algorithme doit attendre une certaine période de temps avant de calculer l'estimation de la taille du réseau ; cette période est le temps requis pour que le gossip ou la rumeur se propage vers tous les pairs du réseau et pour que les valeurs convergent. Rappelons que cette méthode converge vers la valeur exacte dans un environnement stable tel que démontré dans [JM04]. Rappelons aussi qu'il a été démontré dans [KDG03] que les protocoles de gossip convergent de façon exponentielle avec un coût de communication faible .

## 4.8 ÉVALUATION EXPÉRIMENTALE

Dans cette section, nous évaluons la performance d'ASAP par simulation en utilisant le simulateur PeerSim [JMJV]. Cette section est organisée comme suit. Tout d'abord, nous décrivons notre environnement de simulation, les paramètres utilisés pour l'évaluation de la performance. Ensuite, nous étudions l'impact du nombre de pairs et le nombre de résultats souhaité par l'utilisateur sur la performance d'ASAP, et ainsi nous montrons comment notre approche passe à l'échelle. Puis, nous étudions l'effet du nombre de répliques sur la performance d'ASAP. Enfin, nous étudions l'effet des pannes des pairs sur la précision des résultats retournés par ASAP.

### 4.8.1 Environnement de simulation

Nous avons implémenté notre simulation en utilisant le simulateur PeerSim. PeerSim est un framework libre, écrit en java et destiné à simuler le mode de fonctionnement des réseaux P2P. Il permet donc de concevoir et de tester toute sorte d'algorithme des réseaux P2P dans un environnement dynamique. Ce framework supporte deux types de simulation qui sont :

– La simulation par cycle : la simulation s'effectue dans un ordre séquentiel. Dans chaque cycle, chaque protocole peut exécuter son comportement.
– La simulation par évènement : elle supporte la concurrence ; un ensemble d'évènements est planifié et les protocoles d'un nœud sont exécutés en fonction des évènements survenus.

Nous effectuons notre expérimentation sur une machine Intel Pentium 2,4 GHz et ayant 2 Go de mémoire. Le Tableau 5.1 décrit les paramètres de notre simulation. Nous utilisons les valeurs des paramètres qui sont typiques aux systèmes P2P [GSG02]. Le temps de latence entre deux pairs est un nombre aléatoire normalement distribué avec une moyenne de 200 ms. Étant donné que les utilisateurs sont généralement intéressés par un petit nombre de résultats, nous mettons $k = 20$ comme valeur par défaut. Dans nos expériences nous varions la taille du réseau de 1000 à 10000 pairs. Dans le but de simuler l'hétérogénéité des pairs nous attributions des capacités aux pairs du système conformément aux résultats présentés dans [GSG02]. Ce travail mesure la capacité des pairs dans le système Gnutella. Sur la base de ces résultats, nous générons environ 10% de pairs faiblement capables, 60% de pairs moyennement capables, et 30% de pairs énormément capables. Les pairs énormément capables sont 3 fois plus capables que les pairs moyennement capables et 7 fois plus capables que les pairs faiblement capables.

Dans notre expérimentation, nous réalisons 30 tests pour chaque expérience et nous reportons la moyenne des résultats observés. Nous présentons les résultats des approches d'ASAP basées sur un seuil dynamique désigné par respectivement ASAP-Dscore et ASAP-Drank. ASAP-Dscore utilise un impact d'amélioration basé sur le score des résultats intermédiaires et ASAP-Drank utilise un impact d'amélioration basé sur le rang des résultats intermédiaires. Les approches d'ASAP faisant usage d'un seuil dynamique se sont révélées meilleures que celles basées sur un seuil statique sans être coûteux en terme de coût de communication. Dans toutes nos expériences, nous nous utilisons $H(x) = -0.2x + 0.2$ comme fonction de seuil dynamique et 0 comme seuil de couverture locale des réponses des pairs pour l'approche ASAP-Dscore. Dans le cas de Asap-Drank, nous utilisons $H(x) = -0.5x + 0.5$ comme fonction de seuil et 0.05 comme seuil de couverture locale des réponses des pairs.

#### 4.8.1.1 Jeux de données

Dans le cadre de nos simulations chaque pair de notre système P2P a une table $\mathcal{R}(data)$ dans lequel l'attribut $data$ est un nombre réelle. Le nombre de lignes $\mathcal{R}$ sur chaque pair est un nombre aléatoire uniformément répartie sur tous les pairs du système. Ce nombre est plus grand que 1000 et plus petit que 20000. Sauf indication contraire, nous supposons qu'il n'y a qu'une seule copie pour chaque donnée présente dans le système (c.-à-d. il n'y a pas de réplication des données). Nous assurons aussi qu'il n'y a pas des données différentes qui ont le même score. Dans tous nos tests, nous utilisons comme charge de travail la requête suivante, que nous notons $q_{load}$ :

`SELECT` $val$ `FROM` $\mathcal{R}$ `ORDER BY` $F(\mathcal{R}.data, val)$ `STOP AFTER` $k$

Le score $F(\mathcal{R}.data, val)$ est calculé comme suit :

$$\frac{1}{1 + |\mathcal{R}.data - val|}$$

Tableau 4.1 **Les paramètres de simulation**

| **Paramètres** | **Valeurs par défaut** |
|---|---|
| Temps de latence | Nombre aléatoire uniformément distribué, Moy=200 ms, Var=100 |
| Nombre de pairs | 10000 |
| Degré moyen des pairs | 4 |
| *ttl* | 9 |
| $k$ | 20 |
| Nombre de répliques | 1 |

#### 4.8.1.2 Approches de base

Dans notre simulation, nous comparons ASAP par rapport à *Fully Distributed* (FD) [APV06], une approche de base pour le traitement des requêtes top-$k$ dans les systèmes P2P non-structurés, qui fonctionne comme suit. Chaque pair qui reçoit une requête, l'exécute localement (c.-à-d. sélectionne les $k$ meilleurs scores), et attend les résultats de ses fils. Après avoir reçu tous les score-listes de ses fils, les pairs fusionnent leurs résultats locaux avec ceux reçus de leurs fils, sélectionne les $k$ meilleurs et envoient le résultat à leurs parents.

#### 4.8.1.3 Métriques

Dans nos expériences, pour évaluer la performance d'ASAP par rapport à FD, nous utilisons les métriques suivants :

(i) **Qualité restante cumulée :** Comme défini à la Section 5.2, c'est la somme des différences de qualité entre les ensembles des résultats top-$k$ intermédiaires reçus par l'utilisateur jusqu'au temps de stabilisation et l'ensemble des résultats top-$k$ final.

(ii) **Temps de stabilisation :** Le temps de stabilisation est le temps d'obtention de tous les $k$ meilleurs résultats.

(iii) **temps de réponse :** Le temps de réponse est le temps que doit atteindre l'utilisateur pour que l'exécution d'une requête top-$k$ prenne fin.

(iv) **Coût de communication :** Nous mesurons le coût de communication en termes de nombre de messages de réponse et de volume de données transféré sur le réseau pour le traitement d'une requête top-$k$.

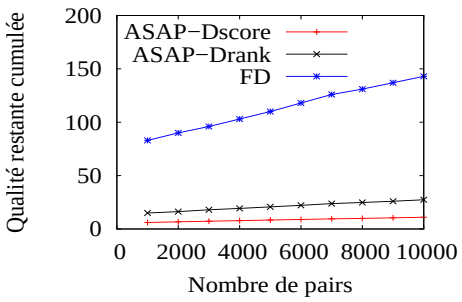
(v) **Précision des résultats :** Nous définissons la précision des résultats comme suit. Étant donnée une requête top-$k$ $q$, soit $V$ l'ensemble des $k$ meilleurs résultats dont disposent tous les pairs ayant reçu $q$, et soit $V'$ l'ensemble des résultats top-$k$ qui est retourné à l'utilisateur comme réponse de la requête $q$ à la fin de l'exécution de $q$. Nous notons la précision par $ac_q$ et nous la définissons comme suit :

$$ac_q = \frac{\|V \cap V'\|}{\|V\|}$$

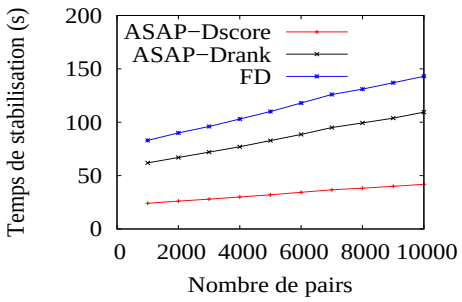
(iv) **Le nombre total de résultats :** Nous mesurons le nombre total de résultats comme le nombre de résultats reçu par le pair initiateur d'une requête durant l'exécution d'une requête.

### 4.8.2 Résultats et analyses des performances

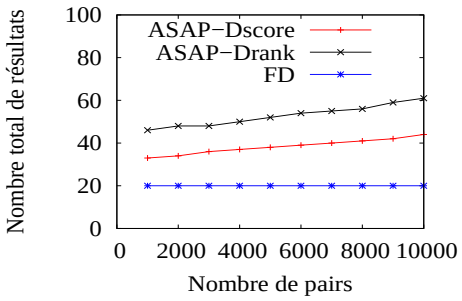
Dans cette section, nous présentons et analysons les résultats de notre expérimentation.



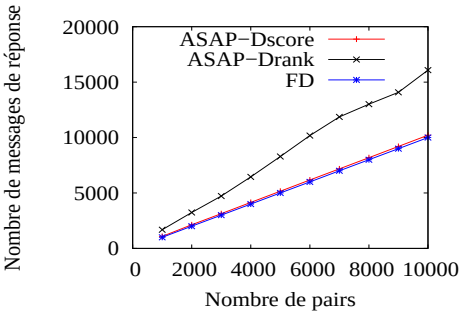

(a) Qualité restante cumulée vs. Nombre de pairs



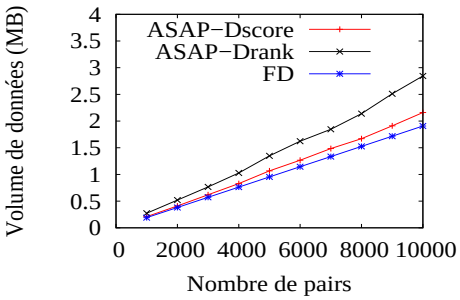

(b) Temps de stabilisation vs. Nombre de pairs



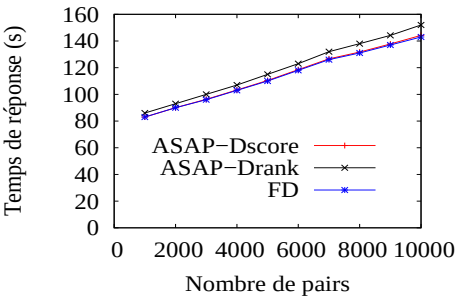

(c) Nombre total de résultats vs. Nombre de pairs




(d) Nombre de messages de réponse vs. Nombre de pairs




(e) Volume de données transféré vs. Nombre de pairs




(f) Temps de réponse vs. Nombre de pairs

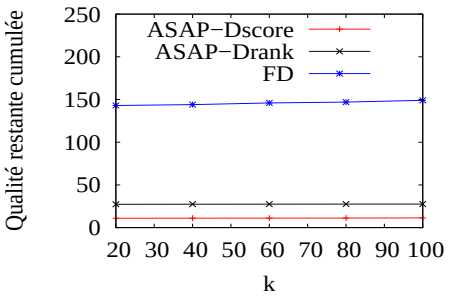
Figure 4.2 **Impact du nombre de pairs sur la performance d'ASAP**

#### 4.8.2.1 Effet du nombre de pairs

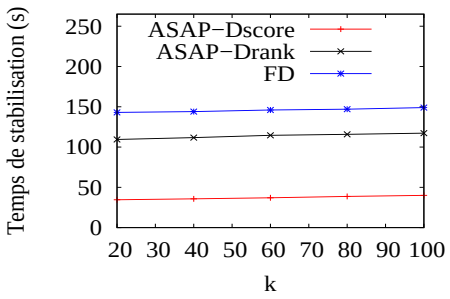
Nous étudions l'impact du nombre de pairs sur la performance d'ASAP. Pour cela, nous menons des expériences pour étudier la façon dont la qualité restante cumulée, le temps de stabilisation, le nombre de messages de réponse, le volume de données transféré, le nombre de résultats intermédiaires et le temps de réponse évoluent avec l'augmentation du nombre de pairs du système. Notons que les autres paramètres de la simulation sont définis comme dans le Tableau 5.1.
Figure 4.2(a) et 4.2(b) montrent respectivement comment la qualité restante cumulée et le temps de stabilisation évoluent avec le nombre de pairs. Les résultats montrent que les qualités restantes cumulées d'ASAP-Dscore et d'ASAP-Drank sont toujours beaucoup plus petites que celle de FD, ce qui signifie que ASAP renvoie des résultats de bonne qualité plus rapidement que FD. Les résultats montrent également que le temps de stabilisation d'ASAP-Dscore est toujours beaucoup plus petit que celui d'ASAP-Drank et celui de FD. La raison est que ASAP-Dscore est sensible aux scores c.-à-d. à l'apparition des résultats de grande qualité, ainsi l'ensemble des résultats top-$k$ final est retourné rapidement.
Figure 4.2(c) montre que le nombre total de résultats reçu par l'utilisateur augmente avec le nombre de pairs dans le cas de l'ASAP-Dscore et ASAP-Drank pendant que ce nombre est toujours constant dans le cas de FD. Cela est dû au fait que FD ne retourne pas de résultats intermédiaires aux utilisateurs. Les résultats montrent également que le nombre de résultats reçu par l'utilisateur dans le cas d'ASAP-Dscore est inférieure à celui d'ASAP-Drank. La raison principale est que ASAP-Dscore est sensible aux scores à la différence d'ASAP-Drank.
Figure 4.2(d) et Figure 4.2(e) montrent que le nombre de messages de réponse et le volume de données transféré augmentent avec le nombre de pairs. Les résultats montrent aussi que le nombre de messages de réponse et le volume de données transféré d'ASAP-Drank sont toujours plus élevés que ceux d'ASAP-Dscore et de FD. Les résultats montrent également que les différences entre ASAP-Dscore et FD en termes de nombre de messages de réponse et le volume de données transféré ne sont pas significatives. La raison principale est que ASAP-Dscore est sensible aux scores à la différence d'ASAP-Drank. Ainsi, seul les résultats de qualité élevée sont remontés rapidement à l'initiateur de la requête.
Figure 4.2(f) montre comment le temps de réponse évolue quand le nombre de pairs augmente. Les résultats montrent que la différence entre le temps de réponse d'ASAP-Dscore et celui de FD n'est pas significative. Les résultats montrent également que la différence entre le temps de réponse d'ASAP-Drank et celui de FD augmente légèrement en faveur d'ASAP-Drank quand le nombre de pairs augmente. La raison en est que ASAP-Drank induit davantage de trafic réseau que ASAP-Dscore et FD ce qui a un impact sur le temps de réponse.

