République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche scientifique

Université IBN KHALDOUN - Tiaret
Faculté des Sciences et des Sciences de l'Ingénieur
Département Informatique

ÉCOLE DOCTORALE STIC

Sciences et Technologies de l'Information et de la Communication

Option : Systèmes d'Information et de Connaissance
- SIC -

Mémoire de Magister

Présenté Par :

Abdelkader ALEM

Thème

Contribution à l'Étude de Requêtes à Préférences dans un Contexte Distribué

Jury

Président :

Youcef. DAHMANI — Maître de conférences, Université d'Ibn Khaldoun, Tiaret

Examinateurs :

Mohammed Amine. CHIKH — Maître de conférences, Université d'Abou Beker Belkaid, Tlemcen

Abdelhafidh. CHADLI — Chargé de cours, Université d'Ibn Khaldoun, Tiaret

Directeur du Mémoire :

Allel HADJALI — Doctorat d'État, Maître de Conférences
IRISA/ENSSAT, Université Rennes 1

ANNEE 2009-2010

Contribution à l'Étude des Requêtes à Préférences dans un Contexte Distribué

Résumé. Les requêtes à préférences est un thème de recherche qui a suscité un intérêt croissant ces dernières années. Dans les systèmes d'interrogation flexible, les utilisateurs peuvent introduire des "préférences" dans leurs critères exprimés d'une manière souple en utilisant des prédicats flous. De telle requêtes "graduelles" retournent alors des données plus au moins pertinentes en fonction de leur satisfaction vis-à-vis de ces préférences. Ce mémoire étudie plus particulièrement l'interrogation flexible dans un environnement P2P. L'idée suggérée est d'envoyer la requête seulement aux sources de données qui sont susceptibles de fournir les meilleures réponses à la requête, ce qui nécessite d'évaluer chaque source au moyen de son résumé de données. Deux démarches sont proposées, la première permet de construire un index global de routage qui décrit les données des différentes sources, afin de trouver l'ensemble des pairs pertinents pour la requête posée. La seconde présente un algorithme de recherche des réponses satisfaisant le mieux cette requête.

Mots-clés. Requêtes à préférences, Interrogation flexible, Système P2P, Résumés linguistiques de données.

Abstract. Preferences queries is a topic which has gained a great interest in the last years. In database applications, fuzzy querying systems allow for adding preferences in queries criteria. Preferences are expressed in a flexible way by using fuzzy predicates. In this context, the result of a query is no longer a flat set of elements, but those which more or less match the predicates and they are assigned a degree of satisfaction reflecting the level of match. This work deals with the problem of flexible queries evaluation in a P2P context. The suggested idea is to send the query only to the data sources which are likely to provide the best answers to the query, which requires a manner of evaluating each source by means of its data summary. Two steps are proposed. The first one makes possible to build a routing index which describes data of the various sources, in order to find the relevant peers for the submitted query. The second advocates an efficient algorithm of research of the answers satisfying the query at hand with a better satisfaction degree.

Keywords. Preferences queries, flexible database querying, P2P system, linguistic summaries.

Remerciements

Je tiens à remercier.

Monsieur Y. DAHMANI, Maître de conférences, Université d'Ibn Khaldoun de Tiaret, qui me fait l'honneur de présider ce jury.

Monsieur M. A. CHIKH, Maître de conférences, Université d'Abou Beker Belkaid de Tlemcen, et Monsieur A. CHADLI, Chargé de cours, Université d'Ibn Khaldoun de Tiaret, pour l'intérêt qu'ils ont porté à ce mémoire en acceptant d'en être examinateurs.

Monsieur A. CHIKH, sans qui cette école doctorale n'aurait jamais pu commencer. Je le remercie pour le grand effort qu'a fait pour que notre école doctorale réussisse.

Monsieur A. HADJALI, Maître de conférences à l'IRISA/ENSSAT, Université Rennes 1, pour avoir encadré ce mémoire. Qu'il soit remercié pour son aide, ses conseils précieux, ses idées, ses encouragements et sa patience.

Enfin, je remercie toute ma famille, mes amis et collègues pour leur encouragement et leur soutien.

Table des matières

Introduction Générale

Les systèmes de Gestion de Bases de Données Relationnelles (SGBDR) sont devenus, sans conteste, le noyau de tout système informatique. Cependant, la diversification des applications des bases de données a montré les limites des SGBDR notamment sur le plan de modélisation des données imprécises et de l'interrogation flexible.

Le contexte scientifique de ce mémoire est donc à l'intersection des domaines suivants : “les requêtes à préférences et l'interrogation flexible”, “les résumés linguistiques de données” et “les système P2P”. Dans cette introduction, nous présentons ces différents domaines, le cheminement qui nous a amené à les réunir dans ce travail et nous situons notre contribution.

1. Contexte

Requête à préférences et Interrogation flexible

Le thème principal de ce mémoire est " l'évaluation des requêtes à préférences dans un contexte distribué".

Les requêtes à préférences est un thème de recherche qui a suscité un intérêt croissant ces dernières années, voir par exemple [BKS 01] [CHO 03] [LTL 06] [HKD 08]. En particulier, un des formalismes les plus utilisés pour représenter les préférences des utilisateurs est les requêtes dites "Skyline" [BKS 01]. Ce type de requêtes permet de retourner un ensemble d'objets qui ne sont dominés par aucun autre objet de la base de données. Ce type de requête a reçu une attention considérable dans la communauté des bases de données. Cependant, la plupart de ces travaux ont été réalisés dans un environnement centralisé. Par ailleurs, une autre approche, dite "Interrogation flexible", permet aussi d'exprimer des requêtes à préférences d'une manière plus générale. Divers travaux portant sur l'expression et l'interprétation des requêtes flexibles dans un contexte centralisé ont été proposés dans la littérature [BOP 95, CON 99, CRO 03].

Le principe de "l'interrogation flexible" se distingue par les techniques utilisées pour exprimer les préférences et les combinées, et pour représenter les réponses (sous forme d'un ensemble d'éléments discriminés plutôt qu'un ensemble plat). Plusieurs systèmes ont été proposés dans la littérature, il a été montré dans [BOP 92] qu'une approche basée sur la "théorie des ensembles flous" les unifie. Dans les modèles basés sur la théorie des ensembles flous, les requêtes utilisent des prédicats flous qui sont plus ou moins satisfaits par les données. Ainsi le résultat d'une requête est un ensemble flou contenant des éléments plus ou moins satisfaisants.

Par exemple, supposons qu'un utilisateur recherche une voiture d'occasion vièlle de 1 ou 2 ans (avec une préférence de 1 an). Un système d'interrogation flexible distinguera trois types de données : les voitures de 1 an (qui satisfont tout à fait l'utilisateur), les voitures de 2 ans (qui satisfont moyennement l'utilisateur) et les voitures plus anciennes (qui ne satisfont pas du tout l'utilisateur). Cette distinction "qualitative" caractérise le principe de l'interrogation flexible.

Les résumés linguistiques de données

Ces dernières années, la taille, la diversité et la distribution des bases de données ont crû très fortement. Une conséquence est que la quantité de données disponible dépasse largement notre capacité d'appréhension. Il est aussi de plus en plus difficile de pouvoir retrouver toute, et uniquement, l'information pertinente dans les bases de données (BD) de taille importante.

Ainsi une classe de méthodes, dédiées à la construction de résumés de données, a vu le jour. Leur principe consiste essentiellement à réaliser une approximation des données auxquelles elles s'appliquent, c'est-à-dire les enregistrements de bases de données. Ces méthodes s'inscrivent notamment dans le cadre des résumés linguistiques proposés par Yager [YAG 91]. Le modèle SaintEtiQ [RNM 02] fait partie de ces méthodes. L'utilisation de résumés accélère l'accès aux données et conduit à des réponses plus compactes, plus facilement appréhendables et, en contrepartie, moins précises. Le principe de ces méthodes est présenté dans le chapitre 4.

Les systèmes P2P

Au cours de la dernière décennie, le paradigme des systèmes distribués, en particulier les systèmes dits P2P, est devenu très populaire en permettant le partage des ressources et

l'échange d'information entre des millions d'utilisateurs. De plus, les P2P offrent de nombreux autres avantages comme l'auto-organisation, l'autonomie et le "passage à l'échelle". Parmi les systèmes P2P les plus connus, on peut citer Gnutella [Gnu], Napster [Nap]. Cependant, un des problèmes majeurs dont souffrent ces systèmes est la localisation des sources (ou pairs) pertinentes (c.-à-d., celles contenant les réponses qui satisfont au mieux les besoins de l'utilisateur). Ce problème a fait l'objet de plusieurs études et de nombreuses techniques efficaces ont été proposées pour la localisation de données pertinentes dans les systèmes P2P. Les index de routage [CGM 02] font partie des techniques proposées dans la littérature pour une évaluation efficace des requêtes dans les réseaux P2P (en évitant la stratégie d'inondation de tout le réseau). Rappelons qu'un index de routage est une structure de données (avec un ensemble d'algorithmes) qui, étant donnée une requête sur un pair, retourne la liste de pairs voisins ordonnés selon leur pertinence à la requête considérée. Ces index permettent donc de ne renvoyer la requête qu'aux pairs qui sont les plus probables à fournir de réponses. On distingue deux catégories d'index : les index locaux et globaux. Dans Gnutella [Gnu], chaque nœud maintient un index local sur les données qu'il possède (un nœud diffuse la requête à tous ses voisins dans le réseau). Un index global peut être centralisé ou distribué, par exemple, Napster [Nap] stocke un index global de toutes les données sur un pair central (ou super pair), alors que Chord [SMN 01] distribue un index global sur tous les nœuds du réseau.

2. Contribution

Le travail présenté dans ce mémoire traite le problème d'évaluation des requêtes flexibles dans un contexte distribué, illustré par un système P2P. L'idée suggérée est de n'envoyer la requête qu'aux sources de données qui sont susceptibles de fournir les meilleures réponses à la requête, ce qui nécessite d'évaluer chaque source au moyen de son résumé de données. Deux contributions majeures sont proposées : La première concerne la construction d'un index de routage global (et distribué), qui décrit les données des différentes sources, afin de trouver l'ensemble des pairs pertinents pour la requête (flexible) posée. La seconde décrit un algorithme de recherche des réponses satisfaisant au mieux cette requête.

3. Structure du mémoire

Le mémoire est structuré en quatre chapitres organisés comme suit.

Le premier chapitre aborde les principales notions sur les requêtes à préférences et se focalise particulièrement sur les requêtes Skyline. D'autre part, il étudie les différentes approches proposées pour le traitement des requêtes Skyline dans un contexte centralisé.

Le chapitre 2 présente une vue globale sur les systèmes P2P et rappelle quelques notions nécessaires à l'entreprise. Le problème de recherche dans ces systèmes est également abordé.

Dans le chapitre 3, on s'intéresse aux principales méthodes d'évaluation des requêtes Skyline dans le cas des bases de données distribuées. Pour chaque méthode, on décrit explicitement son principe et un exemple illustratif est proposé.

Le chapitre 4 est consacré à la démarche proposée pour évaluer les requêtes flexibles dans un environnement P2P, en premier, on présente l'approche de construction d'un index de routage dans un système P2P. Ensuite l'algorithme d'évaluation des requêtes flexibles est expliqué.

Une annexe est fournie à la fin du manuscrit qui rappelle quelques notions de base sur la théorie des ensembles flous.

CHAPITRE 1

Requêtes à Préférences

Introduction

Habituellement, dans des systèmes de gestion de base de données (BD), une grande quantité de données est stockée (BD de taille de plus en plus astronomique). L'interrogation de ces systèmes est basée sur les requêtes booléennes (tous les n-uplets de la BD qui satisfont les conditions de la requête sont retournées). Ceci conduit souvent au problème de réponses pléthoriques et non ordonnées où l'utilisateur ne peut pas sélectionner les éléments désirés (suivant ses préférences). Donc il est nécessaire que les systèmes d'interrogation soient performants en termes de qualité d'information délivrée (en retournant les éléments satisfaisant le mieux les préférences de l'utilisateur).

Généralement, on distingue deux approches pour l'expression des requêtes intégrant les préférences de l'utilisateur. Une approche quantitative où les préférences sont spécifiées indirectement en utilisant des fonctions de préférence ou de score (**scoring function**). Une deuxième approche explicite (qualitative) où les préférences sont définies en comparant les n-uplets deux à deux (relations de préférences binaires).