#### 4.8.2.2 Effet de $k$

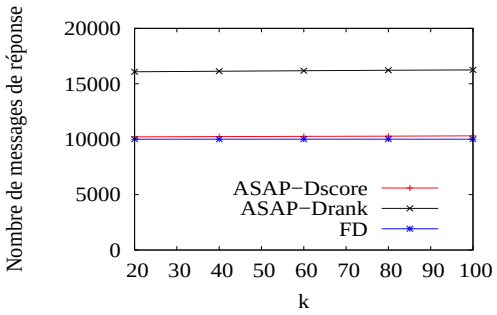
Nous étudions dans cette partie l'impact de $k$, c.-à-d. le nombre de résultats demandé par l'utilisateur, sur la performance d'ASAP. En utilisant notre simulateur, nous étudions comment la qualité restante cumulée, le temps de stabilisation et le volume de données transféré évoluent quand $k$ varie de 20 à 100, avec les autres paramètres de la simulation définis comme indiqué dans le Tableau 5.1. Les résultats (voir Figure 4.3(a), Figure 4.3(b)) montrent que $k$ a un faible impact sur la qualité restante cumulée et le temps de stabili-



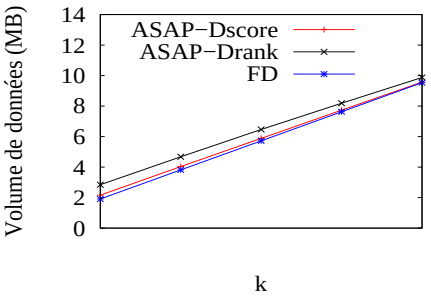

(a) Qualité restante cumulée vs. $k$




(b) Temps de stabilisation vs. $k$

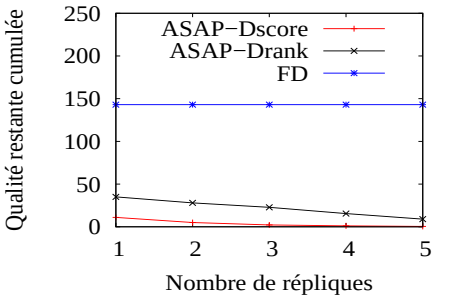


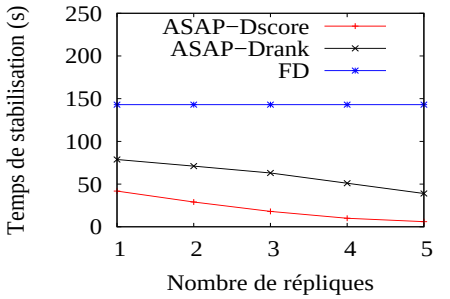

(c) Nombre de messages de réponse vs. $k$.




(d) Volume de données transféré vs. $k$

Figure 4.3 **Impact de $k$ sur la performance d'ASAP**




(a) Qualité restante cumulée vs. Nombre de répliques

(b) Temps de stabilisation vs. Nombre de répliques

Figure 4.4 **Impact de la réplication des données sur la performance d'ASAP**

sation d'ASAP-Dscore et d'ASAP-Drank. Les résultats (voir la Figure 4.3(d) montrent également que quand $k$ augmente, le volume de données transféré d'ASAP-Dscore et d'ASAP-Drank augmente moins que celui de FD. Cela est dû au fait que ASAP-Dscore et ASAP-Drank élimine plus de résultats intermédiaires quand $k$ augmente.

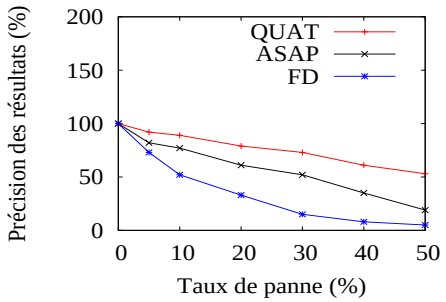


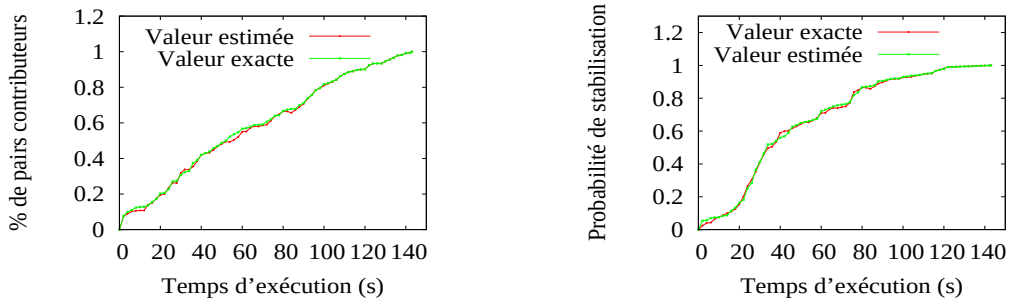

Figure 4.5 **Précision des résultats vs. Taux de panne**

#### 4.8.2.3 Effet de la réplication des données

La réplication est largement utilisée dans les systèmes P2P non-structurés pour améliorer la recherche ou assurer la disponibilité des données. Par exemple, les systèmes non-structurés comme BubbleStorm [TKLB07] font usage d'un grand nombre de répliques de chaque objet stocké dans le système pour améliorer leurs algorithmes de recherche.
Dans cette section, nous étudions l'effet du nombre de répliques des données sur la performance d'ASAP. Pour cela nous faisons usage de la stratégie de réplication uniforme [FJC+07]). En utilisant notre simulateur, nous étudions comment la qualité restante cumulée et le temps de stabilisation évoluent quand le nombre de répliques augmente, avec les autres paramètres de la simulations définis comme indiqué dans le Tableau 5.1. Les résultats (voir Figure 4.4(a) et Figure 4.4(b)) montrent que l'augmentation du nombre de répliques d'ASAP et de FD fait décroître la qualité restante cumulée et le temps de stabilisation d'ASAP-Dscore et d'ASAP-Drank. Cependant, la qualité restante cumulée et le temps de stabilisation de FD restent toujours constants. La raison est que ASAP retourne plus rapidement les résultats de grande qualité contrairement à FD qui retourne les résultats uniquement à la fin de l'exécution de la requête. Ainsi, en augmentant le nombre de répliques, ASAP retrouve plus rapidement les résultats ayant des scores élevés.

#### 4.8.2.4 Efficacité de notre solution de calcul de "garanties probabilistes"

Dans cette section, nous étudions l'efficacité de la solution proposée dans la Section 4.7 pour le calcul des garanties probabilistes, en les comparant avec les valeurs optimales. Pour cela, nous menons des expériences pour étudier la façon dont les valeurs de nos garanties probabilistes évoluent comparativement aux valeurs optimales (c.-à-d. des valeurs exactes) au cours de l'exécution des requêtes dans le cas d'ASAP-Dscore. Figure 4.6(a) et Figure 4.6(b) montrent que la différence entre les valeurs de nos garanties probabilistes et leurs valeurs exactes est très faible. Les résultats montrent également que les valeurs de nos garanties probabilistes convergent vers les valeurs exactes et ceci avant la fin de l'exécution de la requête. Cela signifie que notre solution offre des garanties probabilistes fiables pour l'utilisateur sur des résultats intermédiaires.




(a) Proportion des pairs contributeurs vs. Temps d'exécution (s)

(b) Probabilité de stabilisation vs. Temps d'exécution (s)

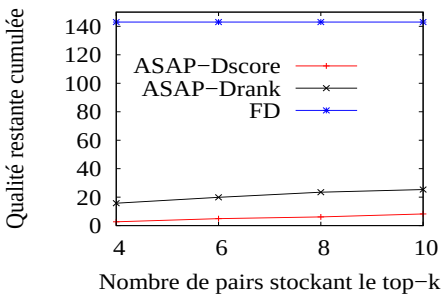
Figure 4.6 **Efficacité de notre mécanisme de calcul de garanties probabilistes**

#### 4.8.2.5 Effet de la distribution des données

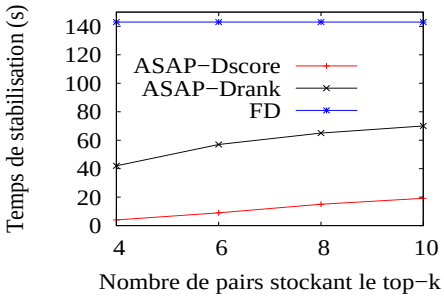
Une étude sur la distribution des données est essentielle pour les algorithmes de recherche dans les systèmes P2P, car la distribution des données n'est pas toujours uniforme. Il est fréquent que les données pertinentes soient regroupées ensemble et stockés sur un groupe de pairs voisins. Si ces groupes de pairs possèdent plusieurs données pouvant être dans l'ensemble des résultats des requêtes top-$k$, ils constituent alors une région idéale du réseau qui peut beaucoup contribuer à un ensemble de résultats top-$k$ final. Pour étudier l'effet de la distribution des données sur la performance d'ASAP, nous distribuons aléatoirement les $k$ meilleurs données des requêtes que nous utilisons comme banc d'essai sur respectivement 4, 6, 8 et 10 pairs du système P2P et les autres données (c.-à-d. ceux qui ne sont pas dans le top-$k$) de façon uniforme sur tous les pairs du système. En utilisant notre simulateur, nous étudions comment la qualité restante cumulée, le temps de stabilisation et le volume de données transféré évoluent quand respectivement seulement 4, 6, 8 et 10 pairs système P2P stockent les résultats top-$k$ avec les autres paramètres de simulation définis comme indiqué dans le Tableau 5.1. Les résultats (voir Figure 4.7(a) et Figure 4.7(b)) montrent que ASAP peut tirer profit de la distribution des données pour fournir des résultats de bonne qualité plus rapidement aux utilisateurs, contrairement à FD. Les résultats (voir Figure 4.7(c)) montrent également que dans le cas d'ASAP, plus les données faisant partie du top-$k$ sont regroupées, le plus petit est le volume de données transféré sur le réseau, tandis que dans le cas de FD ce volume reste constant.

#### 4.8.2.6 Effet de la panne des pairs

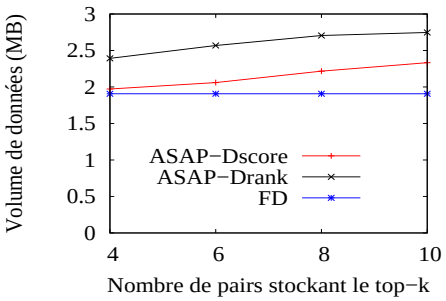
Dans cette section, nous étudions l'effet des pannes des pairs sur la précision de l'ensemble des résultats top-$k$ final. Dans nos tests, nous varions la valeur du taux de panne et observons son effet sur la précision des résultats top-$k$. Figure 4.5 montre que la précision des résultats top-$k$ d'ASAP-Dscore, ASAP-Drank et FD quand nous augmentons le taux de panne, et les autres paramètres de la simulation définis comme indiqué dans le Tableau 5.1. Les pannes des pairs ont moins d'impact sur ASAP-Dscore et ASAP-Drank que sur FD. La raison est que ASAP-Dscore et ASAP-Drank retournent les résultats de bonne qualité




(a) Qualité restante cumulée gap vs. Nombre de pairs stockant les résultats top-$k$




(b) Temps de stabilisation vs. Nombre de pairs stockant les résultats top-$k$




(c) Volume de données transféré vs. Nombre de pairs stockant les résultats top-$k$

Figure 4.7 **Impact de la distribution des données sur la performance d'ASAP**

le plus tôt possible à l'utilisateur. Cependant, l'augmentation du taux de pannes dans le cas de FD engendre une diminution significative de la précision des résultats top-$k$. La raison principale est que dans FD, chaque pair attend les résultats de tous ses fils ainsi en cas de panne d'un pair, tous les scores-listes déjà reçus par ce pair sont perdus.

## 4.9 CONCLUSION

Dans ce chapitre, nous avons présenté une nouvelle approche de traitement des requêtes top-$k$ d'une manière ASAP (le plutôt possible) dans les systèmes P2P. Nous avons proposé une définition formelle de ce qu'est le problème du traitement ASAP d'une requête top-$k$ grâce à de nouvelles métriques qui sont le temps de stabilisation et la qualité restante cumulée. Nous avons ensuite présenté ASAP, une nouvelle famille d'algorithmes basée sur une technique de seuil qui considère le score et le rang des résultats intermédiaires afin de retourner les résultats de bonne qualité le plus rapidement aux utilisateurs. Nous avons validé ASAP à travers une implémentation et une évaluation expérimentale très poussée. Les résultats ont montré que ASAP surpasse de manière significative les algorithmes de base en retournant le résultat top-$k$ final nettement aux utilisateurs. Enfin, les résultats

ont montré que en cas de pannes de pairs, ASAP fournit des résultats top-$k$ approximatifs avec une précision meilleure à celle des algorithmes de base.

# CHAPITRE 5
# TRAITEMENT DES REQUÊTES TOP-$k$ DANS LES SYSTÈMES P2P SURCHARGÉS

Dans le chapitre précédent nous avons proposé ASAP une famille d'algorithmes permettant à un utilisateur d'avoir les résultats de bonne qualité le plus tôt possible sans être pénalisé par le pair le plus lent du système. Bien que ASAP résout ce problème, il n'est pas cependant adapté aux systèmes surchargés dans lesquels les pairs peuvent recevoir un très grand nombre de requêtes utilisateurs dans un court laps de temps. Pour cela les approches de traitement des requêtes top-$k$ existantes ne sont pas adaptées pour de nombreuses applications P2P populaires, telles que les moteurs de recherche P2P, le partage de données P2P pour les communautés en ligne, car ces systèmes sont souvent exposées à un grand nombre de requêtes et peuvent donc facilement être surchargées.
Dans ce chapitre, nous adressons le problème de réduction des temps d'attente des utilisateurs lors du traitement des requêtes top-$k$ dans le contexte des systèmes P2P surchargés. Pour cela nous nous appuyons sur les nouvelles métriques que nous avons présenté dans le Chapitre 3 (*temps de stabilisation* et *qualité restante cumulée*) pour définir formellement ce problème. Ensuite, pour résoudre ce problème, nous proposons QUAT [1] une approche qui prend en compte les charges des pairs et qui fait usage des descriptions synthétiques des pairs pour permettre à l'utilisateur d'avoir les résultats top-$k$ le plus tôt possible, sans attendre le résultat top-$k$ final. Enfin, nous évaluons expérimentalement cette approche. Les résultats obtenus montrent que les performances de QUAT en termes de temps de stabilisation, de qualité restante cumulée et de temps de réponse sont bien meilleures que celles d'ASAP et de FD en présence de forte charge de requêtes.