L'objectif de ce chapitre est de mettre en évidence les caractéristiques de l'approche qualitative et plus précisément les requêtes dites skyline du point de vue expression et implémentation (différents algorithmes).

1.1 Définitions

1.1.1 Requêtes à préférences

Des requêtes qui expriment (intègrent) les préférences des utilisateurs et retournent uniquement les éléments désirés.

1.1.2 Relations de Préférence

Définition : Soit une relation de schéma R (A_1, .., A_k) tel que U_i, $1 \leq i \leq k$, est le domaine de l'attribut A_i, une relation Ω est une relation de préférence sur R si elle constitue un sous-ensemble de $(U_1 x \ldots x U_k) \times (U_1 x \ldots x U_k)$

On dit que le n-uplet $\mathbf{t_1}$ *domine (ou est préférable à)* le n-uplet $\mathbf{t_2}$ dans le contexte de la relation Ω. **Si** $t_1 \; \Omega \; t_2$ **alors** t_1 est au moins aussi bon que t_2 pour tous les attributs et meilleur que t_2 dans un attribut. Deux tuples a et b pour lesquelsles ni a Ω b, ni b Ω a sont incomparables (indifférents).

1.1.3 Propriétés d'une relation de préférence : Les propriétés suivantes sont vérifiées :

- i) Irréfléxivité : $\forall$ x ; $x \not\Omega x$
- ii) Assymétrie : $\forall$ x, y ; $x \; \Omega \; y \Rightarrow y \not\Omega \; x$
- iii) Transitivité : $\forall$ x, y, z ; $(x \; \Omega \; y \wedge y \; \Omega \; z) \Rightarrow x \; \Omega \; z$
- iv) Transitivité Négative : $\forall$ x, y, z ; $(x \not\Omega \; y \wedge y \not\Omega z) \Rightarrow x \not\Omega \; z$
- v) Connectivité : $\forall$ x, y ; $x \; \Omega \; y \vee y \; \Omega \; x \vee x = y$

Une relation est un ordre partiel strict (OPS) si elle est (i) + (iii) + (ii), elle est un ordre faible (OF) si elle est (iv) + OPS et elle est un ordre total (OT) si elle est (v) + OPS.

1.2 Approche Quantitative

Cette approche est fondée sur le principe des fonctions de scores où les préférences sont exprimées par des valeurs numériques (on associe une valeur à chaque n-uplet), et pour chaque n-uplet les scores élémentaires de ces attributs sont agrégés (combinés) en utilisant une fonction de score dite aussi fonction d'utilité ou de préférence.

1.2.1 Définition

Soit une relation R avec N n-uplets t_1, t_2, …t_N et M attribues A_1, A_2, …, A_M. Le score de chaque n-uplet t_i dans la relation R, pour une requête Q= {$A_1 = q_1$, $A_2 = q_2$, …, $A_M = q_M$}, est une fonction du score de chaque attribut individuel A_j avec la valeur cible q_j (j=1, M).

$$\text{Score } (Q, t) = f(s_1, s_2, \ldots, s_M) = \sum_{j=1}^{M} w_j \cdot s_j$$

- La fonction d'agrégation f permet de calculer un score général pour chacun des n-uplets.
- s_j = Score (q_j, t [A_j]) ∈ [0, 1] indique de combien la valeur t [A_j]) est reliée (proche) à q_j (t [A_j] désigne la valeur du n-uplet t pour l'attribut A_j)
- w_j est poids indiquant l'importance relative de l'attribut A_j dans la requête Q

Un n-uplet $\mathbf{t_i}$ est préféré à un autre $\mathbf{t_j}$ si et seulement si :

$$\textbf{Score } (t_i) > \textbf{Score } (tj) \Leftrightarrow f(s_1^i,\ldots,s_m^i) > f(s_1^j,\ldots,s_m^j).$$

Les « **Top K Queries** » représentent un bon exemple de cette approche, elles visent à retourner les objets avec les scores globaux les plus élevés, la réponse consiste en un ensemble ordonné qui contient les K meilleurs n-uplets, pour plus de détails, voir [MBG 04].

1.3 Approche Qualitative

L'approche qualitative est plus générale que L'approche quantitative. Elle définit directement les préférences, en utilisant des relations de préférences binaires.

Afin de comparer les n-uplets, et si on considère plusieurs préférences atomiques, les valeurs de chaque attribut sont comparées les unes par rapport aux autres, dans ce cas seul un ordre partiel peut être défini sur les n-uplets. En particulier, l'ordre de **Pareto** peut être utilisé.

1.3.1 Ordre de Pareto

On souhaite comparer deux vecteurs de valeurs, V et U tel que V = (v_1,v_2,…,v_n), U = (u_1, u_2,…,u_n). L'ordre de Pareto est défini par la formule suivante :

$$V >_{Pareto} U \Leftrightarrow (\forall i\ v_i \geq u_i \text{ et } \exists j\ v_j > u_j).$$

La préférence « *une voiture de couleur verte et qui consomme peu* », est définie sur les attributs couleur et consommation. Chaque n-uplet est associé à un vecteur composé d'une couleur et d'une consommation. Par exemple V_1 est associé à (verte, 8), V_2 est associé à (verte, 9) et V_3 est associé à (noire, 7). L'ordonnancement des vecteurs se base sur les relations de préférences atomiques et l'on peut utiliser une relation d'ordre partiel définie par l'ordre de Pareto. Pour cet exemple V_1 est préféré à V_2 et V_3 ne peut pas être comparé à V_1 ou V_2.

Les « **skyline Queries** (section 1-4) », «**Preference SQL** » et « **l'approche logique** » font partie de cette famille d'approches.

1.4 Requêtes Skyline

Les requêtes skyline ont été, de plus en plus, employées dans la prise de décision multicritère et les applications d'extraction de données.

L''idée principale dans ce type de requête est de rechercher un ensemble de points non-dominés dans un ensemble de points potentiellement grand. Un point est dit non-dominés s'il est au moins bon dans toutes les dimensions et meilleur dans au moins une dimension.

Considérons l'exemple d'une personne désirant choisir un hôtel pour son voyage. L'ensemble des hôtels est décrit dans le tableau 1-1.

Tableau 1-1 Exemple des hôtels avec leurs prix et distance

Hôtel	*prix*	*distance*
H1	90$	100m
H2	100$	200m
H3	80$	400m
H4	70$	650m
H5	100$	900m
H6	80$	200m
H7	60$	500m
H8	30$	400m
H9	20$	300m
H10	33$	800m
H11	15$	900m
H12	10$	1000m
H13	20$	600m

Si la personne est intéressée par le prix et la distance par rapport à la plage, elle peut utiliser une requête skyline pour rechercher les hôtels qui sont près de la plage et ayant le plus bas prix. Le résultat de cette requête inclura H1, H9 et le skyline de cet ensemble est S= {H1, H6, H9, H11, H12}. Ces hôtels ne sont dominés par aucun autre hôtel. Par conséquent, ils seront intéressants à l'utilisateur. H2, et H5 sont dominés par {H1, H6}. Alors que H1 est meilleur que H2, H5 dans les deux dimensions (prix, distance). De même, H4, H3, H7, H8, H10 et H13 sont dominés par H9. En éliminant les hôtels qui sont complètement dominés, le processus décisionnel peut être facilité à l'utilisateur.

1.4.1 Définitions

1.4.1.1 Principe de dominance

Soit un ensemble D d'objets, et n fonctions de score $S = \{s_1, s_2, \ldots, s_n\}$ et soient d_1 et d_2 deux éléments de D, une relation ***domine*** (dont le symbole est $\mathbf{\Omega_S}$) peut être définie comme suit:

d_1 ***domine*** ($\mathbf{\Omega_S}$) d_2 si et seulement si

(i) $\forall\ s \in S,\ s(d_1) \geq s(d_2)$, et

(ii) $\exists\ s \in S,\ s(d_1) > s(d_2)$.

C'est-à-dire, d_1 ***domine*** d_2 si d_1 est au moins aussi bon que d_2 pour toutes les fonctions de score et meilleur que d_2 dans une fonction de score, deux objets a_1 et a_2 pour lesquels ni $a_1\ \mathbf{\Omega_S}\ a_2$, ni $a_2\ \mathbf{\Omega_S}\ a_1$ sont appelés incomparables.

1.4.1.2 Propriétés de la relation de dominance (Ω_S)

1. ***Transitive***

La transitivité signifie que si un point domine un autre point et ce point domine un troisième point alors le troisième point est également dominé par le premier point. Considérons trois points, a, b et c, le point a domine b et b domine c alors a domine également c.

2. ***Asymétrique***

La relation de dominance est asymétrique, si a domine b alors (b ne domine pas a). C'est clair que a Ω_S b et b Ω_S a conduisent à une contradiction d'un côté $\forall$ s$\in$S, s(a) $\geq$ s(b) et de l'autre côté $\exists$ s $\in$ S, s(b) > s(a).

3. ***Irreflexive***

Aucune fonction de score **s** ne peut exister pour s(a) > s(a) et la deuxième condition de principe de dominance ne peut être vérifié**.**

1.4.1.3 Définition de Skyline

L'ensemble des éléments maximums de D par rapport à Ω_S est appelé skyline de D par rapport à S (dont le symbole est $\boldsymbol{S_S}$) et il est défini comme suit:

S_S (D) = {d $\in$ D / $\nexists$ $d_1 \in D$, d_1 Ω_S d}

S_S (D) : contient les éléments qui ne sont dominés par aucun élément sur toutes les dimensions.

1.4.1.3 Propriétés de skyline

- La taille de skyline augmente avec l'augmentation de la dimensionnalité.
- La taille de skyline augmente également avec l'augmentation du nombre de points dans la base de données.
- La taille de skyline dépend également de la distribution des valeurs de points dans la BD.
- Une autre propriété du skyline d'un ensemble D, est celle pour n'importe quelle fonction de score monotone S.

$$S: \ D \rightarrow \Re.$$

Si p $\in$ D maximise (minimise) cette fonction S, alors p appartient au skyline (votre objet préféré est toujours dans le skyline).

- Les points identiques sont tous des points de skyline.

1.4.2 Expression des requêtes skyline

Borzsonyi, Kossmann et Stocker **[**BKS 01**]** étaient les premiers qui ont discuté l'idée du problème de vecteur maximum dans le contexte de base de données. Ils ont proposé une extension de SQL. Le bloc SELECT du SQL est enrichi par une clause **SKYLINE OF** facultatif décrivant les attributs (dimensions) qui devraient être considérés dans le calcul de skyline.

SELECT ……. **FROM**……..**WHERE**
GROUP BY ……. **HAVING**…….
SKYLINE OF [DISTINCT] d_1 [MIN | MAX | DIFF],…, d_m [MIN | MAX | DIFF]
ORDER BY…

- d_1,…, d_m sont les dimensions de skyline (par exemple: prix, distance,…).
- *[MIN | MAX | DIFF]* spécifie si la valeur dans cette dimension doit être minimisée, maximisée ou simplement indifférente.
- *DISTINCT* est facultatif et indique comment traiter les duplications (points identiques). Si p=q pour tous i=1…n alors p et q sont indifférents et peuvent tous les deux faire partie de skyline si aucun DISTINCT ni indiqué, sinon p ou q est maintenu.
- *SKYLINE OF* : la clause SKYLINE OF décrit pour chaque dimension si elle doit être maximisée, minimisée ou être indifférente. Cette clause est exécutée après le: **SELECT** ……. **FROM**……..**WHERE….GROUP BY…..HAVING** de la requête.

La clause SKYLINE OF permet d'obtenir tous les n-uplets intéressants (n-uplets qui ne sont pas dominés par aucun autre n-uplets).

Par exemple soient deux points p et q de dimension n

$p = (p_1, \ldots, p_k, p_{k+1}, \ldots, p_l, p_{l+1}, \ldots, p_m, \ldots)$

$q = (q_1, \ldots, q_k, q_{k+1}, \ldots, q_l, q_{l+1}, \ldots, q_m, \ldots)$

p domine q pour la requête skyline :

SKYLINE OF dim_1 *min*, …, dim_k *min*
dim_{k+1} *max*, …, dim_l *max*
dim_{l+1} *diff*, …, dim_m *diff*

Si les conditions suivantes sont satisfaites :

$$p_i \leq q_i \quad \forall\ i = 1, \ldots, k$$
$$p_i \geq q_i \quad \forall\ i = k+1, \ldots, l$$
$$p_i = q_i \quad \forall\ i = l+1, \ldots, m$$

Remarque : Les autres dimensions dim_{m+1}……dim_n sont non pertinentes pour le calcul de skyline.