## 5.1 MODÈLE DU SYSTÈME

Dans cette section, nous présentons le modèle général des systèmes P2P non-structurés qui est nécessaire pour décrire notre solution. Ensuite, nous proposons un modèle et des définitions pour les requêtes top-$k$.

1. Quality-based Early Top-$k$ Query Processing refers to Khat, an African plant whose leaves are chewed as a stimulant.

### 5.1.1 Modèle de réseau P2P non-structuré

Nous reconsidérons notre modèle de réseau P2P non-structuré présenté au Chapitre 4. Nous définissons la capacité $c_i$ d'un pair $p_i$ comme étant le nombre de requêtes que $p_i$ peut traiter en une unité de temps. Si un pair reçoit de requêtes de ses voisins à un taux supérieur à ce qu'il est est capable de traiter, alors ces requêtes sont mises en attente dans une queue jusqu'à ce qu'il les traitent (traitement locale et propagation vers les autres voisins). Notons que généralement le nombre maximal de connexions (canaux de communication) qu'un pair peut maintenir simultanément avec ses voisins est proportionnel à sa capacité. Toutefois, un pair peut fixer ce nombre à une valeur inférieure à la valeur maximale qu'il peut maintenir s'il le souhaite.

### 5.1.2 Requêtes top-$k$

Le modèle de requête top-$k$ que nous considérons dans ce chapitre est identique à celui présenté dans le Chapitre 4.

### 5.1.3 Description synthétique du contenu des pairs

Dans notre système, chaque pair est caractérisé par une description synthétique des données qu'il partage dans le système. La construction de cette description synthétique est hors de portée de cette thèse. Nous supposons donc qu'elle est obtenue grâce à une fonction d'agrégation de description qui prend en entrée un ensemble de données et génère une seule description synthétique pour toutes ces données comme par exemple dans [HRVM08] et [VLCV09b]. Nous définissons formellement une fonction d'agrégation comme suit.

**Définition 10.** ***Fonction d'agrégation.*** *Soient $\mathcal{D}$ l'ensemble des éléments de données, et $\mathcal{R}^{\mathcal{D}}$ l'ensemble des descriptions. Une fonction, notée $agr$, est une fonction d'agrégation si et seulement si elle a la signature suivante :*

$$agr : \left| \begin{array}{lcl} \mathcal{D} & \rightarrow & \mathcal{R}^{\mathcal{D}} \\ \\ \{d_1, \cdots, d_m\} & \mapsto & agr(\{d_1, \cdots, d_m\}) \end{array} \right.$$

Pour éviter une reconstruction de la description des données d'un pair à partir de tous ses éléments de données quand celui-ci ajoute une nouvelle donnée dans sa base de données locale, il est judicieux que la fonction d'agrégation soit incrémentale. Nous définissons une fonction d'agrégation incrémentale comme suit.

**Définition 11.** ***Fonction d'agrégation incrémentale.*** *Soient $\{d_1, \cdots, d_m\} \subseteq \mathcal{D}$ un ensemble des éléments de données, et $d \in \mathcal{D}$ un élément de donnée. Une fonction d'agrégation, notée $agr$, est une fonction d'agrégation incrémentale si et seulement si il existe une fonction*

$$add : \left| \begin{array}{lcl} \mathcal{R}^{\mathcal{D}} \times \mathcal{D} & \rightarrow & \mathcal{R}^{\mathcal{D}} \end{array} \right.$$

*telle que $add(agr(\{d_1, \cdots, d_m\}), \{d\}) = agr(\{d_1, \cdots, d_m, d\})$*

Pour permettre aussi d'obtenir simplement la description de deux ou plusieurs pairs sans passer par leurs ensembles de données, il est nécessaire que la fonction d'agrégation soit composable. Nous définissons une fonction d'agrégation composable comme suit.

**Définition 12.** ***Fonction d'agrégation composable.*** *Soient* $\{d_1^1, \cdots, d_m^1\} \subseteq \mathcal{D}$ *et* $\{d_1^2, \cdots, d_{m'}^2\} \subseteq \mathcal{D}$ *deux ensembles d'éléments de données. Une fonction d'agrégation, notée agr, est une fonction d'agrégation composable si et seulement si il existe une fonction*

$$merge : \left| \begin{array}{rcl} \mathcal{R}^{\mathcal{D}} \times \mathcal{D} & \rightarrow & \mathcal{R}^{\mathcal{D}} \end{array} \right.$$

*telle que* $merge(agr(\{d_1^1, \cdots, d_m^1\}), agr(\{d_1^2, \cdots, d_{m'}^2\}),) = agr(\{d_1^1, \cdots, d_m^1\} \cup \{d_1^1, \cdots, d_m^1\})$

Il est à noter que pour construire une description synthétique à partir de deux ou plusieurs pairs ayant des schémas différents, nous pouvons utiliser l'une des techniques de mapping de schémas de façon décentralisée que nous avons présenté dans l'état l'art, par exemple [TIM+03] ou [AMPV06a].

Pour être en mesure de se baser sur la description d'un pair pour décider d'interroger ou non un pair, il est nécessaire que l'estimation de la qualité des résultats d'une requête top-$k$ par rapport à une description synthétique ne soit pas inférieure aux scores des données qui ont été prises en compte dans la construction de cette description synthétique. Cela signifie que la fonction d'agrégation doit être optimiste. Nous définissons une fonction d'agrégation optimiste comme suit.

**Définition 13.** ***Fonction d'agrégation optimiste.*** *Soient* $\{d_1, \cdots, d_m\} \subseteq \mathcal{D}$ *un ensemble d'éléments de données et q une requête top-k. une fonction d'agrégation, notée agr, est une fonction d'agrégation optimiste par rapport la fonction de score de q si et seulement si :*

$$\forall d_i \in \{d_1, \cdots, d_m\}, q.f(aggr(\{d_1, \cdots, d_m\}), q.c) \geq q.f(d_i, q.c)$$

Dans la suite de ce chapitre nous supposons que la fonction d'agrégation utilisée par les pairs pour construire leurs descriptions synthétiques est : incrémentale, composable et optimiste. Notons qu'il existe dans la littérature des descriptions qui satisfont ces hypothèses : c'est le cas par exemple des descriptions sémantiques proposées dans [VLCV09b].

Pour illustrer l'une des propriétés clé des descriptions synthétiques à savoir la propriété optimisme, nous considérons une table relationnelle contenant les notes des étudiants au contrôle continu et à l'examen comme présenté dans la Figure 5.1. Une description synthétique de cette table pourrait par exemple être constituée uniquement des notes maximum au partiel et à l'examen. Supposons maintenant que nous voulons avoir la liste des 5 premiers étudiants c.-à-d. les 5 étudiants qui ont les moyennes les plus élevées. Dans ce cas la fonction de score à utiliser est donc $F(x, y) = \frac{x+y}{2}$ où $x$ est la note du partiel et $y$ la note de l'examen. La description nous permet de savoir que la plus grande moyenne qu'on peut obtenir ne peut pas dépasser en $\frac{16+16.5}{2}$ soit 16.25. La description nous indique donc que nous ne pouvons pas espérer avoir un étudiant ayant plus de 16.25 (la propriété optimiste). Cette propriété est intéressante dans un système P2P surchargé car elle peut être utilisée pour la priorisation des requêtes à traiter.

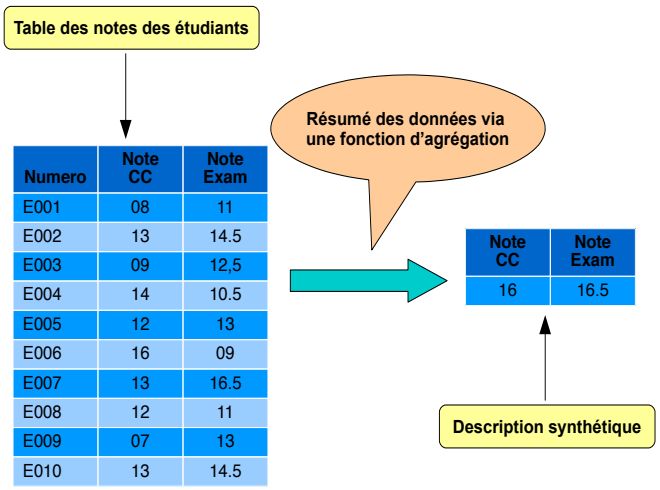



Figure 5.1 **Exemple de description synthétique de données d'une table**

## 5.2 DÉFINITION DU PROBLÈME

### 5.2.1 Préliminaires

Dans ce chapitre, nous adressons le problème du traitement des requêtes top-$k$ dans les systèmes P2P surchargés où les pairs peuvent recevoir un très grand nombre de requêtes dans un court laps de temps. Pour cela, nous définissons *le temps de stabilisation* et la *qualité restante cumulée pour une période de temps*, et notre objectif est de développer des algorithmes efficaces du point de vue de ces mesures. Le temps de stabilisation et de la qualité restante cumulée pour une periode de temps $T$ sont respectivement la moyenne du temps de stabilisation et la moyenne de la qualité restante cumulée pour toutes requêtes émises pendant la période de temps $T$. Nous formalisons cela dans les Définitions 14 et 15.

**Définition 14.** ***Temps de stabilisation sur une période de temps***. *Étant donné $Q$, un ensemble de requêtes top-k émises durant une période $T$, le temps de stabilisation sur la période $T$, noté $S_T$ est :*

$$S_T = \frac{\sum_{q \in Q} s(q)}{\|Q\|} \tag{5.1}$$

*où $s(q)$ est le temps de stabilisation de $q \in Q$.*

**Définition 15.** ***Qualité restante cumulée sur une période de temps***. *Étant donné $Q$, un ensemble de requêtes top-k émises durant une période $T$, la qualité restante cumulée sur la période $T$, notée par $C_{qgT}$ est :*

$$C_{qgT} = \frac{\sum_{q \in Q} c_{qg}(q)}{\|Q\|} \tag{5.2}$$

*où $c_{qg}(q)$ est la qualité restante cumulée de $q \in Q$.*

### 5.2.2 Énoncé du problème

Étant donnée une période de temps $T$, soient $S_T$ et $C_{qgT}$ respectivement le temps de stabilisation et la qualité restante cumulée sur la période $T$. Notre objectif est de réduire $S_T$ et $C_{qgT}$ tout en fournissant des ensembles de résultats top-$k$ corrects et tout en évitant l'effondrement du système.

## 5.3 TRAITEMENT DES REQUÊTES TOP-$k$ AVEC QUAT

Dans QUAT, chaque pair maintient une description de ses données locales et des descriptions des données de son voisinage (c.-à-d. les descriptions des données de ses voisins directs et les descriptions des données des voisins directs de ses voisins). Ces descriptions sont utilisées pour créer des indices de routage pour le traitement des requêtes top-$k$. Nous donnons plus de détails sur la construction et la maintenance de ces indices de routage dans la Section 5.4. Comme dans le cas de ASAP presenté dans le Chapitre 4, le traitement des requêtes top-$k$ se fait en deux phases principales. La première phase est la phase de transmission de la requête et son exécution local par les pairs et la deuxième phase est la remontée des résultats des pairs vers le pair initiateur via l'arbre de propagation de la requête. Dans cette section, nous présentons d'abord l'architecture d'un pair et nous montrons comment les deux phases de traitement d'une requête top-$k$ sont mise en œuvre dans QUAT. Nous terminons cette section en proposant une optimisation permettant de minimiser la consommation de ressources dans QUAT.

### 5.3.1 Architecture d'un pair dans QUAT

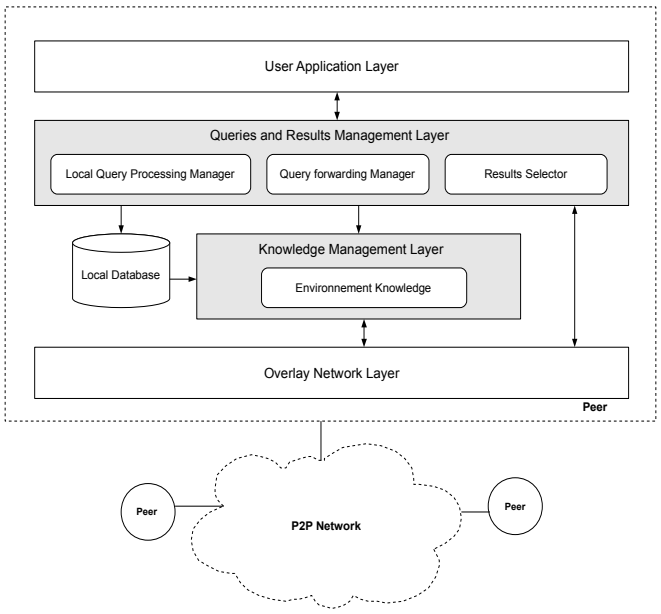



Figure 5.2 **Architecture d'un pair**

Dans cette section, nous présentons l'architecture QUAT en présentant ces différents composants. L'architecture proposée est basée sur quatre composants principaux : (1) une interface de soumission des requêtes et d'affichage des résultats (User Application Layer), (2) une couche de gestion des requêtes et des résultats (Queries and Results Management Layer), (3) une couche de gestion de la connaissances du pair (Knowledge Management Layer), et (4) une couche de réseau overlay (Overlay Network Layer) qui fournie un service de communication avec la couche de gestion des requêtes et des résultats et aussi avec la couche de gestion des connaissances du pair.

- **User Application Layer** : cette couche permet à l'utilisateur de poser ses requêtes dans le système P2P. Elle permet aussi la présentation à l'utilisateur des résultats des requêtes qu'il a soumis dans le système.
- **Queries and Results Management Layer** : cette couche propose trois services qui sont **Local Query Processing Manager**, **Query Forwarding Manager** et **Results Selector**. Le Local Query Processing Manager se charge de la priorisation des requêtes qui sont en attente et qui doivent être traitées localement par un pair. Cette priorisation est effectuée grâce à l'estimation des qualités des résultats de ces requêtes par rapport aux descriptions synthétiques du pair. Le Query Forwarding Manager gère la priorisation des requêtes qui sont en attentes pour être transmises aux pairs voisins. Le Query Forwarding Manager s'appuie sur la couche Knowledge Management Layer pour obtenir des informations sur les données partagées (c.-à-d. les descriptions synthétiques des données partagées) par les voisins pour pouvoir assigner des priorités à ces requêtes et ceci en fonction de l'estimation des qualités des résultats que ces voisins peuvent retourner. Le **Results Selector** permet au pair de pouvoir identifier parmi les résultats déjà obtenus ceux qui sont de bonne qualité pour pouvoir les renvoyer le plus tôt possible à l'utilisateur.
- **Knowledge Management Layer** : cette couche offre un service qui permet au pair de créer une table de description de son voisinage en échangeant sa propre description avec celles des voisins quand le pair entre dans le système P2P. Cette couche permet aussi de maintenir cette table de descriptions quand des modifications (niveau données, arrivée d'un nouveau nœud voisin ou sortie d'un nœud voisin) surviennent dans le voisinage du pair.
- **Overlay Network Layer** : cette couche assure la communication avec les pairs du réseau.

  Dans la section suivante nous présentons le mécanisme de traitement d'une requête top-$k$ dans QUAT.

### 5.3.2 Propagation et exécution locale des requêtes

Dans cette section, nous présentons le mécanisme de propagation et d'exécution locale des requêtes dans QUAT.

#### 5.3.2.1 Propagation des requêtes

Dans les approches classiques de propagation des requêtes top-$k$ dans les systèmes P2P comme par exemple dans [APV06], chaque pair transmet en parallèle toute requête reçue à tous ses voisins. Cependant, dans un système P2P surchargé, cette approche peut rapidement entrainer l'effondrement du système car cela exige habituellement beaucoup de

**Algorithme 5** : $do\_Forward(QF_{all}, QF_{old}, percent, mCon)$

```
   input : QF_all, liste des requêtes à traiter localement triée en fonction des estimations des qualités de réponses ;
           QP_old, liste des requêtes qui sont qualifiées de vieilles ; percent, la proportion de connexions disponible à
           allouer aux requêtes de la liste QF_all.
 1 begin
 2   while (getAvailableCon() > 0) do
 3     nbAllQueries = getAvailableCon();
 4     nbQueries = trunc(percent * nbAllQueries);
 5     nbOldQueries = nbAllQueries − nbQueries;
 6     connection_Allocation(QF_all, QF_old, nbQueries, mCon);
 7     nbOldQueries = nbOldQueries + nbQueries;
 8     connection_Allocation(QF_old, QF_all, nbOldQueries, mCon);
 9   end
10 end
```

**Algorithme 6** : $connection\_Allocation(L_1, L_2, nbCon, mCon)$

```
   input : L_1 et L_2 sont des listes de requêtes ; nbCon est le nombre de connexions à allouer aux requêtes qui sont
           dans L_1 ; mCon est le nombre maximal de connexions qu'un pair peut maintenir simultanément pour une
           requête.
 1 begin
 2   if (L_1 = ∅) then
 3     q = L_1.getFirst();
 4     while (query.getGiveCon() == mCon) do
 5       q = L_1.getNextQuery(q);
 6     end
 7     forwardBestneighbor(q);
 8     nbCon = nbCon − 1;
 9     if (q.getneighborsToSend() == 0) then
10       removeFromLists(L_1, L_2, q);
11     else
12       score = getEstimation(NextBestNeigborhood(q));
13       if (getTTL(q) <= 2 and getMin_kCurr(q) >= score) then
14         removeFromLists(L_1, L_2, q);
15       else
16         updatePosition(L_1, q);
17       end
18     end
19     if (nbCon > 0) then
20       connection_Allocation(L_1, L_2, nbCon);
21     end
22   end
23 end
```

ressources de calcul. Pour cela nous cherchons à éviter de charger plus le système lorsque la charge est importante. Pour atteindre cet objectif, nous proposons que la requête ne soit pas immédiatement envoyée à tous les voisins, mais que le nombre d'envois simultanés soit limité. Cela pose le problème du choix de l'ordre dans lequel envoyer aux voisins. Comme d'autres, nous proposons d'utiliser pour cela les descriptions des voisins permettant d'évaluer leur capacité à répondre à la requête. Dans notre approche, une requête est diffusée simultanément en parallèle à tous les voisins si l'environnement est peu chargé. Le mécanisme de diffusion s'adapte à la charge jusqu'à séquentialiser totalement la diffusion en environnement très chargé. En environnement moyennement chargé, la diffusion est réalisée en parallèle, mais à un nombre limité de voisins. Dès qu'un voisin a terminé, un nouveau voisin peut être sollicité. Notons que dans ce cas, les voisins qui ne sont pas sollicité en premier sont informés de l'état en cours du top-$k$. Ainsi, ils évitent de renvoyer des

résultats n'apportant pas d'amélioration. Si l'estimation de la qualité des résultats d'un voisin se révèle inférieure au plus petit élément du top-$k$ courant, il n'est pas nécessaire de le solliciter. En effet, sous l'hypothèse d'optimisme de la description (confère la Section 5.2), ces résultats ne peuvent améliorer ceux déjà obtenus. La propriété d'optimisme, conjuguée à une démarche plus séquentielle de la diffusion de la requête permet donc d'éviter la sollicitation de certains pairs. Cela permet donc une diminution de la charge liée au traitement d'une requête.

Notre algorithme permettant aux pairs d'attribuer les connexions aux requêtes en attente pour être transmises aux voisins fonctionne de la façon suivante. Chaque pair maintient deux listes $QF_{all}$ et $QF_{old}$. La première liste, $QF_{all}$ contient toutes les requêtes qui sont en attente d'être transmises. Cette liste est triée en fonction de l'estimation maximale des qualités des résultats par rapport à la connaissance de la description des données de chacun de ses voisins, excepté le pair à partir duquel il a reçu la requête. La seconde liste $QF_{old}$ contient les requêtes qui sont dans $QF_{all}$ triées par date de leur arrivée et qui sont qualifiées de trop vieilles. Cette liste $QF_{old}$ est importante pour éviter le problème de famine pour certaines requêtes. Notons que la période pendant laquelle une requête doit rester dans la liste $QF_{all}$ pour être qualifiée de trop vieille est fixée par l'application. Dans le cadre de notre travail ce délai au niveau de chaque pair est fixé au temps moyen de traitement en local des $\gamma$ dernières requêtes de $QF_{all}$ que ce pair a transmis à ses voisins.

**Algorithme 5** montre comment les connexions sont allouées aux requêtes en attente. Chaque pair vérifie régulièrement s'il a des connexions disponibles. Si tel est le cas, le pair assigne un certain pourcentage de ces connexions aux requêtes qui sont dans la liste $QF_{all}$ et le reste aux requêtes qui sont dans la liste $QF_{old}$. Ce pourcentage peut être fixé par l'application et sa valeur par défaut est 50%. Ensuite, le pair fait appel à la méthode *connection_Allocation* pour assigner les différents nombres de connexions aux requêtes (lignes 2-9, Algorithme 5). Si la liste des requêtes pour laquelle le pair veut assigner des connexions n'est pas vide, le pair récupère le premier élément de cette liste et vérifie si le nombre maximum de connexions simultanées actuellement ouvertes pour cette requête ne dépasse pas la valeur maximale fixée par ce pair pour une requête. Si ce nombre dépasse la limite maximale fixée pour une requête, le pair parcourt séquentiellement la liste jusqu'à trouver une requête qui satisfait à cette condition (lignes 2-6, Algorithme 6). Ensuite, le pair transmet cette requête à son voisin le plus intéressant (Celui, parmi ceux n'ayant pas encore reçu la requête, dont l'espérance que l'on obtienne de bon résultats en le sollicitant est la plus élevée) et diminue de 1 le nombre de connexions à allouer à cette liste de requêtes (lignes 7-8, Algorithme 6). Notons aussi que chaque pair inclut dans chaque message de requête qu'il envoie à son voisin, les résultats top-$k$ déjà obtenus. A la suite d'une attribution de connexion à une requête donnée d'une liste, le pair vérifie si le nombre de voisins à qui cette requête doit encore être envoyée. Si le nombre est égal à 0 alors le pair supprime cette requête dans les deux listes de requêtes (c.-à-d. $QF_{all}$ et $QF_{old}$). Sinon, si le *ttl* de la requête est inférieur ou égale 2 (car chaque pair maintient les descriptions des données des ses voisins directs et celles des voisins de ces voisins directs) et si le *mink* de l'ensemble des résultats top-$k$ courant du pair est supérieur ou égal à l'estimation de la qualité des résultats par rapport à sa connaissance de la description des données via le prochain voisin le plus intéressant, alors le pair supprime cette requête dans les deux listes. Sinon le pair met à jour la position de cette requête dans les deux listes de requêtes (lignes 15-17, Algorithme 6). Enfin, si le nombre de connexions total disponible sur le pair

est supérieur à 0, le pair fait appel à nouveau à la méthode *connection_Allocation*.

#### 5.3.2.2 Exécution locale des requêtes

Dans les approches actuelles comme par exemple dans [APV06], un pair exécute localement les requêtes dans l'ordre dans lequel elles arrivent, c'est à dire en utilisant une politique *FIFO* (First-In-First-Out). Cependant, dans un système P2P surchargé, où les files d'attentes de requête des pairs sont souvent très longues, cette approche peut augmenter considérablement les temps d'attentes des utilisateurs. Cela est dû au fait que les requêtes pour lesquelles certains pairs peuvent fournir des résultats de scores élevés peuvent être pénalisées par celles auxquelles ces pairs ne peuvent fournir que des résultats de très faibles scores. Pour améliorer ce point, nous avons proposé de changer la politique de gestion de la file d'attente. Plutôt que d'utiliser un FIFO nous avons opté pour une file prioritisée. La priorité est donnée aux requêtes dont on espère que le traitement apportera de bons résultats. Ainsi l'ordre d'exécution des requêtes sur les pairs est basé sur l'estimation de la qualité des résultats de ces requêtes par rapport aux descriptions des données locales des pairs. En traitant en priorité les requêtes présentant les meilleures estimations un pair met en attente les autres requêtes. Il est alors très vraisemblable qu'il reçoive des réponses de ses voisins avant d'avoir lui même traité la requête. Si l'analyse de ces résultats montre que la plus petite valeur du top-$k$ courant est plus forte que l'estimation des résultats qu'il peut lui même obtenir (à partir de ses données locales), il peut éviter de traiter localement la requête. C'est la propriété d'optimisme de la description qui, par un raisonnement strictement identique à celui mené dans la section précédente, permet d'arriver à cette conclusion.
Notre algorithme d'exécution locale des requêtes en attente fonctionne comme suit. Chaque pair maintient deux listes $QP_{all}$ et $QP_{old}$. La liste $QP_{all}$ contient toutes les requêtes qui sont en attente d'être exécutées localement. Cette liste est triée en fonction de l'estimation des qualités des résultats des requêtes par rapport à la description des données locales du pair. La liste $QP_{old}$ contient des requêtes qui sont dans $QP_{all}$ et qui sont qualifiées de trop vieilles. La liste $QP_{old}$ est triée par date d'arrivée des requêtes.
**Algorithme 7** montre le mécanisme d'exécution locale des requêtes en attente. Chaque pair vérifie la valeur de variable booléenne $flag$ (la valeur de $flag$ est fixée initialement à $true$). La valeur de $flag$ indique par quelle liste l'exécution locale des requêtes en attente doit débuter. Si la valeur de $flag$ est égale à $true$, la requête à exécuter est choisie dans la liste $QP_{all}$. Sinon cette requête est choisie dans la liste $QP_{old}$ (lignes 2-7, Algorithme 7). Le pair exécute ensuite la requête choisie et la supprime dans les listes $QP_{all}$ et $QP_{old}$ (lignes 8-9, Algorithme 7). Enfin, la valeur de $flag$ est positionnée à $!flag$ (ligne 10, Algorithme 7).

### 5.3.3 Remontée des réponses

Une solution naïve pour réduire le temps d'attente des utilisateurs dans le traitement d'une requête top-$k$ est de retourner les résultats locaux de chaque pair directement au pair initiateur de la requête dès que ces pairs ont fini d'exécuter la requête. Cependant, cela peut entrainer le transfert d'un grand nombre de résultats qui peut donc augmenter considérablement le trafic réseau et ainsi créer un goulot d'étranglement sur le pair initiateur de la requête. Pour cela l'approche naïve n'est pas efficace pour la remontée des réponses dans les systèmes surchargés.

**Algorithme 7** : $execute\_Query(QP_{all}, QP_{old}, flag)$

```
input : QP_all, liste des requêtes à traiter localement triée en fonction des estimations des qualités de réponses ;
        QP_old, liste des requêtes qui sont qualifiées de vieilles ; flag, indique dans quelle liste choisir la requête à
        traiter localement.
1  begin
2      if (flag) then
3          query = QP_all.getFirst();
4      end
5      else
6          query = QP_old.getFirst();
7      end
8      executeLocally(query);
9      removeFromLists(QP_all, QP_old, query);
10     flag =!flag;
11 end
```

Une autre variante de cette solution consiste à faire remonter les réponses non pas directement à l'utilisateur, mais via l'arbre de propagation de la requête. Sans précaution supplémentaire, cela peut augmenter drastiquement le nombre de message dans le réseau sans résoudre le problème de goulot d'étranglement. La solution vient du fait que les pairs peuvent conserver localement les résultats de la requêtes et ne faire remonter les nouvelles réponses que si elles présentent une amélioration. Ainsi, toutes les réponses ne parviennent pas jusqu'à l'initiateur et le nombre de message supplémentaire peut ne pas être trop élevé. Encore faut-il que la stratégie de remontée soit finement choisie. Nous proposons ici d'utiliser celle qui s'est révélée la meilleure lors des évaluations de la méthode ASAP. Il s'agit donc de l'approche de remontée de réponses d'ASAP faisant usage d'un impact d'amélioration basé sur les scores des résultats et utilisant un seuil dynamique (confère Chapitre 4).
Notons aussi que la remontée des réponses via l'arbre de propagation de la requête présente un autre intérêt : celui de permettre aux pairs de se baser sur les résultats déjà obtenus des pairs appartenant à leurs sous-arbres pour éviter de traiter localement certaines requêtes et ceci à la propriété d'optimisme des descriptions synthétiques.

### 5.3.4 Optimisation

Durant l'exécution d'une requête, un pair peut recevoir de ses fils des résultats top-$k$ intermédiaires dont la qualité est meilleure que celle que peuvent fournir certains de ses voisins directs qui ont déjà reçu la requête mais qui n'ont encore remonté aucun résultat. Dans ce cas, nous proposons une optimisation de QUAT pour éviter aux pairs de traiter localement cette requête et aussi ne pas faire remonter vers leurs parents des résultats qui ne sont pas intéressants. Notre mécanisme d'optimisation fonctionne comme suit. Quand un pair $p$ reçoit de ses fils des résultats intermédiaires d'une requête top-$k$ $q$, $p$ compare le $mink$ de son ensemble des résultats top-$k$ courant avec l'estimation de la qualité des résultats de $q$ par rapport aux descriptions des voisins directs de $p$ qui ont déjà reçu $q$ mais qui n'ont pas encore retourné aucun résultat à $p$. Si le $mink$ est supérieur ou égal à l'estimation de la qualité des résultats de la requête, alors $p$ envoie un message urgent contenant son ensemble de résultats top-$k$ intermédiaire à ses voisins. Ce message permet aux pairs qui l'ont reçu de ne pas exécuter localement la requête $q$, et de ne pas remonter les résultats dont les scores sont inférieurs à ce $mink$. Chaque voisin de $p$ qui reçoit un

message urgent procède de manière identique à $p$ quand le $mink$ de son top-$k$ courant est supérieur ou égal à l'estimation de la qualité des résultats de ses voisins qui ont déjà reçu la requête mais qui n'ont pas encore retourné de résultat. Bien que cette optimisation a un faible impact sur le nombre de messages de réponse, il peut cependant diminuer considérablement le volume de données transféré sur le réseau et la charge des pairs du système.