1.4.3 Algorithmes pour le Calcul de Skyline

Dans cette section nous présentons les différents algorithmes et approches de calcul de skyline.

1.4.3.1 L'algorithme standard

Le calcul de skyline par l'algorithme standard est simple, il suffit simplement de comparer chaque n-uplet à tous les autres n-uplets de la BD sauf à lui-même, en respectant les propriétés de la relation de dominance à savoir, deux n-uplets identiques font partie de skyline tous les deux.

Cet algorithme est le plus facile et il est très performant dans le cas de petites bases de données (ou petites dimensions), mais il n'est pas possible de comparer chaque élément à tous les autres dans des applications modernes qui utilisent des bases de données astronomiques (volumineuses).

Le skyline de l'ensemble de données de notre premier exemple (le tableau 1-1) est montré sur la Figure 1-1 (les points de skyline (les hôtels) sont marqués en rouge)

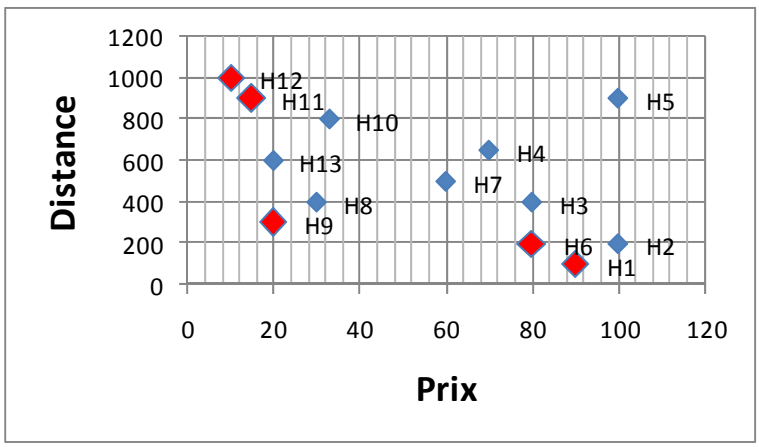


Figure 1-1 Le Skyline des Hôtels

1.4.3.2 L'algorithme BNL (Block-Nested-Loops)

Cet algorithme est basé sur le premier algorithme. L'idée est de maintenir une fenêtre (window) de n-uplets incomparables dans la mémoire centrale et de comparer chaque n-uplet de la BD aux n-uplets de la fenêtre.

Quand un n-uplet **t** est lu de l'entrée, t est comparé à tous les n-uplets de la fenêtre. En se basant sur cette comparaison, **t** est retiré, placé dans la fenêtre ou dans une file temporaire. Trois cas peuvent se produire :

1. Si **t** est dominé par un n-uplet de la fenêtre alors **t** est supprimé et ne sera pas considéré dans la prochaine itération (c'est clair que **t** n'a pas besoin d'être comparé à tous les éléments de la fenêtre).
2. Si **t** domine un ou plusieurs éléments de la fenêtre, dans ce cas, ces éléments sont supprimés de la fenêtre et ne seront pas considérés dans de futures itérations. **t** est alors inséré dans la fenêtre
3. Si **t** est incomparable avec tous les n-uplets de la fenêtre, **t** est alors insérée s'il y a de la place dans la fenêtre, autrement **t** est écrit dans une file temporaire sur le disque.

A la fin de chaque itération nous pouvons faire sortir les n-uplets de la fenêtre qui ont été comparés à tous les autres n-uplets de la BD, ces n-uplets ne sont dominés par aucun autre n-uplet (c'est à dire: ils font partie du skyline). Les autres n-uplets de la fenêtre peuvent sortir (si on ne les élimine pas) pendant les prochaines itérations.

Afin de garder la trace quand un n-uplet de la fenêtre peut sortir, on dénote par un simple compteur (**timestamp**) chaque n-uplet dans la fenêtre et dans la file temporaire. Ce compteur enregistre dans quel ordre les n-uplets ont été insérés dans la fenêtre et la file. Si un n-uplet est lu de la file temporaire avec un timestamp k, tous les n-uplets de la fenêtre qui ayant un timestamp inférieur à k peuvent être sortir. Ce compteur (timestamp) garantit que l'algorithme se termine, et que deux n-uplets ne sont jamais comparés deux fois.

Exemple (suite): On applique l'algorithme BNL à notre exemple avec une fenêtre de taille 3. Le Tableau 1.2 présente toutes les étapes pour calculer le skyline de notre exemple. Le nombre entre parenthèses après chaque point est le *timestamp* du point.

Premièrement la fenêtre est vide et H1 est inséré, l'hôtel suivant H2 est éliminé immédiatement (H2 est dominé par H1) et après H3 et H4 sont insérés dans la fenêtre (ne sont pas dominés par aucun autre point de la fenêtre). Ensuite H6 est comparé aux points de la

fenêtre, on trouve que H6 domine H3 alors H3 est supprimé et H6 est inséré dans la fenêtre. Le point H7 n'est dominé par aucun autre point de la fenêtre et comme il n y a pas de places dans la fenêtre, l'algorithme crée une file temporaire et H7 est inséré (le **timestamp** est égale a 7), de même H8 est inséré dans la file, H4 est supprimé de la fenêtre et H9 est inséré (H9 domine H4). H10, H13 sont éliminés et H11, H12 sont insérés dans la file temporaire.

A la fin de cette itération les points de skyline sont H1, H6 et H9 et la file temporaire contient H7, H11 et H12. Après les points de la fenêtre qui ont été comparés à tous les points de l'ensemble (qui ont un timestamp inferieur à 7), dans ce cas H1 et H6, sont sorties (insérés dans le skyline) et l'algorithme continue le traitement des éléments de la file temporaire de la même manière jusqu'à le vidage de la file temporaire. Le résultat final de skyline est S= {H1, H6, H9, H11, H12}.

Tableau 1-2 Etape d'exécution de l'algorithme BNL

Etape	Entré	Fenêtre	File temp	Skyline
1	H_1	$H_1(1)$	Ø	Ø
2	H_2	$H_1(1)$	Ø	Ø
3	H_3	$H_1(1), H_3(3)$	Ø	Ø
4	H_4	$H_1(1), H_3(3), H_4(4)$	Ø	Ø
5	H_5	$H_1(1), H_3(3), H_4(4)$	Ø	Ø
6	H_6	$H_1(1), H_4(4), H_6(6)$	Ø	Ø
7	H_7	$H_1(1), H_4(4), H_6(6)$	$H_7(7)$	Ø
8	H_8	$H_1(1), H_4(4), H_6(6)$	$H_7(7), H_8(8)$	Ø
9	H_9	$H_1(1), H_6(6), H_9(9)$	$H_7(7), H_8(8)$	Ø
10	H_{10}	$H_1(1), H_6(6), H_9(9)$	$H_7(7), H_8(8)$	Ø
11	H_{11}	$H_1(1), H_6(6), H_9(9)$	$H_7(7), H_8(8), H_{11}(11)$	Ø
12	H_{12}	$H_1(1), H_6(6), H_9(9)$	$H_7(7), H_8(8), H_{11}(11), H_{12}(12)$	Ø
13	H_{13}	$H_1(1), H_6(6), H_9(9)$	$H_7(7), H_8(8), H_{11}(11), H_{12}(12)$	Ø
Chargement des points de la file temporaire				
14	H_7	$H_9(9)$	$H_8(8), H_{11}(11), H_{12}(12)$	$H_1(1), H_6(6)$
15	H_8	$H_9(9)$	$H_{11}(11), H_{12}(12)$	$H_1(1), H_6(6)$
16	H_{11}	$H_9(9), H_{11}(11)$	$H_{12}(12)$	$H_1(1), H_6(6)$
17	H_{12}	$H_9(9), H_{11}(11), H_{12}(12)$	Ø	$H_1(1), H_6(6)$
18		Ø	Ø	$H_1(1), H_6(6), H_9(9), H_{11}(11), H_{12}(12)$

Afin d'améliorer le temps d'exécution et réduire le nombre de comparaisons, deux variantes à cet algorithme ont été proposées [BKS 01] :

1- *Considérer la fenêtre comme une liste à organisation automatique :*

Dans l'algorithme de base de BNL, la comparaison des n-uplets prend beaucoup de temps. Pour accélérer ces comparaisons, [BKS 01] ont proposé d'organiser la fenêtre comme une liste qui s'organise automatiquement. Quand un n-uplet **t** de la fenêtre domine un autre n-uplet, alors **t** est déplacé au début de la liste (fenêtre), le prochain n-uplet de l'entrée sera dans un premier temps comparé à **t**.

Cette variante est efficace s'il y a des n-uplets qui dominent plusieurs autres n-uplets ou s'il y a plusieurs n-uplets *neutres* (qui font partie de skyline et ne dominent pas d'autres n-uplets).

2- Remplacement des n-uplets dans la fenêtre:

En tant qu'autre variante, [BKS 01] ont essayé de mettre l'ensemble de n-uplets les plus dominants dans la fenêtre. Ceci est réalisé par le remplacement des n-uplets de la fenêtre. Dans [BKS 01], la politique de remplacement utilisée est basée sur la surface couverte par un n-uplet, afin de décider quels tuples devraient être maintenus dans la fenêtre.

Par exemple supposons qu'on a une fenêtre contenant les deux hôtels suivant:

(H1, 50$, 1000 m) et (H2, 59$, 900m)

Supposons aussi que le prochain hôtel considéré est (H3, 60$, 100 m).

H3 est incomparable avec les deux hôtels H1 et H2. Dans cet exemple, il est clair qu'une fenêtre contenant H1 et H3 est susceptible d'éliminer plus d'hôtels que la fenêtre de H1 et de H2. H3 est seulement légèrement cher mais plus près de la plage que H2. En d'autres termes, H3 est mieux en terme de *prix*distance* (surface couverte par H3) que H2. Donc H3 devrait remplacer H2 dans la fenêtre et H2 devrait être écrit à la file temporaire.

L'algorithme BNL a de bonnes performances pour des ensembles de données ayant une petite dimensionnalité, des distributions de données corrélées ou indépendantes. Le meilleur cas pour BNL est évidemment quand tous les points de skyline (points candidats) entrent dans la fenêtre, et l'algorithme se termine en 1 ou 2 itérations et sa complexité est de l'ordre $O(n)$; n étant le nombre de tuples (n-uplets) dans l'entrée. Son problème principal est la dépendance de la mémoire centrale, une petite mémoire (petite fenêtre) peut mener à de nombreuses itérations et sa complexité devient $O(n^2)$. Un autre inconvénient est son

insuffisance de progressivité (voir tableau 1-6). L'algorithme doit lire l'ensemble entier des données avant qu'il ne renvoie le premier point de skyline.

1.4.3.3 Algorithme à base d'index

Soit un ensemble de points de données **D** de d dimensions, l'algorithme d'index organise (partitionne) **D** en **d** listes. (Un point p est assigné au i^{ime} liste si et seulement si la valeur de p dans la i^{ime} dimension est la petite valeur dans toutes les autres dimensions de p) c.à.d. ($\forall 1 <= i, j <= d; p_i < p_j$ et $i \neq j$). Les points d'une même liste et qui ont même rang sont maintenus dans un lot (**Tableau 1-4** contient les lots de l'exemple des hôtels).

Tableau 1-3 Normalisation des valeurs de l'exemple des hôtels

Hôtel	Prix	Distance	Rang d'hôtel	
			Pour le Prix	Pour la Distance
H1	90	100	9	1
H2	100	200	10	2
H3	80	400	8	4
H4	70	650	7	7
H5	100	900	10	9
H6	80	200	8	2
H7	60	500	6	5
H8	30	400	4	4
H9	20	300	3	3
H10	33	800	5	8
H 11	15	900	2	9
H12	10	1000	1	10
H13	20	600	3	6

On note par :

- **Minval :** la valeur minimale du point p sur toutes les dimensions.
- **Maxval :** la valeur maximale des points de skyline courant pour toutes les dimensions.

L'algorithme examine chaque liste et commence par le lot qui a le minval minimum. Si le *minval* >= *maxval* alors il y a des points dans le skyline courant qui dominent les points de ce lot, et tous les lots suivants dans les différentes listes sont ignorés. Autrement le traitement d'un lot a deux étapes. Premièrement, un skyline local est calculé. Ensuite chaque point de ce skyline est comparé aux points de skyline courant (global) s'il est dominé par un ou

plusieurs points de skyline global, il est alors éliminé. Sinon il est inséré dans le skyline global comme un nouveau point et la valeur de maxval est mise à jour.