## 5.4 INDICES DE ROUTAGES DISTRIBUÉS

Dans cette section, nous décrivons d'abord comment construire les indices de routage distribué pour le traitement des requêtes top-$k$ et ensuite comment maintenir ces indices.

### 5.4.1 Construction des indices de routages

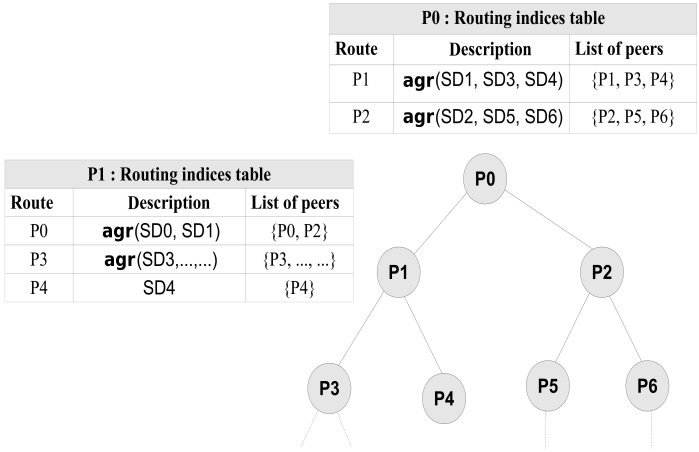



Figure 5.3 **Routing Indices Tables of Peers**

Classiquement un indice de routage (ou un index de routage tout court) permet simplement d'envoyer une requête aux voisins pouvant fournir des réponses à cette requête [CGM02]. Cependant dans notre cas, un indice de routage est utilisé pour permettre à un pair de pouvoir déterminer la priorité des pairs voisins dans la transmission ou le forwarding des requêtes quand il y a une forte charge de requêtes dans le système. Il permet aussi aux pairs d'éviter de transmettre une requête aux voisins si les résultats de ces voisins à cette requête ne peuvent pas améliorer l'ensemble des résultats top-$k$ courant. Dans QUAT, un index de routage d'un pair $p$ est donc une structure de données qui, étant donnée une requête, retourne la liste des voisins de $p$ triée en fonction de leurs potentialités à répondre à cette requête. La création de ces indices de routages est effectuée par les pairs du système de la manière suivante. Quand un nouveau pair $p_i$ entre dans le système P2P, il échange sa propre description avec celles de ses voisins directs et celles des voisins directs de ces voisins (c.-à-d. des pairs se trouvant à 2 sauts ou pas de $p_i$). En utilisant ces descriptions, le pair $p_i$ crée une table de description de son voisinage. Cette table contient un identifiant

de chaque pair $p_j$ voisin de $p_i$ et l'agrégation de la description locale de $p_j$ avec celles de ses voisins directs (voir Figure 5.3). Les tables de descriptions sont utilisées comme indices de routage pour le traitement d'une requête top-$k$.

### 5.4.2 Maintenance des indices de routages

Les mises à jour des données d'un pair peut entraîner une modification de sa description. Si tel est le cas, la nouvelle description doit être envoyée à tous les pairs voisins pour assurer une meilleure précision des résultats retournés à l'utilisateur. Une solution naïve pour maintenir à jour les descriptions est de diffuser en cas de modification des données d'un pair, un message de mise à jour contenant la nouvelle description du pair et ayant un $ttl = 1$ à tous les voisins directs de ce pair. Chaque pair voisin qui reçoit ce message de mise à jour diminue son $ttl$ de 1 et transmet à son tour ce message à tous ses voisins directs (excepté le pair qui lui a transmis le message) jusqu'à ce que la valeur du $ttl$ atteigne 0. La maintenance d'un index de routage après une modification de la description d'un pair est donc effectuée en $O(\varphi + \varphi^2)$ messages où $\varphi$ est le degré moyen des pairs du système P2P.
Pour des raisons d'efficacité, il est plus judicieux qu'un pair ne propage pas sa nouvelle description à ses voisins si l'ancienne description est plus optimiste que celle-ci. Cette technique permet de réduire le coût de maintenance des indices de routage tout en préservant la précision des résultats. Cependant une mise à jour périodique est nécessaire pour gérer les mises à jour des indices de routages.
Enfin, une mise à jour spéciale peut survenir en cas de déconnexion des pairs du système. Quand un pair $p_i$ détecte la déconnexion (dû par exemple à une panne ou une sortie volontaire) d'un de ses voisins $p_j$, alors $p_i$ met à jour son index de routage en supprimant la ligne de $p_i$ dans sa table de description. Ensuite, il informe ses voisins directs, en leur envoyant un message de mise à jour avec $ttl = 1$. Chaque voisin qui reçoit ce message de mise à jour diminue $ttl$ de 1 et envoie à ce message à son tour à ses voisins (jusqu'à ce que la valeur du $ttl$ atteigne 0).

## 5.5 ÉVALUATION EXPÉRIMENTALE

Dans cette section, nous évaluons la performance de QUAT par simulation en utilisant le simulateur PeerSim [JMJV]. Cette section est organisée comme suit. Tout d'abord, nous décrivons notre environnement de simulation, les paramètres utilisés pour l'évaluation de la performance. Ensuite, nous étudions l'effet du taux d'arrivée des requêtes, du nombre de pairs et le nombre de résultats souhaité par l'utilisateur sur la performance de QUAT, et ainsi montrer comment QUAT passe à l'échelle. Ensuite, nous étudions l'effet des pannes des pairs sur la précision des résultats retournés par QUAT.
Nous effectuons notre expérimentation sur une machine Intel Pentium 2,4 GHz et ayant 2 Go de mémoire. Le Tableau 5.1 décrit les paramètres de notre simulation. Nous utilisons les valeurs des paramètres qui sont typiques aux systèmes P2P [GSG02]. Le temps de latence entre deux pairs est un nombre aléatoire normalement distribué avec une moyenne de 200 ms. Étant donné que les utilisateurs sont généralement intéressés par un petit nombre de résultats, nous mettons $k = 20$ comme valeur par défaut. Dans nos expériences nous varions la taille du réseau de 1000 à 10000 pairs. Dans le but de simuler l'hétérogénéité

Tableau 5.1 **Les paramètres de simulation**

| **Paramètres** | **Valeurs par défaut** |
|---|---|
| Temps de Latence | Nombre aléatoire, Moyenne=200 ms, Variance=100 |
| Nombre de pairs | 10000 |
| Degré moyen des pairs | 4 |
| $ttl$ | 9 |
| $k$ | 20 |
| Taux d'arrivée des requêtes | 50 requêtes par seconde (qps) |
| $\gamma$ | 500 |

des pairs nous attribuons des capacités aux pairs du système conformément aux résultats présentés dans [GSG02]. Ce travail mesure la capacité des pairs dans le système Gnutella. Sur la base de ces résultats, nous générons environ 10% de pairs faiblement capables, 60% de pairs moyennement capables, et 30% de pairs énormément capables. Les pairs énormément capables sont 3 fois plus capables que les pairs moyennement capables et 7 fois plus capables que les pairs faiblement capables. Nous avons fixé à 50 $qps$ [2] la valeur par défaut du taux d'arrivée des requêtes dans le système. Le nombre de requêtes considérées pour calculer le temps moyen d'attente d'une requête pour qu'elle soit qualifiée de très vieille est fixé à $\gamma = 500$. Chaque expérience est exécutée pendant 2 heures, qui sont mappées sur des unités de temps de simulation.

### 5.5.1 Environnement de simulation

#### 5.5.1.1 Jeux de données

Nous menons nos expériences sur deux ensembles de données : les logs de serveur HTTP et un ensemble de données synthétiques. L'Internet Traffic Archive [3] fournit un énorme logs de serveur HTTP d'environ 1,3 milliards de requêtes utilisateurs de la coupe du monde FIFA 1998. Nous avons regroupé certaines informations de ces logs dans une table relationnelle dont le schéma est $Log(interval, userid, bytes)$, l'agrégation du trafic (en octets) pour chaque utilisateur et ceci dans l'intervalle d'une journée. Cette table est partitionnée horizontalement et de façon uniforme sur tous les pairs du système P2P. Les requêtes utilisées pour les tests concernent les top-$k$ utilisateurs actives, c.-à-d. les $k$ utilisateurs qui ont le plus grand trafic dans un intervalle donné (comme au "$1^{er}$ juin").
En ce qui concerne l'ensemble des données synthétiques nous les avons générées comme suit. Chaque pair dispose d'une table $\mathcal{R}(score, data)$ dans laquelle l'attribut $score$ est un nombre réel aléatoire dans l'intervalle $[0, 1]$ avec une distribution uniforme, et l'attribut $data$ une variable aléatoire de distribution normale avec une moyenne de 1 et une variance de 64. L'attribut $score$ représente les scores des données et l'attribut $data$ représente les données qui seront retournées à l'utilisateur comme résultat du traitement de la requête. Le nombre de tuples de $\mathcal{R}$ sur chaque pair est un nombre aléatoire (uniformément distribué sur tous les pairs). Ce nombre est compris entre 1000 et 10000. Une requête typique utilisée dans nos tests est "SELECT $\mathcal{R}.data$ FROM $\mathcal{R}$ ORDER BY $\mathcal{R}.score$ STOP AFTER $k$".
A partir des données assignées à chaque pair $p$, nous construisons la description synthétique

2. query-per-second
3. http ://ita.ee.lbl.gov

de $p$ grâce à la technique de résumées de données des bases de données relationnelles proposées dans [HRVM08] . Cependant, nous assurons que les descriptions obtenues à partir de cette technique respectent aux propriétés des descriptions synthétiques énoncées dans la Section 5.1.3.

#### 5.5.1.2 Métriques

Pour évaluer la performance de l'approche QUAT par rapport aux autres approches que nous présenterons dans la section suivante, nous utilisons les métriques suivantes :

(i) **Qualité restante cumulée sur une période de temps :** confère Section 5.2 pour la définition.

(ii) **Temps de stabilisation sur une période temps :** confère la Section 5.2 pour la définition.

(iii) **Temps de réponse sur une période de temps :** Nous définissons le temps de réponse sur une période de temps par la moyenne de toutes les temps de réponses de toutes les requêtes émises dans le système sur cette période de temps. Notons que le temps de réponse est le temps que doit atteindre l'utilisateur pour que l'exécution d'une requête top-$k$ prenne fin.

(iv) **Proportion de requêtes reçues par pair :** Nous définissons cette proportion par la moyenne du ratio entre nombre de requêtes reçues par un pair et le nombre de requêtes émises dans le système sur une période de temps.

(v) **Proportion de requêtes traitées par pair :** Nous définissons cette proportion par la moyenne du ratio entre nombre de requêtes exécutées localement par un pair et le nombre de requêtes émises dans le système sur une période de temps

(vi) **Coût de communication sur une période de temps :** Nous mesurons le coût de communication en termes de moyenne en nombre de messages de réponse et de volume de données transféré sur le réseau pour le traitement des requêtes top-$k$ émises dans le système sur une période de temps.

(vii) **Précision des résultats sur une période de temps :** Nous définissons la précision des résultats sur une période temps par la moyenne de toutes précisions des requêtes top-$k$ émises sur une période de temps. Notons que la précision est la proportion des résultats exacts dans l'ensemble des résultats reçu par l'utilisateur à la fin de l'exécution de la requête.

#### 5.5.1.3 Approches de base

Nous avons comparé QUAT aux approches Fully Distributed (FD) [APV06] que nous avons présenté dans l'état de l'art et As Soon As Possible (ASAP) [DLAV10a] que nous avons présenté dans le chapitre précédent. Les deux approches sont des solutions de traitement de requêtes top-$k$ dans les systèmes P2P non-structurés et utilisant comme hypothèse une distribution horizontale des données. Dans FD, Chaque pair qui reçoit une requête, l'exécute localement (c.-à-d. sélectionne les $k$ meilleurs scores), et attend les résultats de ses fils. Après avoir reçu tous les score-listes de ses fils, les pairs fusionnent leurs résultats locaux avec ceux reçus de leurs fils, sélectionne les $k$ meilleurs et envoie le résultat à leurs parents. Contrairement à FD, dans ASAP, un pair n'attend pas tous les résultats de ses fils avant de remonter les résultats vers son parent. Chaque pair (excepté le pair initiateur)

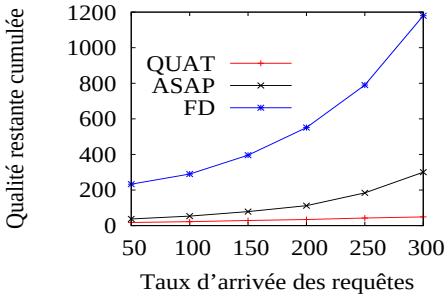
retourne le plus tôt possible à son parent ses résultats intermédiaires qui sont de bonne qualité et donc qui sont susceptibles d'être dans le top-$k$ final.

### 5.5.2 Résultats et analyses des performances

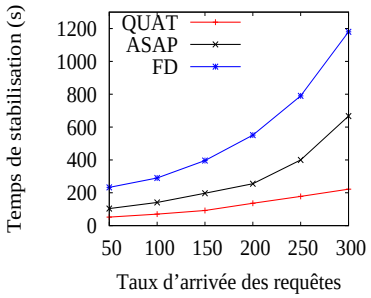
Dans cette section, nous présentons et analysons les résultats de notre expérimentation.