Avant que l'algorithme ne soit applicable et si les valeurs des dimensions n'ont pas le même domaine alors une normalisation de ses valeurs est nécessaire. Cette normalisation est faite en substituant les valeurs réelles de chaque dimension par leurs rangs (pour notre exemple voir Tableau 1-3).

On a deux listes pour notre exemple favori, l'hôtel H1 (4,1) est mis dans la deuxième liste tant que H12(1,10) est mis dans la liste1. Tous les lots des deux dimensions sont montrés dans le Tableau 1-4.

Tableau 1-4 Exemple des listes pour l'algorithme à base d'index

Liste1		**Liste2**	
Minval	**lot**	**Minval**	**lot**
1	H12	1	H1
2	H11	2	{H2, H6}
3	{H9, H13}	4	{H3, H8}
5	H10	5	H7
7	H4	9	H5

L'algorithme commence par les lots (hôtels) qui ont minval=1 dans les deux listes, donc H12, H1 sont ajoutés au skyline comme ils ne sont pas dominés par d'autres hôtels dans le skyline (initialement le skyline est vide) et la valeur maxval est mise à jour (maxval=10), de même H11 est inséré dans le skyline. Ensuite l'algorithme pointe vers la liste2 et traite le lot {H2, H6}, le skyline local de ce lot est H6, puis H6 est comparé au points S=(H1, H11, H12), il n'est pas dominé par ces points et il est inséré dans S. Similairement pour le lot {H9, H13}, H9 est inséré dans S, le lot suivant est {H3, H8} mais son minval=4 est supérieure au maxval=3 donc tous les lots suivants sont ignorés et l'algorithme termine avec un résultat de skyline S={H1, H6, H9, H11, H12}.

Cet algorithme peut rapidement renvoyer le skyline, il permet de retourner les points qui sont meilleurs dans une dimension en premier. C'est son avantage principal dans les applications qui nécessitent le calcul de skyline complet, mais ceci n'est pas utile dans des scénarios en ligne (l'utilisateur est en interaction avec l'application) ou le calcul rapide d'une

grande image sur l'ensemble de données est nécessaire. L'algorithme ne peut être appliqué directement pour des données ayant différents domaines de valeurs d'attributs, les listes (ainsi les structures d'index correspondantes) obtenues pour *d* dimensions ne peuvent pas être utilisées pour rechercher le skyline pour n'importe quel sous ensemble de dimensions, parce que la liste à laquelle un élément appartient change selon le sous ensemble de dimensions choisies. En général pour supporter des requêtes sur des dimensions arbitraires, un nombre exponentiel de listes doit être crée.

1.4.3.4 Algorithme Branch and Bound (BBS)

Comme l'algorithme du plus proche voisin [KRR 02], l'algorithme de skyline *Branch and Bound* BBS [PTF 03, PTF 05] est basé sur la recherche du plus proche voisin et de la notion de fonction de distance.

On va présenter cet algorithme en l'appliquant à notre exemple des hôtels (Tableau 1-5). Comme fonction de distance en utilise la fonction f= prix + distance. Un R-arbre est utilisé comme une structure d'indexation, les distances sont calculées quand on visite les nœuds de l'arbre.

Tableau 1-5 Fonction de distance pour les hôtels

Hôtels	Prix	Distance	Fonction de distance
H1	90	100	190
H2	100	200	300
H3	80	400	480
H4	70	650	720
H5	100	900	1000
H6	80	200	280
H7	60	500	560
H8	30	400	430
H9	20	300	320
H10	33	800	833
H11	15	900	915
H12	10	1000	1010
H13	20	600	620

L'ensemble des hôtels est organisé dans un arbre R (Figure 1-3) de capacité de trois points par nœud. Les hôtels de H1 à H13, les nœuds feuilles et intermédiaires sont montrés dans un plan à deux dimensions (Figure 1-2). La distance *mindist* d'un point est égale à la somme de ses coordonnés (prix +distance) et la distance d'une entrée intermédiaire (nœud non feuille) égale au *mindist* de son point faisant le coin bas-gauche.

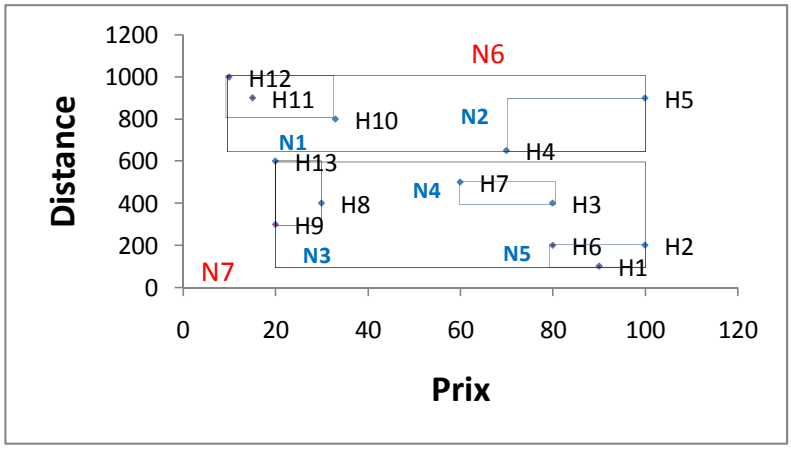


Figure 1-2 Exemple des nœuds d'un R arbre sur deux dimensions.

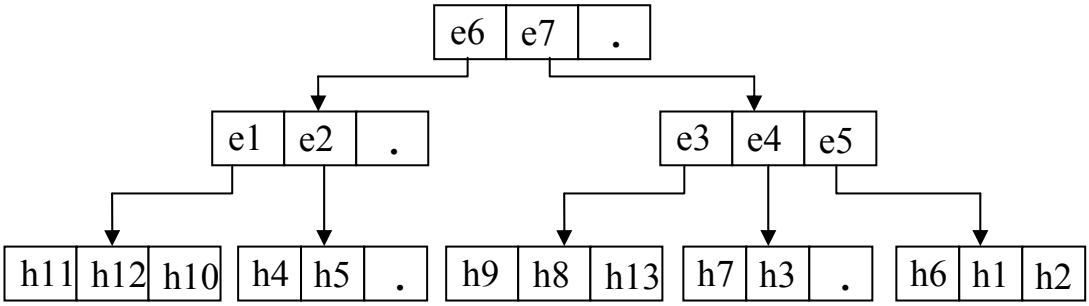


Figure 1-3 Exemple d'un R arbre.

Initialement, tous les fils du nœud racine r de l'arbre sont insérés dans un tas ordonné selon leurs distances *mindist.* Dans chaque itération, l'entré e qui a le *mindist* minimum est supprimée du tas et examinée contre les points de skyline courant. Si e est dominé par un ou plusieurs points de skyline, e est éliminé. Autrement Si e est un nœud de l'arbre, il est dépensé en insérant tous ses fils dans le tas s'ils ne sont pas dominés par des points de skyline courant. Si e est un point de données, il est inséré dans la liste des points de skyline comme un nouveau point, et l'algorithme termine lorsque le tas est vide. Pour notre exemple l'algorithme commence par le nœud racine r de l'arbre et insère toutes les entrées (e6, e7), ensuite e7 qui a le *mindist* minimum est augmentée à e3, e4 et e5. Ces trois entrées sont

également insérées dans le tas selon leur distance. Encore l'entrée avec la plus basse distance est augmentée (e5) et ses points (h1, h2, h6) sont insérés dans le tas, et l'algorithme trouve les points de skyline h1, h6. Maintenant l'algorithme continue avec l'entrée suivante et la procédure d'augmentation continue jusqu'à ce que le tas soit vide comme il est montré dans la Figure 1-4

Action	Contenu du tas	Liste de skyline S
Racine	(e7, 120) (e6, 660)	∅
e7	(e5, 180)(e3, 320)(e4, 460)(e6, 660)	∅
e5	(**h1, 190**)(**h6, 280**)[(h2, 300)] (e3, 320)(e4, 460)(e6, 660)	H1, H6
e3	(**h9, 320**)[(h8, 430)](e4, 460)(h13, 620)(e6, 660)	H1, H6, H9
e6	(e2, 720)(e1, 810)	H1, H6, H9
e2	[(h4, 720)](e1, 810)(h5, 1000)	H1, H6, H9
e1	[(h10, 833)]**(h11, 915)**[(h5, 1000)]**(h12, 1010)**	H1, H6, H9, H11, H12
	∅	H1, H6, H9, H11, H12

Figure 1-4 Exemple d'exécution de l'algorithme BBS

BBS est une adaptation (extension) de l'algorithme du plus proche voisin [KRR 02], sa force particulière se situe dans la vitesse globale de calcul de skyline. Les points de skyline sont renvoyés dans un ordre croissant de leur distance à l'origine, il est très optimal en terme d'entrée sortie (seulement les nœuds qui peuvent contenir des points de skyline sont visités et un nœud ne doit pas être visité plus d'une fois).

1.5 Discussion

Cette section présente une comparaison des algorithmes précédents. Pour cela, on va se baser sur les critères présentés dans [KRR 02] et [HAC 99] pour évaluer le comportement et

l'applicabilité des algorithmes de skyline en ligne. Un algorithme est dit en ligne (***Online Algorithm***) s'il satisfait les six propriétés suivantes :

1. *Progressivité* : Les premiers résultats devraient être retournés immédiatement et la taille de skyline augmente graduellement.

2. *Complétude des réponses* : L'algorithme devrait générer le skyline entier (complet) : Toutes les réponses qui satisfont la requête et qui ne sont dominées par aucune autre réponse.

3. *Stabilité des réponses* : L'algorithme devrait seulement renvoyer les points qui font partie de skyline. L'algorithme ne devrait pas retourner une réponse et après la remplacer par une autre réponse qui est meilleure que la première.

4. *Justesse* : L'algorithme ne devrait pas favoriser les points qui sont particulièrement bons dans une dimension. C'est une propriété importante pour garantir une grande image des réponses (les points retourner en premier sont bons dans toutes les dimensions).

5. *Incorporation des préférences* : L'utilisateur doit contrôler le processus de calcul, il est possible que l'utilisateur change ou ajoute des préférences pendant la phase d'exécution de l'algorithme.

6. *Universalité* : L'algorithme devrait être applicable à n'importe quelle distribution de données et pour n'importe quelle dimensionnalité.

Tableau 1-6 Comparaison entre les algorithmes BNL, Index et BBS

	BNL	Index	BBS
Progressivité	**NON**	**OUI**	**OUI**
Complétude des réponses	**OUI**	**OUI**	**OUI**
Stabilité des réponses	**NON**	**OUI**	**OUI**
Justesse	**OUI**	**NON**	**OUI**
Incorporation des préférences	**NON**	**NON**	**OUI**
Universalité	**OUI**	**NON**	**OUI**

Dans le tableau 1-6, on résume les propriétés de chaque algorithme. Un OUI indique la satisfaction d'une propriété par l'algorithme, alors qu'un NON indique la non satisfaction. Comme il est indiqué dans le tableau et selon les études réalisées dans [BKS 01, TEO 01, KRR 02, PTF 05], l'algorithme BBS est le meilleur dans tous les cas. Notons que BNL est utilisé dans diverses applications avec n'importe quelle dimensionnalité quelconque. Son problème majeur est la dépendance totale de la mémoire centrale. L'algorithme à base d'index est très rapide pour calculer les premiers points de skyline, le problème est qu'il retourne les réponses qui sont meilleures dans une dimension en premier, ceci n'est pas utile dans les applications interactives ou l'utilisateur nécessite une grande image (réponses qui sont meilleures dans toutes les dimensions) pour réagir en spécifiant d'autre préférences.

Conclusion

Dans ce chapitre, on a présenté une introduction sur les requêtes à préférences et les différentes approches pour exprimer ces requêtes. On s'est focalisé sur l'approche qualitative et plus précisément sur les requêtes dites Skyline du point de vue expression et implémentation. Ensuite, on a passé en revue trois principaux algorithmes pour le traitement des requêtes skyline. Le principe de fonctionnement de chaque algorithme est illustré au travers d'un exemple décrivant un ensemble d'hôtels avec leurs prix et distances de la plage. La requête considérée comme la recherche d'un hôtel le moins *cher* et *le plus prés* de la plage. Enfin, une comparaison est fournie selon les critères présentés dans [KRR 02].

Ainsi, chaque algorithme a ses avantages et ses inconvénients. Le choix d'un algorithme exige une analyse complète de l'application considérée en pesant le pour et le contre de chaque algorithme.

Les algorithmes présentés dans ce chapitre sont conçus pour être utilisés dans un contexte centralisé (une seule base de données centrale). Aucun de ces algorithmes ne peut être utilisé pour résoudre le problème de skyline dans un environnement distribué [HOS 05]. Le chapitre 3 traite ce problème et discute les solutions proposées. Mais, tout d'abord, introduisons une brève présentation sur les systèmes distribués en chapitre 2.

CHAPITRE 2

Environnement Distribué (ED)

Introduction

Depuis plusieurs années et avec l'évolution des technologies, les systèmes informatiques centralisés sont progressivement remplacés par des systèmes distribués. Ces systèmes sont constitués d'un ensemble d'entités autonomes de calcul (ordinateurs, périphériques, processeurs, processus, etc.) interconnectées entre eux via un réseau et qui peuvent communiquer.

Les systèmes distribués peuvent être construits selon deux modèles : le modèle client/serveur plat ou hiérarchique et le modèle pair à pair qui peut être pur ou hybride.

Dans ce chapitre, on va présenter quelques notions de deux principaux exemples de systèmes distribués : le Web qui se base sur le paradigme Client/serveur comme mode de fonctionnement et les systèmes pair à pair (P2P). Premièrement, le modèle client serveur est présenté, ensuite un rappel sur les systèmes P2P et ses différentes architectures est donné. Enfin, on aborde le problème de routage dans les réseaux P2P et plus particulièrement la technique des index de routage (Rounting indexes).

2.1 Le WEB et le modèle Client/Serveur:

La technologie Web a été inventée au *CERN* (Centre Européen de Recherche Nucléaire) en 1989 par Tim Berners-Lee. L'objectif était la diffusion de l'information en physique nucléaire. L'idée essentielle était d'organiser les informations sous forme d'un hypertexte permettant les liens entre les documents.

Ainsi est née l'idée d'un vaste système d'information distribué à l'échelle mondiale sur lequel les utilisateurs peuvent surfer depuis leurs terminaux : la toile d'araignée mondiale des documents désignée par l'acronyme WWW ou le diminutif Web (World Wide Web).

2.1.1 Définition :

Le web est le service le plus connu, le plus récent et maintenant le plus utilisé de consultation d'hyper documents. C'est un ensemble infini de sites reliés entre eux, hébergés sur les réseaux d'Internet et accessible par l'utilisation d'un logiciel : le browser (ou navigateur) capable de lire le langage *HTML*. Les applications web fonctionnent selon un environnement Client/serveur, cela signifie que des machines clientes (des machines faisant partie du réseau) contactent un serveur, une machine généralement très puissante en terme d'entrée/sortie, qui leur fournit des services.

2.1.2 Modèle client/serveur

Bien que le terme client/serveur soit devenu le maître mot en informatique, il n'existe aucun consensus sur sa signification exacte. Voici donc une de ces définitions. Comme l'indique le terme, ce sont deux entités distinctes, fonctionnant entre elles sur un réseau pour accomplir une tâche.

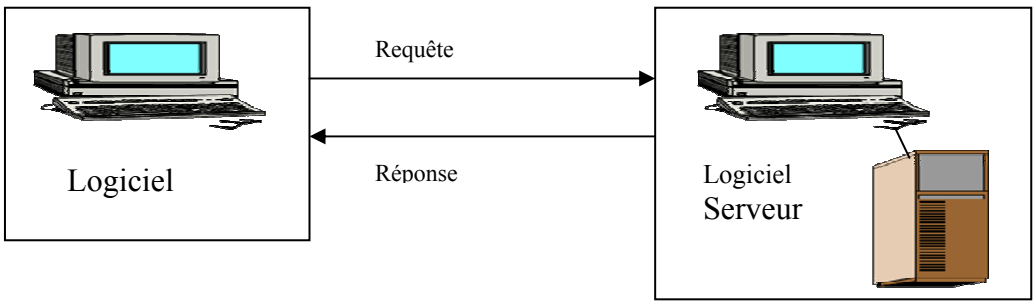


Figure 2-1 Architecture Client/serveur

Le client émet une requête vers le serveur grâce à son adresse et le port, qui désigne un service particulier du serveur. Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine cliente et de son port. Mais malheureusement, cette architecture présente de nombreux inconvénients :

- Le nombre de clients, le téléchargement et la demande de bande passante grandissants peuvent amener les serveurs à ne plus servir davantage de clients.
- Une caractéristique de cette architecture, est qu'elle nécessite très peu de puissance de calcul et de ressources du côté client, ce qui été justifié à l'ère où cette architecture a vu le jour, alors qu'à notre ère, la plupart des systèmes clients sont devenus très puissants.
- Le client dans une telle architecture agit de manière passive : il demande des services auprès des serveurs mais il est incapable d'offrir des services aux autres.

2.1.3 Avantages et inconvénients de l'architecture client/serveur.

Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux atouts sont :

- Ressources centralisées : étant donné que le serveur est au centre du réseau, il peut gérer des ressources communes à tous les utilisateurs, comme par exemple une base de données centralisée, afin d'éviter les problèmes de redondance et de contradiction.
- Une meilleure sécurité : car le nombre de points d'entrée permettant l'accès aux données est moins important.
- Une administration au niveau serveur : les clients ayant peu d'importance dans ce modèle, ils ont moins besoin d'être administrés.
- Un réseau évolutif : grâce à cette architecture il est possible de supprimer ou rajouter des clients sans perturber le fonctionnement du réseau et sans modification majeure.

L'architecture client/serveur a tout de même quelques lacunes parmi lesquelles :

- Un coût élevé dû à la technicité du serveur.
- Un maillon faible : le serveur est le seul maillon faible du réseau client/serveur, étant donné que tout le réseau est architecturé autour de lui ! Heureusement, le serveur a une grande tolérance aux pannes.

Imaginons le potentiel de stockage et de puissance de calcul que représentent les clients, libres ou emprisonnées dans leurs réseaux locaux : en considérant que seulement 10 millions

de machines sont connectées à Internet à n'importe quel moment et que chacune possède seulement 100 Mbit d'espace de stockage non utilisé, 1kbit/s de bande passante et 10% du pouvoir de calcul inexploité. A n'importe quel moment ces clients représentent donc 10^6 Go d'espace de stockage libre, 10 Gbit/s de bande passante disponible et 10^8 Mhz de «traitement» perdu.

L'architecture P2P (Peer to Peer) est la clé pour bénéficier de ce potentiel, en donnant aux clients la possibilité d'offrir des services aux autres clients. Contrairement à l'architecture client/serveur, le réseau P2P ne dépend pas d'un serveur central pour accéder à un service quelconque. Le réseau P2P évite l'organisation centralisée de l'architecture client/serveur. L'avantage principal des réseaux P2P est qu'ils distribuent les responsabilités de fournir des services sur l'ensemble des pairs du réseau.

On considère souvent que le P2P apporte une nouvelle philosophie de partage, un échange réciproque et fluide, sans passer par un serveur central.

2.2. Les systèmes Pair à Pair (P2P)

Le concept de peer-to-peer est aussi "vieux" qu'Internet lui-même, l'ancêtre d'Internet, plus connu sous le nom d'ARPAnet, a été conçu initialement comme un système P2P permettant le partage de ressources informatiques à travers les Etats-Unis. La plupart des sites ARPAnet étaient donc connectés non comme des clients-serveurs mais d'égal à égal. Un exemple permet de mieux comprendre la nature du P2P à cette époque est celui de FidoNet. Créé en 1984, celui-ci était un réseau décentralisé utilisé pour échanger des messages entre les utilisateurs à travers un "Bulletin Board System" (BBS). Il fonctionnait comme un réseau P2P où chaque BBS était traité comme un peer. Les services traditionnels tels que le FTP ou le Telnet dépendaient malgré tout d'un serveur central. Mais ce fut avec l'arrivée du World Wide Web et du protocole HTTP qu'Internet se tourna totalement vers un modèle client-serveur.

2.2.1 Définition du Peer To Peer

Le P2P est un modèle de communications dans lequel chaque partie possède les mêmes capacités. Il permet de symétriser la relation des machines qui interagissent : chaque pair peut jouer à la fois le rôle du client dans une transaction et celui du serveur dans une autre.

Plusieurs définitions ont été attribuées au concept P2P et voici une des plus significatives et complètes :

D'après [BRU 00] « Le P2P est une classe de systèmes **auto-organisants** qui permettent de réaliser une fonction globale de manière décentralisée en tirant parti du partage des ressources disponibles sur un **grand nombre d'extrémités de l'Internet».** On peut aussi définir un système P2P comme un système reparti utilisant l'architecture de réseaux P2P.

2.2.2 Caractéristique du modèle P2P

Les principales caractéristiques du modèle pair à pair sont :

- aucun pair n'a de vue globale du système;
- Chaque pair peut agir en tant que client et serveur;
- Aucune administration ou contrôle centralisé. Les pairs communiquent et s'organisent entre eux de manière autonome et coopérative ;
- La scalabilité qui veut dire qu'un véritable réseau P2P devrait être capable de garder les mêmes performances en terme de localisation de données, de routage de messages et d'insertion de nouveaux pairs quelque soit le nombre de pairs composant le réseau P2P ;
- Assurer de la disponibilité des ressources en absence des pairs qui les détiennent (chose qui est prise en charge par le serveur dans les réseaux client/serveur).

2.3 Classification des systèmes P2P

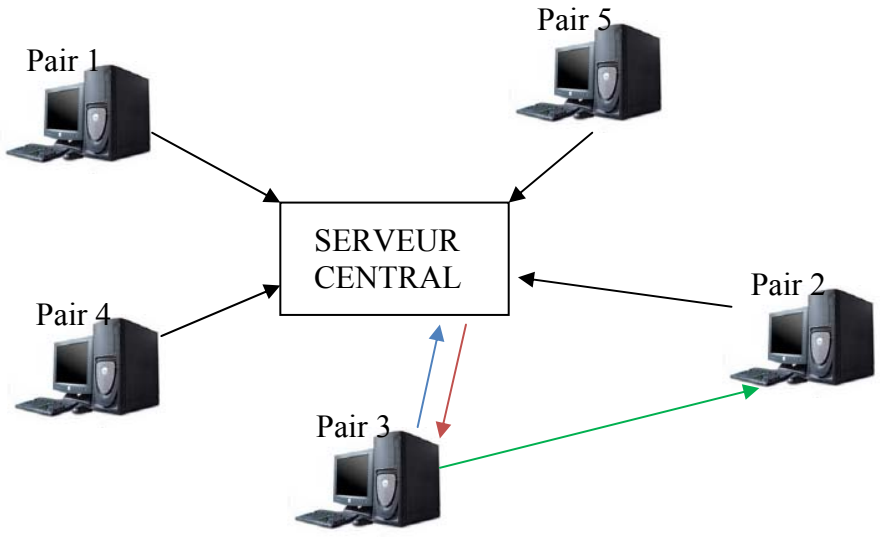
Avec la croissance des réseaux, le *peer-to-peer* suscite de plus en plus l'attention dans la recherche comme alternative aux réseaux client/serveur. Plusieurs études essayent de classifier les systèmes *peer-to-peer*. Trois grandes catégories peuvent être identifiées : centralisé, décentralisé, et hybride. La différence principale entre ces systèmes est le

mécanisme utilisé pour rechercher des ressources dans le réseau *peer-to-peer*. La catégorie décentralisée peut encore être divisée en décentralisé mais structuré, où le réseau possède une organisation automatique qui facilite la recherche du contenu ; et en décentralisé et non structuré où la recherche se fait de proche en proche à travers le réseau.

2.3.1 Architecture Centralisée

La caractéristique de cette architecture est qu'elle repose sur un serveur central auquel se connectent les utilisateurs. Ce serveur est chargé de les mettre en relation directe. L'intérêt de cette technique réside dans l'indexation centralisée de tous les répertoires et intitulés de fichiers partagés par les abonnés sur le réseau.

Lorsqu'un pair souhaite partager un fichier, il le déclare au serveur central. Celui-ci répertorie alors son adresse IP ainsi qu'un numéro de port donné par le client où il pourra être contacté pour un téléchargement.



- Le pair3 recherche un fichier en envoyant une requête au serveur central.
- Le serveur répond en envoyant la liste des pairs (pair 2) qui possèdent le fichier demandé.
- Le pair 3 télécharge le fichier directement à partir du pair 2.