#### 5.5.2.1 Effet du taux d'arrivée des requêtes

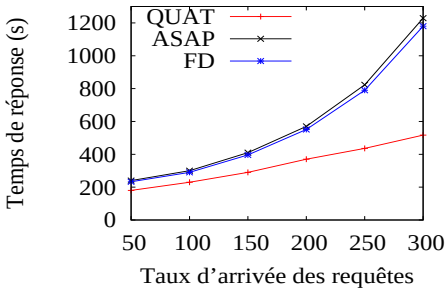
Dans cette section, nous étudions l'effet du taux de l'arrivée des requêtes sur les performances de QUAT. Pour cela, nous avons effectué des tests en utilisant les jeux des données des logs HTTP pour étudier comment évoluent, la qualité restante cumulée, le temps de stabilisation, le temps de réponse, la proportion des requêtes reçues, la proportion des requêtes traitées et le coût de la communication quand le taux d'arrivée des requêtes varie de 50 à 300 requêtes par seconde. Notons que les autres paramètres de la simulation sont fixés comme indiqué dans le Tableau 5.1.
Figures 5.4(a) et 5.4(b) montrent respectivement comment la qualité restante cumulée et le temps de stabilisation augmentent avec le taux d'arrivée des requêtes sur une période de 2 heures. Les résultats montrent que la qualité restante cumulée de QUAT est toujours plus petite que celle d'ASAP et celle de FD, ce qui signifie que QUAT retourne les résultats de meilleures qualités beaucoup plus rapidement que ASAP et FD. Les résultats montrent également que le temps de stabilisation de QUAT est toujours plus petit que celui d'ASAP et celui de FD. La raison est que dans QUAT, les pairs traitent en priorité les requêtes auxquelles ils peuvent fournir des résultats de bonne qualité.
Figure 5.4(c) montre que le temps de réponse de QUAT sur une période de 2 heures est toujours bien meilleur que celui d'ASAP et celui de FD. La raison principale est que, les pairs n'exécutent pas localement certaines requêtes auxquelles ils ne disposent pas de données intéressantes, ce qui aident les pairs à sauvegarder leurs ressources pour le traitement d'autres requêtes.
Figures 5.4(d) and 5.4(e) montrent que la proportion de requêtes reçues et la proportion de requêtes traitées par les pairs sur une période de 2 heures diminuent quand le taux d'arrivée des requêtes augmente. Cela vient du fait que, quand le taux d'arrivée des requêtes augmente, les pairs réduisent le nombre de connexions qu'ils peuvent maintenir simultanément pour une requête. En plus de cela, les pairs s'appuient sur la connaissance des descriptions de leurs voisins pour éviter d'envoyer des requêtes à certains voisins. En outre, les pairs exploitent leurs descriptions (c.-à-d. la description des données locales partagées) pour éviter l'exécution locale de certaines requêtes.
Figure 5.4(f) montre l'augmentation du volume de données transféré en fonction du taux d'arrivée des requêtes. Les résultats montrent que le volume de données transféré dans le cas de QUAT est toujours un peu élevé que celui d'ASAP. Les résultats montrent également que la différence entre les volumes de données transférés dans le cas de QUAT et dans le cas FD n'est pas significative.
Figures 5.5(a), 5.5(b) et 5.5(c) montrent respectivement comment la qualité restante cumulée, le temps de stabilisation et le temps de réponse augmentent avec le taux d'arrivée des requêtes sur une période de 2 heures en menant les tests sur les données synthétiques. Les résultats montrent que nous avons les mêmes tendances dans le cas des deux jeux de données (ex. logs HTTP et des données synthétiques).



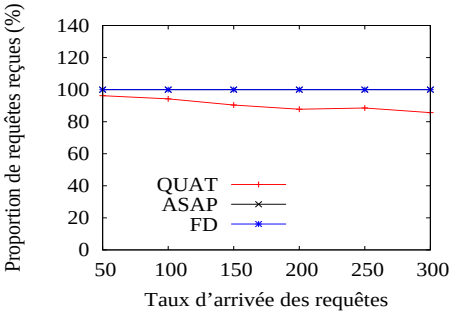

(a) Qualité restante cumulée vs. Charge du système



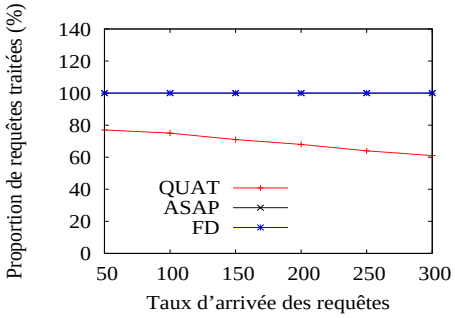

(b) Temps de stabilisation vs. Charge du système



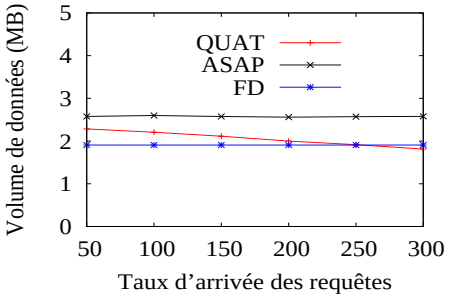

(c) Le temps de réponse vs. Charge du système




(d) % de requêtes reçues vs. Charge du système




(e) % de requêtes exécutées vs. Charge du système

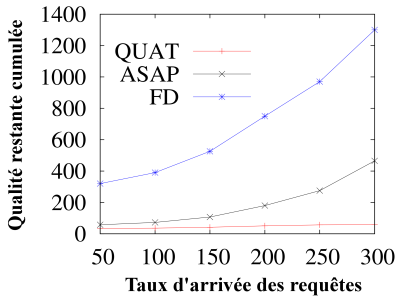



(f) Volume de données transféré vs. Charge du système

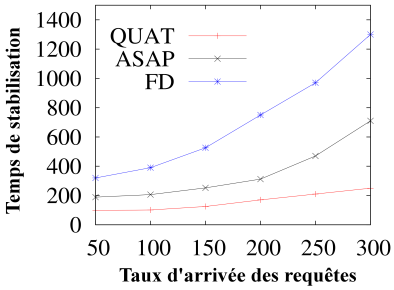
Figure 5.4 **Logs HTTP : impact de la Charge du système sur la performance de QUAT**

#### 5.5.2.2 Effet du nombre de pairs

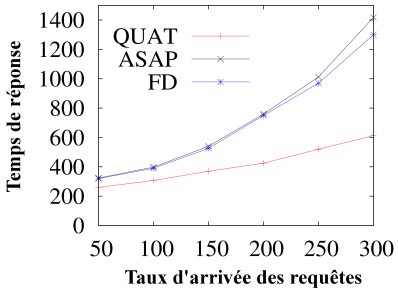
Dans cette section, nous étudions l'impact du nombre de pairs sur la performance de QUAT. Pour cela, nous effectuons des tests en utilisant les données des logs HTTP pour




(a) Qualité restante cumulée vs. Charge du système




(b) Temps de stabilisation vs. Charge du système



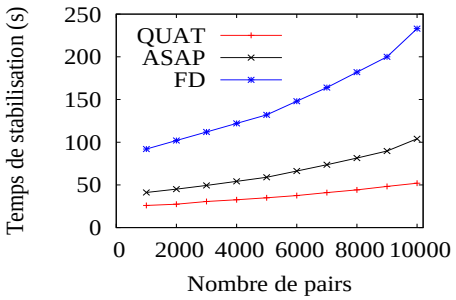

(c) Temps de réponse vs. Charge du système

Figure 5.5 **Données synthétiques : impact de la charge du système sur la performance de QUAT**

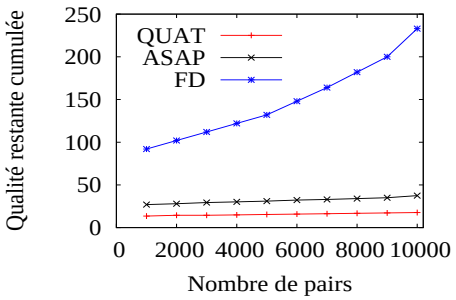
étudier comment la qualité restante cumulée, le temps de stabilisation et le temps de réponse augmentent sur une période de 2 heures en faisant varier le nombre de pairs de 1000 à 10000 et les autres paramètres de simulation fixés comme indiqué dans le dans le Tableau 5.1. Figure 5.6(a), 5.6(b) et 5.6(b) montrent respectivement que la qualité restante cumulée, le temps de stabilisation et le temps augmentent avec le nombre de pairs. Les résultats montrent que la qualité restante cumulée, le temps de stabilisation et le temps de réponse de QUAT sont toujours beaucoup plus petits que ceux d'ASAP et de FD.

#### 5.5.2.3 Effet de $k$

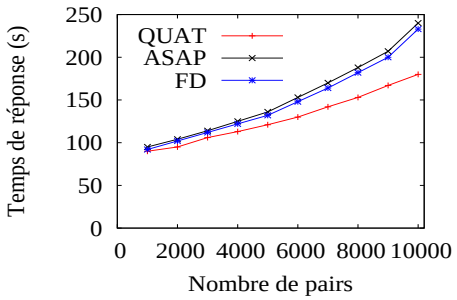
Dans cette section, nous étudions l'impact de $k$, c.-à-d. le nombre de résultats souhaité par l'utilisateur sur la performance de QUAT. En utilisant notre simulateur et des données des logs HTTP, nous menons des tests pour étudier comment la qualité restante cumulée, le temps de stabilisation et le volume de données transféré (sur une période de 2 heures) évoluent en faisant varier $k$ de 20 à 100, avec les autres paramètres de simulation fixés comme indiqué dans le dans le Tableau 5.1. Les résultats (voir Figure 5.7(a) et Figure 5.7(b)) montrent que $k$ à un très faible impact sur la qualité restante cumulée et le




(a) Temps de stabilisation vs. Nombre de pairs




(b) Qualité restante cumulée vs. Nombre de pairs



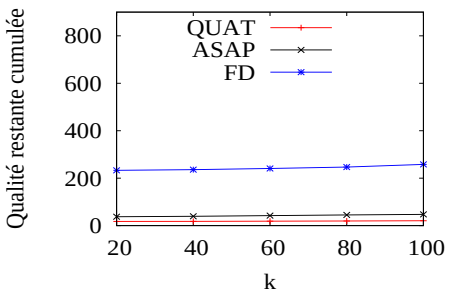

(c) Temps de réponse vs. Nombre de pairs

Figure 5.6 **Logs HTTP : impact du nombre de pairs sur la performance de QUAT**

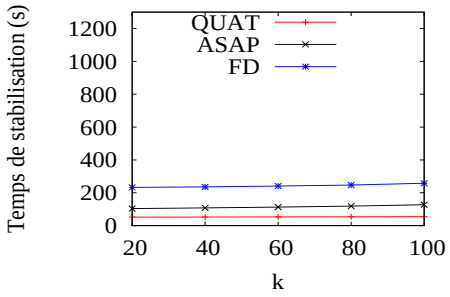
temps de stabilisation de QUAT. les résultats montrent également (voir Figure 5.7(c)) que en augmentant $k$, le volume de données transféré dans le cas de QUAT augmente moins que celui de FD et celui d'ASAP. Cela est dû au fait que QUAT élimine plus de résultats non intéressants quand $k$ augmente.

#### 5.5.2.4 Effet des pannes des pairs

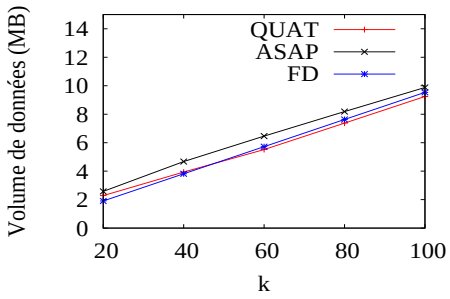
Dans cette section, nous étudions l'effet des pannes des pairs sur la précision de l'ensemble des résultats top-$k$ final. Dans nos tests, nous varions la valeur du taux de panne et observons son effet sur la précision des résultats top-$k$ et ceci sur une période de 2h. Figure 5.8 montre que la précision des résultats top-$k$ de QUAT, ASAP et FD quand le taux de panne dans le système augmente et les autres paramètres de simulations fixés comme indiqué dans le Tableau 5.1. Les résultats montrent que les pannes des pairs ont moins d'impact sur les précisions des résultats retournés par QUAT et ASAP que FD. La raison en est que QUAT et ASAP retournent les résultats de bonne qualité le plus tôt possible à l'utilisateur. Cependant, l'augmentation du taux de pannes dans le cas de FD engendre une diminution significative de la précision des résultats top-$k$. La raison principale est que dans FD, chaque pair attend les résultats de tous ses fils ainsi en cas de panne d'un pair, tous les scores-listes déjà reçus par ce pair sont perdus.




(a) Qualité restante cumulée vs. $k$




(b) Temps de stabilisation vs. $k$




(c) Volume de données transféré vs. $k$

Figure 5.7 **Logs HTTP : impact de $k$ sur la performance de QUAT**

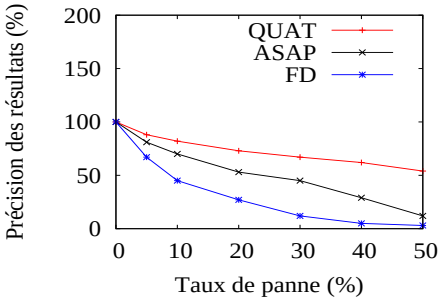



Figure 5.8 **Logs HTTP : précision des résultats vs. Taux de panne**

## 5.6 CONCLUSION

Dans ce chapitre, nous avons adressé le problème de traitement des requêtes dans les systèmes P2P surchargés. Notre objectif est de réduire le temps d'attente des utilisateurs dans le cadre du traitement des requêtes top-$k$ dans système P2P en présence de forte charge de requêtes en lui renvoyant les résultats intermédiaires de bonne qualité le plus

tôt possible. Nous avons donc proposé QUAT, un algorithme efficace qui prend compte la charge des pairs afin de pouvoir retourner les résultats top-$k$ le plus tôt possible à l'utilisateur. Dans QUAT, chaque pair maintient une description synthétique de ses données locales et aussi des descriptions des données de ses voisins se trouvant dans un rayon de 2 pas. QUAT s'appuie sur ces descriptions pour créer des indices de routages lui permettant d'identifier les meilleurs voisins auxquels il faut envoyer une requête donnée pour obtenir rapidement les résultats de meilleurs qualités. Ces descriptions lui permet ainsi de réduire la consommation des ressources (ressources réseau et de calcul) en évitant une exécution locale et une propagation par les pairs des requêtes qui ne peuvent pas améliorer le temps de stabilisation ou la qualité restante cumulée.
Nous avons validé QUAT à travers une implémentation et une expérimentation très poussée en le comparant par rapport à ASAP et FD. Les résultats ont montré que QUAT surpasse ces deux approches en fournissant rapidement l'ensemble des résultats top-$k$ à l'utilisateur. Les résultats ont montré également que l'augmentation de la charge des requêtes entraîne une augmentation significative du temps de réponse de FD et ASAP alors que QUAT s'adapte mieux à cette situation. Enfin, les résultats ont montré que en cas de pannes des pairs la précision des résultats top-$k$ de QUAT est meilleure par rapport à celle de FD et d'ASAP.

# CHAPITRE 6
# CONCLUSION

Les réseaux P2P sont une architecture intéressante pour le support d'applications de partage de données distribuées à grande échelle. Les requêtes complexes de type top-$k$ sont essentielles pour plusieurs applications telles que la recherche d'information, le commerce électronique, l'analyse de données, etc. Bien que le traitement des requêtes top-$k$ a été largement étudié dans la littérature, le traitement de requêtes top-$k$ de façon complètement distribuée dans une communauté virtuelle P2P de partage de données crée de nouveaux défis. Dans ce chapitre, nous résumons nos principales contributions. Nous discutons également des orientations futures que nous voulons donner à ce travail.