Figure 2-2 Architecture centralisé

Ainsi le serveur dispose principalement de deux types d'informations : celles sur le fichier (nom, taille, ...), et celles sur l'utilisateur (nom utilisé, IP, nombre de fichiers, type de connexion, ...). Si un utilisateur désire un fichier, il interroge l'index central du serveur qui lui indique alors la liste des postes (adresses IP) sur lesquels il est susceptible de le trouver. Interroger l'index se fait de la même manière qu'une recherche avec un moteur de recherche classique. L'utilisateur choisit alors parmi les réponses celle qui lui convient le mieux (pertinence de la réponse, bande passante, taille du fichier) et contacte directement le ou les postes choisis. Le fichier en lui-même ne transite pas par le serveur central, qui fonctionne uniquement comme un annuaire.

Lorsque le nombre de participants devient trop grand ce modèle comporte quelques inconvénients dus à son infrastructure centralisée, notamment la surcharge de l'annuaire responsable d'accueillir les informations de tous les participants. Cette catégorie de réseaux *peer-to-peer* ne peut pas s'étendre à de très grands réseaux. De plus, l'existence d'un centre unique, même s'il est dupliqué, ne permet pas une bonne fiabilité du réseau. Celui-ci peut tomber en panne à cause d'un seul nœud. Le représentant le plus connu de ce modèle de P2P est **Napster.**

Exemple du réseau Napster [Nap]

Napster [Nap] est souvent considéré comme le premier réseau *peer-to-peer*. Son architecture est centralisée : les pairs du réseau annoncent les fichiers dont ils disposent au serveur central et contactent ce serveur central pour obtenir les coordonnées (adresse IP et numéro de port) d'un pair possédant les fichiers recherchés. Chaque utilisateur doit posséder le logiciel Napster sur son ordinateur afin de participer au partage des fichiers. Étant connecté à l'Internet, le client établit une connexion TCP avec le Serveur Central Napster et lui déclare les fichiers qu'il souhaite partager. Ce serveur détient donc un annuaire avec toutes les adresses IP des clients participants connectés à lui, ainsi qu'une liste des ressources partagées. Ainsi, tout client qui souhaite obtenir un fichier interroge le Serveur Central. Le Serveur Central lui communique les adresses IP de ceux qui possèdent le fichier désiré. Le logiciel Napster permet ainsi au client de se connecter directement sur les ordinateurs qui proposent le fichier, le fichier en lui-même ne transite pas par le serveur central, qui fonctionne simplement comme un moteur de recherche : une fois l'adresse IP trouvée, les

ordinateurs clients peuvent se connecter directement entre eux. Mais il ne s'agit pas d'un réseau totalement « pair à pair », dans la mesure où sans le serveur central, le réseau ne peut pas fonctionner.

2.3.2 Architecture Décentralisée non Structurée

Les systèmes décentralisés et non structurés sont ceux au sein desquels il n'existe ni annuaire centralisé, ni connaissance où se situent les noeuds du réseau (la topologie), ni adresse sur l'emplacement des fichiers. Le logiciel de P2P Gnutella [Gnu] en est un exemple. L'emplacement des fichiers n'est fondé sur aucune connaissance de la topologie. Pour trouver un fichier, un pair demande tout simplement à ses voisins qui eux-mêmes vont demander à leurs voisins s'ils n'ont pas le fichier considéré, et ainsi de suite. Ces architectures non structurées sont extrêmement résistantes aux noeuds entrant et sortant du système. Par contre, l'actuel mécanisme de recherche n'est pas adapté aux très grands réseaux (on dit que le réseau ne passe pas à l'échelle) et génère une charge importante sur les participants du réseau. Les exemples de cette catégorie sont nombreux comme FreeNet [FNet] ou Gnutella [Gnu].

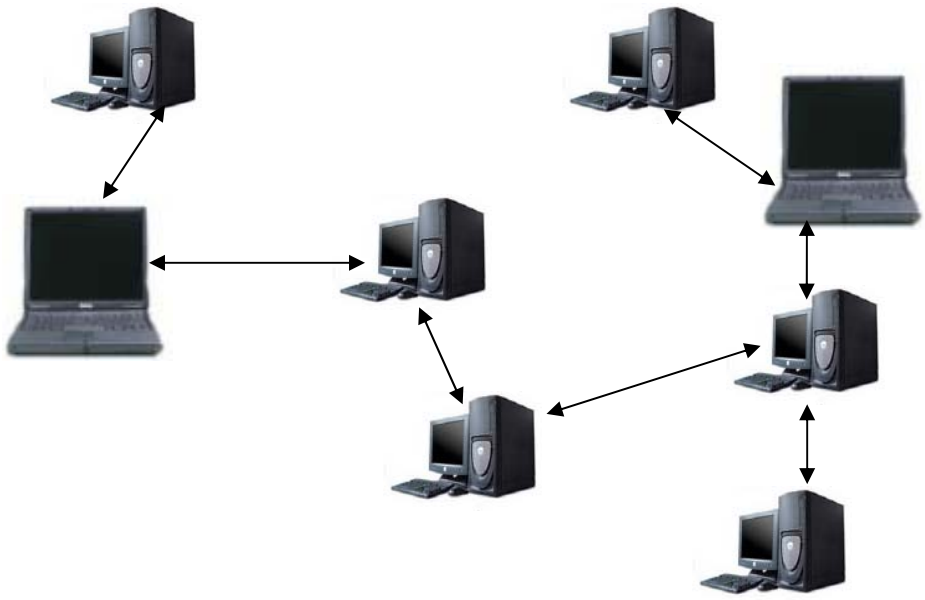


Figure 2-3 Architecture Décentralisé

Exemple du réseau Gnutella [Gnu] :

Gnutella a été le premier réseau *peer-to-peer* totalement décentralisé. Succédant à Napster, dont la centralisation présentait une faiblesse avérée, Gnutella a tiré profit de cette expérience. Gnutella propose un protocole ouvert, décentralisé pour des recherches distribuées sur un ensemble de pairs non hiérarchisés. Dans Gnutella, tous les pairs sont à la fois serveur et client. Ce protocole n'a pas de répertoire centralisé et n'a aucun contrôle sur la topologie ou l'emplacement des fichiers. Le réseau est formé avec des pairs qui intègrent le réseau en suivant quelques règles simples. L'emplacement des données n'est fondé sur aucune connaissance de la topologie. Pour localiser un objet, un client demande à ses voisins qui eux-mêmes demandent à leurs voisins. Ce genre de systèmes permet simplement l'entrée et la sortie des clients, mais le mécanisme utilisé passe mal à l'échelle et génère de fortes charges dans le réseau. La première version de Gnutella utilisait un mécanisme d'inondation pour la recherche, ce qui pose évidemment un problème dès que le réseau devient très grand. Les versions les plus récentes de Gnutella utilisent la définition de super-pairs ou ultra-pairs (pairs avec la meilleure bande passante) pour améliorer la performance du réseau. Cependant, cette solution reste encore limitée toujours à cause des mécanismes d'inondation même si l'inondation ne se fait que sur les ultra-pairs et non sur l'ensemble des pairs. Le protocole Gnutella définit comment les clients sont interconnectés dans le réseau. Il repose sur l'utilisation de descripteurs (Ping, Pong, Query, QueryHit et Push) pour la communication entre les clients et d'un ensemble de règles qui gèrent leurs échanges. Ainsi, lorsqu'un client cherche un objet, il lance une requête (Query) en spécifiant cet objet à tous ses voisins. Si l'un des voisins possède l'objet en question, il répond à la requête en renvoyant une réponse (QueryHit) au voisin qui lui a retransmis la requête, spécifiant où l'objet peut être téléchargé (son adresse IP et son port TCP). La réponse remonte ainsi de proche en proche jusqu'au client qui a initialisé la requête. Le client initiateur de la requête choisit ensuite les fichiers à télécharger en envoyant directement une requête de téléchargement au client qui possède le fichier. Si le noeud qui a reçu la requête ne peut la satisfaire, il propage la requête à tous ses voisins, sauf celui qui a initialisé le mouvement. En d'autres termes, Gnutella inonde le réseau pour trouver l'objet désiré. Néanmoins, pour ne pas inonder le réseau durant un temps trop long, Gnutella utilise un temps maximum (ou temporisateur) pendant lequel l'inondation se propage. Lorsqu'un client souhaite rejoindre le réseau, il doit connaître au moins un autre client déjà connecté, et c'est à ce dernier que le nouveau client est rattaché. Étant connecté,

un client envoie périodiquement des requêtes (Ping) à ses voisins. Cela permet de sonder le réseau à la recherche d'autres clients. Tous les clients qui reçoivent un Ping répondent avec un Pong qui contient l'adresse et la quantité de données partagée. En effet, des Pings fréquents permettent d'assurer les liens entre un client et le reste du réseau, puisque dans l'environnement dynamique que propose Gnutella (les clients rejoignent et quittent le réseau sans arrêt) la connexion n'est pas fiable.

2.3.3 Architecture Décentralisée Structurée

Les systèmes décentralisés mais structurés n'ont toujours pas de serveur central mais ils sont « structurés » dans le sens où leur topologie est strictement contrôlée et les fichiers sont stockés dans des endroits spécifiques pour faciliter les recherches. Ils sont dits structurés car les nœuds participant à l'application P2P sont reliés entre eux selon une structure particulière comme un anneau. Ce type de réseau a été défini pour rendre des services de routage et de localisation fondés sur le *peer-to-peer*.

Les systèmes *peer-to-peer* décentralisés mais structurés possèdent un algorithme de recherche pour lequel une ressource donnée se trouve en un endroit parfaitement déterminé à l'avance. Cet endroit a été calculé pour qu'il soit le plus simple d'accès aux autres *peers* du réseau. L'algorithme de recherche est donc complètement déterministe, et les liens entre les pairs sont faits suivant des règles bien définies. Cette structure permet la découverte efficace des fichiers partagés. De plus, elle est particulièrement appropriée au développement de réseaux à grande échelle. Dans cette catégorie, on peut placer Chord [SMN 01], CAN [RFH 01], Tapestry [ZKJ 00], Pastry [RPD 01], Kademlia [MDM 02] et Viceroy [MNR 02]. Pour trouver un fichier dans un réseau *peer-to-peer* classique, on fait de l'inondation de requêtes sur le réseau, en interrogeant tous les ordinateurs connectés jusqu'à trouver un ordinateur qui le détient. Pour simplifier cette recherche, les concepteurs des réseaux *peer-to-peer* se sont tournés vers des structures de données connues. En effet, améliorer les algorithmes de recherche nécessite d'organiser le réseau, donc de le structurer, ce qui est bien l'objectif de ces réseaux *peer to peer* décentralisés mais structurés.

2.3.4 Architecture Hybride

Les réseaux hybrides permettent de résoudre des problèmes de l'approche purement distribuée tout en gardant l'efficacité de la solution centralisée. Les réseaux fondés sur ces mécanismes de localisation et de routage de données peuvent supporter un nombre de pairs de l'ordre du million. Cette solution combine les caractéristiques des modèles centralisés et décentralisés. La décentralisation signifie, entre autres, l'extensibilité, la tolérance aux fautes et le passage à l'échelle. La centralisation partielle implique quelques autres serveurs qui contiennent des données importantes pour le système. Chaque utilisateur élit son Super-pair, qui est « le serveur central » pour « des noeuds locaux » et peut communiquer avec d'autres Super-pairs. KaZaA [Ka], FastTrack [Fas], eMule [Emule] sont des exemples de cette catégorie.

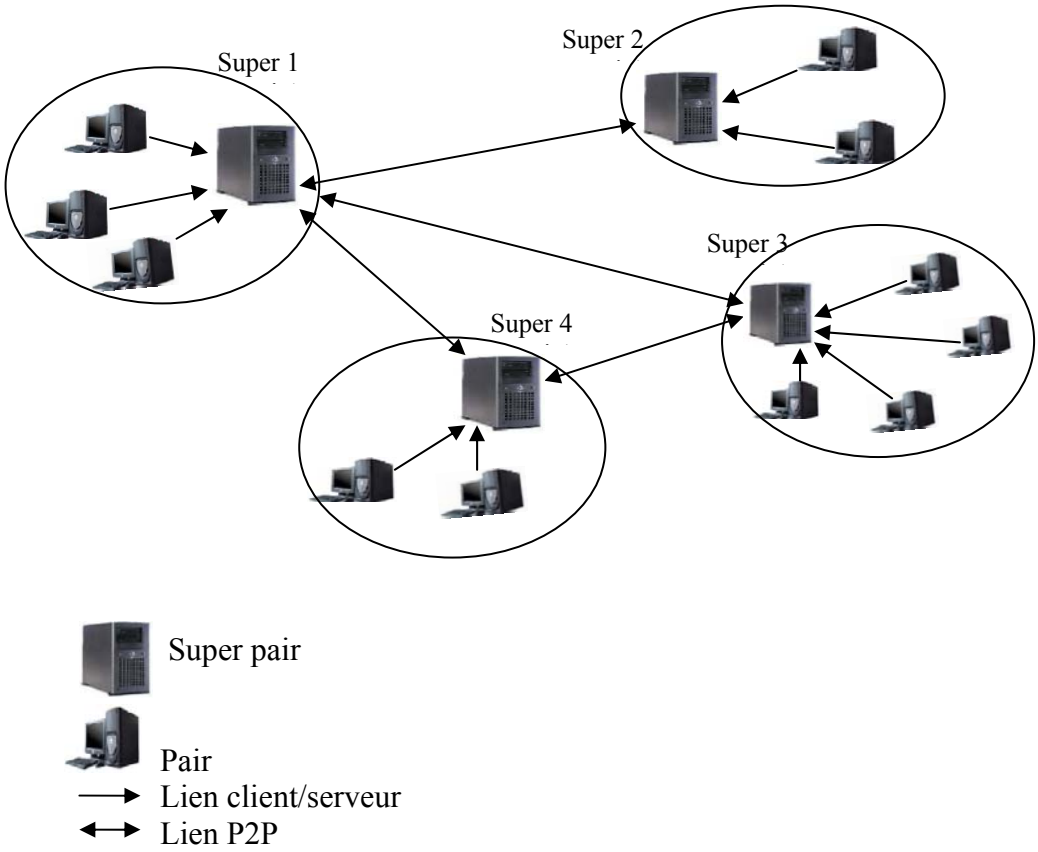


Figure 2-4 Architecture Hybride

2.4 Le routage dans les réseaux P2P

Il y a plusieurs mécanismes de recherche dans les systèmes pair à pair, chacun avec ses avantages et ses inconvénients. Ces techniques peuvent être classées en trois grandes catégories: (i) mécanismes sans index (pas d'index global sur l'ensemble de données du réseau P2P); (ii) mécanismes avec des nœuds spécialisés (un index global centralisé); (iii) le dernier mécanisme avec un index global distribué sur chaque pair. Par exemple Gnutella un réseau totalement décentralisé utilise une technique dans laquelle les nœuds n'ont pas un index global et les requêtes sont routées d'un nœud à un autre jusqu'à ce que les ressources (documents, fichiers,…) recherchées soient trouvés. Ce mécanisme de recherche se fait de la façon d'inondation. Lorsqu'un utilisateur sur un nœud donné cherche un objet, Gnutella envoie un message Requête en spécifiant par exemple le nom du fichier recherché à tous ses voisins dans le graphe. Si un des voisin possède l'objet en question, il répond au nœud en lui envoyant un message réponse spécifiant où l'objet peut être téléchargé (adresse IP ou numéro de port TCP par exemple). Si le nœud qui a reçu la requête ne peut pas la satisfaire, il propagera la requête à tous ses voisins, sauf l'initiateur du mouvement. En d'autres termes, Gnutella inonde le réseau de recouvrement pour trouver l'objet désiré. Afin de ne pas noyer le réseau définitivement, Gnutella utilise un Time To Live (TTL) de la même manière que IP.

La technique de la recherche centralisée (index global centralisé) utilise un nœud spécial (serveur) qui maintient un annuaire (index) sur toutes les données du réseau P2P, pour trouver un document, l'index global est parcouru pour identifier les nœuds pertinents pour la requête. Parmi les avantages de ce mécanisme, on note l'efficacité de recherche par l'indexation centralisée (juste un seule message est nécessaire pour répondre à une requête). Cependant cette technique est vulnérable aux attaques (réseau complètement dépendant du serveur).

La dernière technique maintient un index sur chaque pair (index distribué), *Crespo et ses collègues* [CGM 02] ont introduit la technique des index de routage (Rounting Indices), ces index fournissent l'information sur l'emplacement des données dans le réseau, trois index de routages ont été présentés et évalués dans [CGM 02]: Compound Rounting Index (CRI), Hop-Count Rounting Index (HCRI) et Exponential Rounting Index (ERI). Dans cette partie

on va se baser sur CRI, par la suite on présente le principe d'un CRI, les deux autres alternatives (HRCI et ERI) sont présentées en détail dans [CGM 02].

2.4.1 Index de routage: CRI (Compound rounting index)

Les index CRI sont similaires aux tables de routage des paquets de données dans les réseaux, chaque pair P résume pour chacun de ses voisins N € $\{N_1,\ldots,N_M\}$ les données qui sont disponibles par l'intermédiaire de N. Ceci inclut les données locales de N et toutes les données qui sont disponibles sur les voisins de N (sauf les données de P).

Path	# docs	Documents with topics: DB	N	T	L
B	100	20	0	10	30
C	1000	0	300	0	50
D	200	100	0	100	150

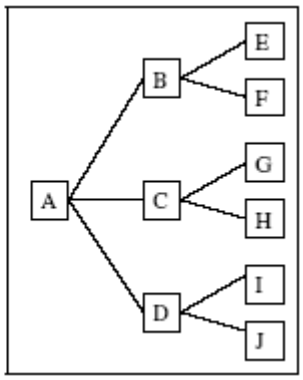


Figure 2-5 Réseau P2P sans cycles

On prend un exemple d'un index CRI comme technique de recherche dans un système P2P, et pour des raisons de simplicité, on suppose qu'il n y a pas de cycles dans le réseau (Figure 2-5). Dans ce système il existe des documents de plusieurs matières, et les requêtes sont posées sur des matières particulières; on suppose qu'on a seulement quatre (4) matières d'intérêt: Base de Données (DB); Réseaux (N : network); Théorie (T) et langages (L).

Chaque nœud a son propre index local pour trouver rapidement les documents locaux quand une requête est reçue. Tous les nœuds possèdent aussi un CRI (Compound Rounting Index) contenant (i) le nombre de documents pour chaque chemin (ii) le nombre de document sur chaque matière. Dans la Figure 2-5, on peut parcourir 1000 documents à travers C (c.-à-d. il y a 1000 documents dans C, G et H), parmi ces documents il y a 300 sur les réseaux et 50 sur les langages. Pour un index CRI, le nombre de documents qui peuvent être trouvés à travers un nœud (dans un chemin) est utilisé comme mesure de pertinence par rapport à la requête (le chemin qui a plus de documents cherchés est le plus pertinent).

Prenant le système P2P décrit sur la Figure 2-5, la Figure 2-6 présente une partie du réseau P2P avec un index de routage attaché à chaque nœud, la première rangée de chaque index contient un résumé de l'index local. En particulier, le résumé d'index local du nœud A montre que le pair A à 300 documents: 30 sur les bases de données, 80 sur les réseaux, pas de document pour la théorie et 10 sur les langages. Le reste des lignes représente un index de routage (CRI). Dans l'exemple, le CRI du nœud A montre qu'il y a 100 documents sur les bases de données à travers D (60 dans D, 25 dans I et 15 dans J). Quand un utilisateur pose une requête (document de BD et L) sur le pair A, premièrement la BD local de A est utilisée pour répondre à la requête.

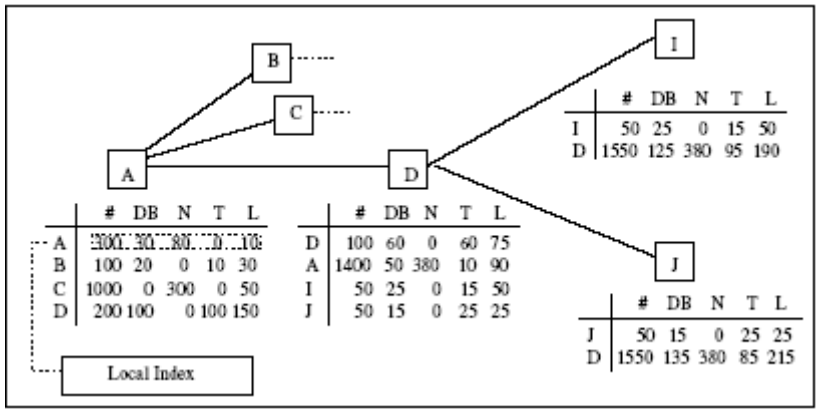


Figure 2-6 Index de routage

Si la condition de requête n'est pas satisfaite, A calcul la pertinence de chaque chemin (D à la meilleure pertinence parmi les voisins de A). Donc la requête est envoyée vers D et à son tour D retourne tous ses résultats locaux à l'utilisateur et si la requête n'est pas encore satisfaite, D calcul la pertinence de I et J et envoie la requête vers son voisins le plus pertinent, et ainsi de suite jusqu'à ce que la condition de la requête est satisfaite.

D'après les expériences de Crespo et ses collègues [CGM 02], un index de routage permet de minimiser le temps de réponse et réduire le nombre de messages par rapport à la solution sans index (ex: dans Gnutella, des milliers de messages crées pour chaque requête).
La procédure de création et de maintenance de l'index de routage, ainsi que deux autres alternatives d''index de routage HCRI et ERI sont décrites en détail dans [CGM 02].

CHAPITRE 3

Approches d'Evaluation des Requêtes Skyline dans un ED

Introduction

Tous les algorithmes présentés dans le premier chapitre se situent dans le cadre de calcul de skyline dans les bases de données centralisées. L'évaluation des requêtes skyline dans un environnement distribué (ED) représente un défi car, dans de nombreuses applications web, les attributs cibles sont disponibles sur différents sites qui sont accessibles seulement à l'aide d'interfaces externes très restreintes.

Les méthodes précédentes ne peuvent pas être appliquées dans un ED (elles sont conçues pour le cas centralisé) à cause de deux points :

- Il est supposé que les valeurs des attributs peuvent être retournées à un faible coût, qui n'est pas vrai pour le cas distribué.
- Les techniques centralisées se fondent sur des indices construits sur les attributs impliqués dans la requête. Cependant, ces structures de données centralisées ne sont pas disponibles dans un ED.

Dans ce chapitre, quelques algorithmes d'évaluation des requêtes skyline dans un ED sont présentés. Suivant l'ED utilisé, on distingue deux grandes classes d'algorithmes pour évaluer ces requêtes : la première classe se focalise sur les méthodes proposées pour évaluer les skylines dans le WEB, et la deuxième se base sur l'environnement P2P.

3.1 Traitement des requêtes Skyline dans le WEB

Dans cette section on va présenter le premier algorithme (BDS et IDS) qui permet d'évaluer les requêtes skyline dans un environnement distribué [BGZ 04] (Bases de données accessibles via le web), ensuite on présente un autre algorithme PDS proposé dans [LYL 05], il permet aussi l'évaluation des requêtes skyline sur des sources de données dans le web et de retourner les réponses progressivement.

3.1.1 Algorithme BDS (Basic distributed Skylining Algorithm)

L'algorithme distribué de base de skyline BDS [BGZ 04] comprend deux phases. (i) La première phase identifie un sous ensemble d'objets qui inclut tous les objets skyline probablement avec quelque autre objets non skyline, (ii) la deuxième filtre cet ensemble en éliminant tous les objets non skyline.

Tableau 3-1 Exemple d'hôtels avec 3 attributs sur 3 sites différents

Site 1		**Site 2**		**Site 3**	
Hôtel	Prix	Hôtel	Distance	Hôtel	Temps à l'aéroport
H12	1	H1	1	H6	1
H11	2	H6	2	H1	2
H9	3	H9	3	H2	3
H8	4	H3	4	H3	4
H10	5	H7	5	H9	5
H7	6	H13	6	H7	6
H4	7	H4	7	H4	7
H6	8	H10	8	H5	8
H1	9	H5	9	H8	9
H2	10	H11	10	H11	10
H5	11	H12	11	H12	11
H3	12	H2	12	H10	12
H13	13	H8	13	H13	13

Dans la première phase les données (objets) sont retournées par l'accès trié (*S: sorted access*) des différentes sources de donnée. Chaque source est invoquée dans un mode

ROUND ROBIN jusqu'à ce que tous les objets qui peuvent faire partie de skyline soient retournés. Au cours de cette phase et pour chaque attribut D_i un tableau K_i est créé pour sauvegarder les valeurs d'attributs des objets retournés par l'accès trié (Figure 3-1). La condition d'arrêt est atteinte lorsque un objet O est détecté (tous les valeurs d'attribut de O sont vues dans les différents tableaux K_i), l'objet qui a toutes ses valeurs d'attributs retournées via l'accès trié est appelé l'objet de terminaison.