## 6.1 CONTRIBUTIONS

Nos contributions dans cette thèse peuvent être résumées en trois points décrits ci-après.

### 6.1.1 Nouvelles métriques de mesure de performances des algorithmes top-$k$

Nous avons proposé deux nouvelles métriques qui sont *temps de stabilisation* et *qualité restante cumulée*. Ces deux métriques viennent en supplément du temps de réponse pour qualifier le temps d'attente de l'utilisateur pour le traitement d'une requête top-$k$.

### 6.1.2 ASAP

Nous avons défini le problème du traitement des requêtes top-$k$ le plus tôt possible dans les systèmes P2P non-structurés et ceci grâce aux nouvelles métriques que nous avons proposées. Ensuite nous avons proposé ASAP une famille d'algorithmes complètement distribuée qui fournit à l'utilisateur des résultats de bonne qualité le plus tôt possible. Pour améliorer les performances sans impacter les coûts, nous avons également introduit des notions d'impact des résultats disponibles et de seuil basé sur la couverture locale des résultats d'un pair pour décider si une réponse donnée peut ou non être renvoyée rapidement. Nous avons démontré par simulation que ASAP permet à l'utilisateur d'obtenir nettement plus tôt toutes les meilleures réponses comparativement aux approches classiques. Les simulations ont montré que les surcoûts en terme de trafic réseau générés par ASAP sont raisonnables. Les simulations ont également montré que notre solution passe à l'échelle pour un grand nombre de pairs et pour un nombre élevé de résultats demandé par l'utilisateur. Les résultats ont montré aussi qu'en cas de dynamicité ou de pannes des pairs, les résultats top-$k$ approximatif qu'on obtient avec ASAP sont de meilleures précisions que celles obtenues avec des solutions existantes. Enfin, nous avons proposé deux mesures qui permettent à l'utilisateur d'avoir une idée sur le top-$k$ courant : la probabilité pour que

le top-$k$ courant soit le top-$k$ exact ou final (probabilité de stabilisation) et la proportion des pairs ayant contribué au top-$k$ courant (proportion des pairs contributeurs).

### 6.1.3 QUAT

QUAT est une approche de traitement de requêtes top-$k$ dont l'objectif est de réduire le temps d'attente des utilisateurs dans les systèmes P2P surchargés, particulièrement critique pour les solutions classiques. QUAT fait usage de descriptions synthétiques des pairs avec deux objectifs : permettre à un pair de traiter en priorité les requêtes pour lesquelles il peut fournir des résultats de bonne qualité ; et router en priorité les requêtes vers les voisins susceptibles de retourner les résultats de meilleure qualité. Nous avons évaluer expérimentalement QUAT en utilisant à la fois des jeux de données synthétiques et réelles. Les résultats ont montré que QUAT offre une nette amélioration par rapport aux approches classiques dans les systèmes surchargés et un comportement équivalent dans les systèmes non chargés.

## 6.2 TRAVAUX FUTURS

Nous envisageons les perspectives suivantes :
- Une expérimentation plus approfondie de QUAT permettant de mesurer le rapport gain/potentiel de cette approche par rapport au coût de maintenance des descriptions synthétiques.
- L'extension de nos approches aux requêtes top-$k$ skyline [BKS01] .
- L'extension de nos approches de requêtes top-$k$ aux données incertaines.

### 6.2.1 Expérimentation plus approfondie de QUAT

Nous pensons qu'il serait très intéressant de mesurer le rapport gain/potentiel de l'approche QUAT (en termes de temps de stabilisation, qualité restante cumulée, temps de réponse et de la charge des pairs et du réseau) par rapport au coût de maintenance (en cas de forte dynamicité du réseau P2P) des descriptions synthétiques des pairs et ceci dans différents contextes correspondants à des scénarios réalistes. Cela nous permettra de mettre en évidence les différents types d'applications pour lesquelles cette solution serait très utile.

### 6.2.2 Les requêtes top-$k$ skyline

Les requêtes skylines [VDKV07, FP08] sont un autre type de requête complexe qui sont très intéressantes dans le cadre d'une communauté virtuelle de partage de données P2P. Le skyline $S$ d'un ensemble de points $d$-dimensionnel est l'ensemble des points qui ne sont pas dominés par tout autre point de $S$. Un point $p$ est dominé par un autre point $q$ si et seulement si $q$ n'est pas pire que $p$ sur aucune des dimensions, et que $q$ est meilleur que $p$ sur au moins une dimension. Cependant, le traitement des requêtes de type skyline dans un environnement P2P pose deux problèmes fondamentaux : le nombre de résultats retournés qui peut être très grand quand la dimension de la requête augmente et le temps de réponse qui peut être long puisque dans un environnement complètement distribué le résultat de

l'ensemble skyline est retourné quand tous les pairs ont évalué la requête. Pour résoudre le premier problème, un autre type de requête skyline a été proposé : le top-$k$ skyline [YM07, GV09]. Il s'agit de retourner à l'utilisateur les $k$ points les plus importants d'un ensemble skyline. Bien que cette solution réduise le trafic réseau, la question du temps de réponse qui peut dépendre du pair le plus lent n'est pas résolu. L'idée est d'étudier comment utiliser notre solution ASAP dans le contexte des requêtes top-$k$ skyline. Cependant, il faut noter que notre solution ASAP n'est pas directement applicable aux requêtes top-$k$ skyline car le score d'une donnée d'un pair dépend du nombre de tuples que possède ce pair et ce score peut changer tout au long de l'exécution de la requête si les résultats d'un pair doivent transiter par des pairs intermédiaires avant d'atteindre le pair initiateur de la requête.

### 6.2.3 Top-$k$ sur les données incertaines

De nos jours des incertitudes sur les données apparaissent fréquemment dans les systèmes distribués tels que les réseaux de capteurs sans fil, les serveurs web distribués et les systèmes P2P. Collecter toutes les données incertaines d'un tel système pour traiter les requêtes top-$k$ de manière centralisée peut engendrer un énorme coût de communication, de temps d'attente et de coût de stockage données. En outre, la plupart des approches centralisées pour le traitement des requêtes top-$k$ sur des données incertaines ne sont pas applicables aux systèmes P2P compte tenu de leurs caractéristiques particulières. Notre objectif est d'adresser le problème du traitement des requêtes top-$k$ dans les systèmes P2P sur les données incertaines. En effet l'incertitude sur les données engendre de nombreuses sémantiques possibles à l'égard des requêtes top-$k$. En présence des sémantiques d'un monde possible, chaque base de données incertaine peut être vu comme un encodage succincte d'une distribution sur les mondes possibles. Chaque monde possible est une table relationnelle (des données certaines) sur laquelle nous pouvons évaluer la requête top-$k$ de façon traditionnelle. Il y a très peu de travaux sur le traitement des requêtes top-$k$ dans les systèmes P2P sur les données incertaines [YLLL10]. Les approches existantes cherchent à minimiser le coût de communication. Notre objectif futur est donc de proposer un algorithme qui permet de réduire le temps d'attente de l'utilisateur en lui fournissant le plus tôt possible des résultats intermédiaires de bonne qualité ayant de forte probabilité.

# Bibliographie


[ACMD+03] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid : a self-organizing structured p2p system. *ACM SIGMOD Record*, 32(3) :29–33, 2003.

[ACMH03] Karl Aberer, Philippe Cudré-Mauroux, and Manfred Hauswirth. The chatty web : emergent semantics through gossiping. In *Int. Conf. on World Wide Web (WWW)*, pages 197–206, 2003.

[AM07] Reza Akbarinia and Vidal Martins. Data management in the appa system. *Journal of Grid Computing*, 5(3) :303–317, 2007.

[AMPV06a] R. Akbarinia, V. Martins, E. Pacitti, and P. Valduriez. *Global Data Management*, chapter Design and Implementation of Atlas P2P Architecture. IOS Press, first edition, 2006.

[AMPV06b] Reza Akbarinia, Vidal Martins, Esther Pacitti, and Patrick Valduriez. Design and implementation of appa. In *Global Data Management (Eds. R. Baldoni, G. Cortese and F.Davide), IOS Press*, 2006.

[AMPV06c] Reza Akbarinia, Vidal Martins, Esther Pacitti, and Patrick Valduriez. Top-k query processing in the appa p2p system. In *VECPAR*, pages 158–171, 2006.

[APV06] Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Reducing network traffic in unstructured p2p systems using top-k queries. *Distributed and Parallel Databases*, 19(2-3) :67–86, 2006.

[APV07] Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Best position algorithms for top-k queries. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 495–506, 2007.

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4) :335–371, 2004.

[BAH+06] R. Blanco, N. Ahmed, D. Hadaller, L. G. A. Sung, H. Li, and M. A. Soliman. A survey of data management in peer-to-peer systems. Technical report, David R. Cheriton School of Computer Science, University of Waterloo, jun 2006.

[BBK01] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces : Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3) :322–373, 2001.

[BGK+02] Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and Ilya Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proc of the Int. Workshop on the Web and Databases (WebDB)*, pages 89–94, 2002.

[BGZ04] Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. Efficient distributed skylining for web information systems. In *Prooceeding of Int. Conf. on Extending Database Technology (EDBT)*, pages 256–273, 2004.

[BKS01] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 421–430, 2001.

[Blo70] Burton H. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM*, 13(7) :422–426, 1970.

[BNST05] Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski, and Uwe Thaden. Progressive distributed top k retrieval in peer-to-peer networks. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 174–185, 2005.

[BO03] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of ACM. Int Conf. on Management of Data (SIGMOD)*, pages 28–39, 2003.

[CAPMN03] Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. Planetp : Using gossiping to build content addressable peer-to-peer information sharing communities. In *Proceedings of IEEE Int. Symp. on High-Performance Distributed Computing (HPDC)*, pages 236–249, 2003.

[CF04] Min Cai and Martin R. Frank. Rdfpeers : a scalable distributed rdf repository based on a structured peer-to-peer network. In *Int. Conf. on World Wide Web (WWW)*, pages 650–657, 2004.

[CFCS03] Min Cai, Martin R. Frank, Jinbo Chen, and Pedro A. Szekely. Maan : A multi-attribute addressable network for grid information services. In *Int. Workshop on Grid Computing*, pages 184–191, 2003.

[CG99] Surajit Chaudhuri and Luis Gravano. Evaluating top-k selection queries. In *Proceedings of Int. Conf. on Very Large Databases (VLDB)*, pages 397–410, 1999.

[CGM02] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *Int. Conference on Distributed Computing Systems (ICDCS)*, pages 23–33, 2002.

[CGM04a] Surajit Chaudhuri, Luis Gravano, and Amélie Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Transactions on Knowledge Data Engineering (TKDE)*, 16(8) :992–1009, 2004.

[CGM04b] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. In *Int. Work. on Agents and Peer-to-Peer Computing (AP2PC)*, pages 1–13, 2004.

[CLX+08] Bin Cui, Hua Lu, Quanqing Xu, Lijiang Chen, Yafei Dai, and Yongluan Zhou. Parallel distributed processing of constrained skyline queries by filtering. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 546–555, 2008.

[CMH+02] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1) :40–49, 2002.

[CP02] Paolo Ciaccia and Marco Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Transactions on Database Systems*, 27(4) :398–437, 2002.

[CW04] Pei Cao and Zhe Wan. Efficient top-k query calculation in distributed networks. In *Proceedings of Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 206–215, 2004.

[Dat09] http ://www.lina.univ-nantes.fr/projets/dataring/. 2009.

[DFM00] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project : Distributed anonymous storage service. In *In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, 2000.

[DGMY03] Neil Daswani, Hector Garcia-Molina, and Beverly Yang. Open problems in data-sharing peer-to-peer systems. In *Int. Conf. on Database Theory (ICDT)*, pages 1–15, 2003.

[DHA03] Anwitaman Datta, Manfred Hauswirth, and Karl Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *Int. Conference on Distributed Computing Systems (ICDCS)*, pages 76–85, 2003.

[DLAV10a] William Kokou Dedzoe, Philippe Lamarre, Reza Akbarinia, and Patrick Valduriez. Asap top-k query processing in unstructured p2p systems. In *Proceedings of IEEE Int. Conf on Peer-to-Peer Computing (P2P)*, pages 187–196, 2010.

[DLAV10b] William Kokou Dedzoe, Philippe Lamarre, Reza Akbarinia, and Patrick Valduriez. Reducing user waiting time for top-k queries in unstructured p2p systems. In *Actes de la conférence Bases de Données Avancées (BDA)*, pages 1–20, 2010.

[DLAV11] William Kokou Dedzoe, Philippe Lamarre, Reza Akbarinia, and Patrick Valduriez. Efficient early top-k query processing in overloaded p2p systems. In *Proceedings of Int. conf. on Database and Expert Systems Applications (DEXA)*, pages 140–155, 2011.

[DMD+03] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Y. Halevy. Learning to match ontologies on the semantic web. *VLDB Journal*, 12(4) :303–319, 2003.

[dVMNK02] Arjen P. de Vries, Nikos Mamoulis, Niels Nes, and Martin L. Kersten. Efficient k-nn search on vertically decomposed data. In *Proceedings of ACM. Int Conf. on Management of Data (SIGMOD)*, pages 322–333, 2002.

[FJC+07] Guofu Feng, Yuquan Jiang, Guihai Chen, Qing Gu, Sanglu Lu, and Daoxu Chen. Replication strategy in unstructured peer-to-peer systems. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, 2007.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of Symposium on Principles of Database Systems (PODS)*, pages 102–113, 2001.

[FP08] Katerina Fotiadou and Evaggelia Pitoura. Bitpeer : continuous subspace skyline computation with distributed bitmap indexes. In *Int. Workshop on Data Management in Peer- to-Peer Systems (DAMAP)*, pages 35–42, 2008.

[GAA03] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. Approximate range selection queries in peer-to-peer systems. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing multi-feature queries for image databases. In *Proceedings of Int. Conf. on Very Large DataBases (VLDB)*, pages 419–428, 2000.

[GGG+03] P. Krishna Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The impact of dht routing geometry on resilience and proximity. In *Proceedings of Int. Conf. of the ACM Special Interest Group on Data Communication(SIGCOMM)*, pages 381–394, 2003.

[Gnu06] Gnutella, 2006. http ://www.gnutelliums.com/.

[GS04] Jun Gao and Peter Steenkiste. An adaptive protocol for efficient support of range queries in dht-based systems. In *International Conference on Network Protocols (ICNP)*, pages 239–250, 2004.

[GSG02] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *Computer Communication Review*, 32(1) :82, 2002.

[GV09] Marlene Goncalves and Maria-Esther Vidal. Reaching the top of the skyline : An efficient indexed algorithm for top-k skyline queries. In *Proceedings of Int. conf. on Database and Expert Systems Applications (DEXA)*, pages 471–485, 2009.

[Hay]

[HHH+02] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex queries in dht-based peer-to-peer networks. In *Int. Workshop on Peer- to-Peer Systems (IPTPS)*, pages 242–259, 2002.

[HHL+03] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with pier. In *VLDB*, pages 321–332, 2003.

[HIMT] Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. Piazza : data management infrastructure for semantic web applications.

[HJS+03] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet : A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[HKP01] Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. Prefer : a system for the efficient execution of multi-parametric ranked queries. In *Proceedings of ACM. Int Conf. on Management of Data (SIGMOD)*, pages 259–270, 2001.

[HLS06] Katja Hose, Christian Lemke, and Kai-Uwe Sattler. Processing relaxed skylines in pdms using distributed data summaries. In *Proceedings of Int. Conf. on Information and Knowledge Management (CIKM)*, pages 425–434, 2006.

[HRVM07] Rabab Hayek, Guillaume Raschia, Patrick Valduriez, and Noureddine Mouaddib. Design of peersum : A summary service for p2p applications. In *Proceedings of Int. conf. on Advances in Grid and Pervasive Computing (GPC)*, pages 13–26, 2007.

[HRVM08] Rabab Hayek, Guillaume Raschia, Patrick Valduriez, and Noureddine Mouaddib. Summary management in p2p systems. In *Proceedings of Int. Conf on Extending Database Technology (EDBT)*, pages 16–25, 2008.

[HS03] Gisli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4) :517–580, 2003.

[JAB01] M Jovanovic, F Annexstein, and K Berman. Scalability issues in large peer-to-peer networks : a case study of gnutella. technical report, ececs department, university of cincinnati. January 2001.

[JM04] Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Int. Conference on Distributed Computing Systems (ICDCS)*, pages 102–109, 2004.

[JMJV] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[Jov00] M Jovanovic. Modelling large-scale peer-to-peer networks and a case study of gnutella. master's thesis, ececs department, university of cincinnati. January 2000.

[Jxt06] Jxta, 2006. http ://www.jxta.org/.

[Kaz06] Kazaa, 2006. http ://www.kazaa.com/.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Steven E. Czerwinski, Patrick R. Eaton, Dennis Geels, Ramakrishna Gummadi, Sean C. Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Y. Zhao. Oceanstore : An architecture for global-scale persistent storage. In *ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 190–201, 2000.

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2003.

[KGZY02] Vana Kalogeraki, Dimitrios Gunopulos, and Demetrios Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *CIKM*, pages 300–307, 2002.

[KOT04] Nick Koudas, Beng Chin Ooi, Kian-Lee Tan, and Rui Zhang 0003. Approximate nn queries on streams with guaranteed error/performance bounds. In *Proceedings of Int. Conf. on Very Large Databases (VLDB)*, pages 804–815, 2004.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS*, pages 84–95, 2002.

[Len02] Maurizio Lenzerini. Data integration : A theoretical perspective. In *ACM Symp. on Principles of Distributed Computing (PODC)*, pages 233–246, 2002.

[LW06] Xiuqi Li and Jie Wu. Searching techniques in peer-to-peer networks. In *Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks, Eds. W. Zheng, X. Liu, S. Shi, J. Hu and H. Dong). Auerbach Publications*, 2006.

[LW08] Xiuqi Li and Jie Wu. Searching techniques in peer-to-peer networks. 2008.

[MBDH05] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Y. Halevy. Corpus-based schema matching. In *In Proceedings of the Int. Conf. on Data Engineering (ICDE)*, pages 57–68, 2005.

[MK02] Daniel A. Menascé and Lavanya Kanchanapalli. Probabilistic scalable p2p resource location services. *SIGMETRICS Performance Evaluation Review*, 30(2) :48–58, 2002.

[MM02] Petar Maymounkov and David Mazières. Kademlia : A peer-to-peer information system based on the xor metric. In *Int. Workshop on Peer- to-Peer Systems (IPTPS)*, pages 53–65, 2002.

[MNR02] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy : a scalable and dynamic emulation of the butterfly. In *Proceedings of Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192, 2002.

[MP03] Peter McBrien and Alexandra Poulovassilis. Defining peer-to-peer data integration using both as view rules. In *Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, pages 91–107, 2003.

[MPV] Vidal Martins, Esther Pacitti, and Patrick Valduriez. Survey of data replication in p2p systems.

[MTW05] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. Klee : A framework for distributed top-k query algorithms. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 637–648, 2005.

[Nap06] Napster, 2006. http ://www.napster.com/.

[NR99] Surya Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 22–29, 1999.

[NSS03] Wolfgang Nejdl, Wolf Siberski, and Michael Sintek. Design issues and challenges for rdf- and schema-based peer-to-peer systems. *ACM SIGMOD Record*, 32(3) :41–46, 2003.

[NWQ+02] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella : a p2p networking infrastructure based on rdf. In *Int. Conf. on World Wide Web (WWW)*, pages 604–615, 2002.

[OST03] Beng Chin Ooi, Yanfeng Shu, and Kian-Lee Tan. Relational data sharing in peer-based data management systems. *SIGMOD Record*, 32(3) :59–64, 2003.

[PTFS03] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of ACM. Int Conf. on Management of Data (SIGMOD)*, pages 467–478, 2003.

[PZSD96] Michael Persin, Justin Zobel, and Ron Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science (JASIS)*, 47(10) :749–764, 1996.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[RFH+] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of Int. Conf. of the ACM Special Interest Group on Data Communication(SIGCOMM)*.

[RM02] Guillaume Raschia and Noureddine Mouaddib. Seq : a fuzzy set-based approach to database summarization. *Fuzzy Sets and Systems*, 129(2) :137–162, 2002.

[RPT09] Paraskevi Raftopoulou, Euripides G. M. Petrakis, and Christos Tryfonopoulos. Rewiring strategies for semantic overlay networks. *Distributed and Parallel Databases*, 26(2-3) :181–205, 2009.

[Sch08] Konstantin Scherbakov. Search request routing in bittorrent and other p2p based file sharing networks. In *SYRCoDIS*, 2008.

[SMK+01] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. In *Proceedings of Int. Conf. of the ACM Special Interest Group on Data Communication(SIGCOMM)*, pages 149–160, 2001.

[SSDN02] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. Hypercup. 2002.

[SW07] Stefan Schmid and Roger Wattenhofer. Structuring unstructured peer-to-peer networks. In *Proceedings of IEEE Int. Conf. on High Performance Computing (HiPC)*, pages 432–442, 2007.

[TIM+03] Igor Tatarinov, Zachary G. Ives, Jayant Madhavan, Alon Y. Halevy, Dan Suciu, Nilesh N. Dalvi, Xin Dong, Yana Kadiyska, Gerome Miklau, and Peter Mork. The piazza peer data management project. *SIGMOD Record*, 32(3) :47–52, 2003.

[TKLB07] Wesley W. Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P. Buchmann. Bubblestorm : resilient, probabilistic, and exhaustive peer-to-peer search. In *SIGCOMM*, pages 49–60, 2007.

[TP03] Peter Triantafillou and Theoni Pitoura. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In *Int.Workshop on Databases, Information Systems and Peer-to-Peer Computing*, pages 169–183, 2003.

[TR03a] Dimitrios Tsoumakos and Nick Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *Peer-to-Peer Computing*, pages 102–109, 2003.

[TR03b] Dimitrios Tsoumakos and Nick Roussopoulos. Adaptive probabilistic search in peer-to-peer networks. In *technical report, CS-TR-4451*, 2003.

[TR03c] Dimitrios Tsoumakos and Nick Roussopoulos. A comparison of peer-to-peer search methods. In *WebDB*, pages 61–66, 2003.

[TRV98] Anthony Tomasic, Louiqa Raschid, and Patrick Valduriez. Scaling access to heterogeneous data sources with disco. *IEEE Trans. Knowl. Data Eng.*, 10(5) :808–823, 1998.

[TV01a] Asterio K. Tanaka and Patrick Valduriez. The ecobase project : Database and web technologies for environmental information systems. *SIGMOD Record*, 30(3) :70–75, 2001.

[TV01b] Asterio K. Tanaka and Patrick Valduriez. The ecobase project : Database and web technologies for environmental information systems. *SIGMOD Record*, 30(3) :70–75, 2001.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997.

[VDKV07] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Michalis Vazirgiannis. Skypeer : Efficient subspace skyline computation over distributed data. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 416–425, 2007.

[VDNV08] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Michalis Vazirgiannis. On efficient top-k query processing in highly distributed environments. In *Proceedings of ACM. Int Conf. on Management of Data (SIGMOD)*, pages 753–764, 2008.

[VLCV09a] Anthony Ventresque, Philippe Lamarre, Sylvie Cazalens, and Patrick Valduriez. Représentation optimiste de contenus dans les systèmes p2p. In *Actes de la conférence Bases de Données Avancées (BDA)*, pages 1–20, 2009.

[VLCV09b] Anthony Ventresque, Philippe Lamarre, Sylvie Cazalens, and Patrick Valduriez. Représentation optimiste de contenus dans les systèmes p2p. In *Actes de la conférence Bases de Données Avancées (BDA)*, pages 1–20, 2009.

[VP04] Patrick Valduriez and Esther Pacitti. Data management in large-scale p2p systems. In *Springer Int. Conf. on High Performance Computing for Computational Science (VECPAR)*, pages 104–118, 2004.

[W3C01] W3c : World wide web consortium on semantic web activities, 2001. http ://www.w3.org/2001/sw.

[WA91] Annita N. Wilschut and Peter M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *Int. Conf. on Parallel and Distributed Information Systems (PDIS)*, pages 68–77, 1991.

[WZF+06] Ping Wu, Caijie Zhang, Ying Feng, Ben Y. Zhao, Divyakant Agrawal, and Amr El Abbadi. Parallelizing skyline queries for scalable distribution. In *Prooceeding of Int. Conf. on Extending Database Technology (EDBT)*, pages 112–130, 2006.

[YGM02] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Int. Conference on Distributed Computing Systems (ICDCS)*, pages 5–14, 2002.

[YLLL10] Mao Ye, Xingjie Liu, Wang-Chien Lee, and Dik Lun Lee. Probabilistic top-k query processing in distributed sensor networks. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 585–588, 2010.

[YM07] Man Lung Yiu and Nikos Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *Proceedings of Int. Conf. on Very Large Databases (VLDB)*, pages 483–494, 2007.

[Zad75] Lotfi A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning - i. *Information Systems*, 8(3) :199–249, 1975.

[ZHS+04] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry : a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications (JSAC)*, 22(1) :41–53, 2004.

[ZTZ07] Keping Zhao, Yufei Tao, and Shuigeng Zhou. Efficient top-k processing in large-scaled distributed environments. *Data and Knowledge Engineering*, 63(2) :315–335, 2007.

[ZTZ09] Lin Zhu, Yufei Tao, and Shuigeng Zhou. Distributed skyline retrieval with low bandwidth consumption. *IEEE Transactions on Knowledge Data Engineering (TKDE)*, 21(3) :384–400, 2009.

[ZZSL07] Haoxiang Zhang, Lin Zhang, Xiuming Shan, and Victor O. K. Li. Probabilistic search in p2p networks with high node degree variation. In *ICC*, 2007.

# Traitement de Requêtes Top-k dans les Communautés Virtuelles P2P de Partage de Données

Kokou DEDZOE

**Résumé**

Dans les communautés virtuelles pair-à-pair (P2P) de partage de données, les requêtes top-$k$ présentent deux avantages principaux. Premièrement, elles permettent aux participants de qualifier les résultats de leurs requêtes par rapport aux données partagées dans le système et ceci en fonction de leurs préférences individuelles. Deuxièmement, elles évitent de submerger les participants avec un grand nombre de réponses. Cependant, les techniques existantes pour le traitement des requêtes top-$k$ dans un environnement complètement distribué présentent l'inconvénient d'un temps d'attente important pour l'utilisateur. Ce temps d'attente est encore très long plus le système est surchargé. Dans un premier temps, nous revisitons le problème du temps d'attente de l'utilisateur dans le traitement des requêtes top-$k$ en introduisant deux nouvelles mesures : le *temps de stabilisation* et la *qualité restante cumulée*. En complément des mesures existantes, elles permettent de qualifier plus précisément le comportement d'un algorithme top-$k$. Dans un deuxième temps, nous proposons une famille d'algorithmes (ASAP), permettant de retourner à l'utilisateur les résultats de bonne qualité le plus tôt possible. Enfin, nous nous intéressons au problème du traitement des requêtes top-$k$ dans le cadre des systèmes P2P surchargés, particulièrement critique pour les solutions classiques, en proposant une nouvelle approche (QUAT). Cette solution fait usage de descriptions synthétiques des données des pairs pour permettre aux pairs de traiter en priorité les requêtes pour les quelles ils peuvent fournir des résultats de bonne qualité.

**Mots-clés :** Communautés virtuelles, Systèmes pair-à-pair, Traitement de requêtes top-$k$, Temps de réponse, Temps de stabilisation, Qualité restante cumulée, Description synthétique d'un pair.

## Top-k Query Processing in P2P Data Sharing Virtual Communities

**Abstract**

Top-$k$ queries have two main advantages for peer-to-peer (P2P) data sharing virtual communities. First, they allow participants to rank the results for their queries based on the existing data in the system as well as on their own preferences. Second, they avoid overwhelming participants with too many results. However, existing top-$k$ query processing techniques for P2P systems make users suffer from long waiting times. This becomes even more problematic in overloaded P2P systems. In this thesis, we first revisit the top-$k$ query processing problem and introduce two new measures: the *stabilization time* and the *cumulative quality gap*. These two novel measures, in addition to existing measures, allow for better evaluating the behavior of top-$k$ query processing techniques. We then propose a new family of top-$k$ query processing techniques (ASAP) that allows to return high quality results as soon as possible. Finally, we study the problem of top-$k$ query processing in overloaded systems. As a result, we propose a new approach, called QUAT, that relies on synthetic data descriptions of peers in order to allow peers to prioritize queries for which they can provide high quality results.

**Keywords:** Virtual communities, Peer-to-Peer systems, Top-$k$ query processing, Response time, Stabilization time, Cumulative quality gap, Peer synthetic description.

**acm Classification**

**Categories and Subject Descriptors :** H.2.4 [**Database Management**]: Systems—*Distributed databases, Query processing.*

Discipline : INFORMATIQUE
Specialité : Bases de Données

Laboratoire : **L**aboratoire d'**I**nformatique de **N**antes **A**tlantique.
cnrs fre 2729. 2, rue de la Houssinire, bp 92 208 – 44 322 Nantes, Cedex 3.