Concernant l'objet de terminaison un important lemme est démontré dans [BGZ 04], quand un objet de terminaison T est détecté est aucun accès trié peut retourner une valeur égale à la valeur de T, tous les objets non retournés ne font pas partie de skyline.

La deuxième phase utilise des accès sélectifs ou aléatoires (*R: Rondom access*) focalisés pour trouver les autres valeurs d'attributs des objets retournés dans la première phase, et performe des comparaisons par pairs entre les objets des différents tableaux, les objets dominés sont éliminés avant de renvoyer le skyline à l'utilisateur.

Pour réduire le nombre de comparaisons ; il est prouvé dans [BGZ 04] qu' un objet O peut être dominé seulement par les objets du même tableau que O.

Pour mieux comprendre le fonctionnement de cet algorithme nous étendons notre exemple des hôtels en ajoutant le critère du temps à l'aéroport (Tableau 3-1) et donc on a trois sources de donnée (attributs) sur trois sites différents.

Puisque on a trois attributs, alors trois tableaux K_i sont créés :

K_1

Hôtel	P	D	T
H12	1		
H11	2		
H9	3	3	5
H8	4		
H10	5		
.	.	.	.

K_2

Hôtel	P	D	T
H1		1	2
H6		2	1
H9	3	3	5
H3		4	4
H7		5	
.	.	.	.

K_2

Hôtel	P	D	T
H6		2	1
H1		1	2
H2			3
H3		4	4
H9	3	3	5
.	.	.	.

Figure 3-1 Objets retournés dans la première phase

Pour cet exemple on suppose qu'il n y a pas des valeurs d'attributs dupliqués, H9 (tous ses valeurs d'attributs sont retournées) est choisi pour être un objet de terminaison ; donc on élimine tous les objets (ces hôtels sont dominés par H9) qui ne sont pas retournés par l'accès trié, les trois tableaux sont passés à la deuxième phase.

Premièrement pour le tableau K1 on fait un accès sélectif sur H12 en ce qui concerne les 2 autres attributs (distance et le temps à l'aéroport), on obtient H12 (1, 11,11) n'est pas dominé par H9 et de même H11 (2, 10,10). On élimine H8 (4,5,7) et H10 (5,8,12) qui sont dominés par H9, et de même pour le tableau K_2 (accès sélectif pour le prix et le temps), K_3 (accès sélectif pour le prix et la distance).

Wolf-Tilo Balke et ses collègues [BGZ 04] prouvent qu'un objet O peut seulement être dominé par d'autre objets qui existent dans tous les tableaux qui contiennent O. Le résultat final des 03 tableaux est retourné à l'utilisateur comme skyline S={H1, H6, H9, H11, H12} et l'algorithme se termine avec succès.

3.1.2 Algorithme IDS (Improved Distributed Skyline algorithm)

Dans l'algorithme précédent, il faut avoir au moins un objet (l'objet de terminaison) avec toutes ses valeurs d'attributs connues pour que l'algorithme se termine. Si cet objet est connu a priori, il est possible d'effectuer moins d'accès trié dans la première phase, et peu de comparaisons dans la deuxième phase. Pour cela les auteurs de [BGZ 04] ont proposé l'algorithme IDS basé sur une heuristique pour trouver l'objet de terminaison qui fait terminer l'algorithme plus rapidement.

Au début IDS réalise deux accès (triés et sélectifs) pour obtenir toutes les informations concernant les objets, et le nombre d'accès trié requis pour retourner toutes les valeurs d'attributs d'un objet O est obtenu on calculant un score s (s est la somme de différences entre les valeurs d'attributs de O et les derniers valeurs retourner par l'accès trié dans les différents tableaux).

IDS permet d'améliorer le comportement de l'exécution globale en réduisant le nombre d'accès de données, son succès dépend de l'exactitude de l'heuristique pour prévoir correctement l'objet de terminaison mais l'heuristique ne réussi pas dans toutes les situations en plus dans les deux algorithmes (BDS et IDS). Un objet est confirmé comme objet de skyline seulement après que la comparaison dans la deuxième phase est accomplie mais dans des applications en temps réel ceci peut être vu comme inconvénient à cause du temps de réponse important avant que le premier objet de skyline soit retourné et pour cela *Lo et ses collègues* [LYL 05] ont proposé leur algorithme PDS (Progressive Distributed Skyline

algorithm) qui permet de déterminer si un objet fait partie de skyline des qu'il est retourné la première fois et il le fait sortir immédiatement s'il n'est pas dominé par d'autres objets.

3.1.3 Algorithme PDS (Progressive Distributed Skyline algorithm)

PDS est basé sur le principe de progressivité et n'attend pas la fin de l'algorithme pour retourner le skyline, il retourne les objets skyline progressivement. Ainsi une méthode d'estimation de rang est nécessaire pour définir l'objet de terminaison.

Les deux principales idées de PDS (la progressivité et l'estimation de rang) sont discutées dans [LYL 05] :

La progressivité : pour permettre la progressivité il faut déterminer si un objet appartient au skyline dés que les valeurs de ses attributs seront retournées et pour cela les auteurs de [LYL 05] ont prouvé le lemme suivant :

Lemme : *si l'accès trié d'une source de donnée D renvoi les valeurs des attributs dans un ordre strictement croissant ; un objet O de D peut seulement être dominé par des objets qui sont retournés avant O.*

La vérification de dominance est faite en comparant un objet O avec tous les objets retournés avant lui dans la même source et pour accélérer le processus et réduire le nombre de comparaisons *Lo et ses collègues* ont choisi un arbre R^*(index multidimensionnel dans la mémoire centrale) et chaque attribut *i* impliqué dans la requête correspond à un arbre R_i contenant les objets skyline découvert basés sur l'accès trié à la source Di. Un objet O retrouvé de D_i est comparé à tous les objets dans R_i et s'il n'y a aucun qui domine O alors O est défini comme objet de skyline et il est inséré dans R_i.

Quand un nouveau objet O est retourné, la vérification de dominance est faite en exécutant une requête Q_0 qui a l'origine comme le coin inférieur gauche et O comme le coin supérieur droit. Si le résultat de Q_0 est vide alors O est un objet de skyline, autrement il existe au moins un objet O' qui est meilleur que O dans toutes les dimensions et par conséquent O est éliminé. Le cas général où il y a des valeurs dupliquées, l'accès trié ne peut renvoyer des valeurs dans un ordre strictement croissant, le lemme précédent ne peut être appliqué directement. Ce problème est résolu dans [LYL 05] en utilisant un buffer B_i pour chaque source de donné D_i, si O est défini comme un objet possible de skyline, il est inséré dans B_i au lieu de le rapporter comme objet de skyline immédiatement. Le buffer B_i est un autre arbre

R^* qui contient les objets possibles d'être dans le skyline avec la même valeur sur l'attribut *i*, par conséquent, pour chaque objet O inséré dans B_i. Une vérification est faite (si O est dominé ou non). Quand un objet avec une grande valeur d'attribut de la source D_i est inséré dans B_i. Alors les objets qui restent dans B_i sont retournés comme objets de skyline définitivement et le buffer est vidé.

Estimation de rang :

Lo et ses collègues supposent qu'un bon objet de terminaison est caractérisé par son rang dans les listes K_i (tableau ordonné de valeurs d'attribut). Pour un objet O, il est noté que :

- $Rank_i(O)$: le rang de l'objet O dans la source Di.
- $Sumrank_i(O)$: est la somme des rangs de O dans toutes les sources de donnée D_i.

Si T est un objet de terminaison alors le nombre minimum d'accès exigés pour le localiser par l'accès trié est $Sumrank_i(T)$. Un bon objet de terminaison est celui qui a le minimum *Sumrank* et par conséquent le nombre d'objets qui doivent être comparés augmente avec l'augmentation de $Sumrank_i(T)$,

L'approche PDS a les informations concernant les objets qui sont retournés par l'accès trié et le rang d'un objet O retourné par le j^{ime} accès trié dans la source D_i est j (le rang du premier objet retourné est 1) mais l'accès sélectif retourne l'attribut d'un objet sans aucune information sur son rang. Les auteurs de [LYL 05] ont proposé d'employer la régression linéaire comme méthode d'estimation de rang. Pour une source D_i, l'équation qui calcule le rang d'un objet O est donnée comme suit:

$$Rank_i(O) = \boldsymbol{a_i} \times value_i + \boldsymbol{b_i} \qquad (1)$$

$value_i$ est la valeur de l'objet O dans la source D_i, supposons qu'on a fait K accès trié à la source D_i, les coefficients a_i et b_i peuvent être obtenus par les formules suivantes:

$$a_i = \frac{K \sum_i Rank_i \; value_i - (\sum_i value_i)(\sum_i Rank_i)}{K \sum_i value_i^2 - (\sum value_i)^2} \qquad (2)$$

$$b_i = \frac{(\sum_i Rank_i)(\sum_i value_i^2) - (\sum_i value_i)(\sum_i Rank_i \; value_i)}{K \sum_i value_i^2 - (\sum value_i)^2} \qquad (3)$$

Exemple:

Reprenons notre exemple des hôtels du tableau 3-1. Premièrement l'algorithme réalise deux accès triés sur chaque source, les objets (hôtels) et les valeurs d'attributs retournées sont S(D1, H12, 1), S(D1, H11, 2) et S(D2, H1, 1), S(D2, H6, 2) et S(D3, H6, 1), S(D3, H1,2). Puisque les valeurs d'attributs sont retournées dans un ordre strictement croissant, les hôtels H12, H1 et H6 font partie de skyline, leurs valeurs inconnues sont retournées par l'accès sélectif. Ensuite l'algorithme vérifie la dominance des objets du deuxième rang (dans notre cas H11 parce que H1 et H6 sont déjà dans le skyline). Selon le lemme précédent H11 fait partie de skyline et les coefficients a_i et b_i peuvent être calculés à partir des informations disponibles. Par exemple, soit l'hôtel (H12, 1) avec le rang 1 et (H11, 2) avec le rang 2 dans la source D1, les valeurs de a_i et b_i sont:

$$a_1 = \frac{2(1.1 + 2.2) - (1 + 2)\ (1 + 2)}{2(1^2 \ | \ 2^2) \quad (1 \ | \ 2)^2} = 1$$

$$b_1 = \frac{(1 + 2)(1^2 + 2^2) - (1 + 2)\ (1.1 + 2.2)}{2(1^2 + 2^2) - \ (1 + 2)^2} = 0$$

Après le calcul des coefficients a_i et b_i, les rangs inconnus des objets dans les différentes sources sont estimés à l'aide de l'équation (1). Par exemple le rang de H12 pour l'attribut distance est : $Rang_{dist}(H12) = 1.11 + 0 = 11$.

L'hôtel H6 est choisi comme objet de terminaison par ce qu'il a le minimum *Sumrank*. Le Tableau 3-2 résume les rangs estimés des différents objets avec leurs *Sumrank*s où les rangs estimés sont soulignés.

Tableau 3-2 Les rangs estimés des hôtels dans l'exemple

Hôtel	rang	Somme des rangs
H12	(1, 11, 11)	23
H11	(2, 10, 10)	22
H1	(9, 1, 2)	12
H6	(8, 2, 1)	11

Ensuite, PDS sélectionne itérativement un attribut dont la valeur de l'objet de terminaison courant n'a pas été retournée par l'accès trié comme prochaine source pour effectuer l'accès

trié. Par exemple, on prend l'attribut prix de H6 et on effectue un accès trié sur la source D1 (prix), l'objet retourné est H9 S (D1, H9, 3) et les valeurs de ses autres attributs sont retournées par l'accès sélectif R (D2, H9, 3) et R (D3, H9, 5). L'élément H9 est inséré dans le skyline parce qu'il n'est pas dominé par aucun objet retourné avant lui.

La procédure est répétée jusqu'à il y a un objet dont toutes les valeurs d'attributs ont été retournées par l'accès trié (dans notre cas l'hôtel H9). A ce moment, l'algorithme se termine avec succès.

3.2 Traitement des requêtes Skylines dans les systèmes P2P

Les algorithmes existants de calcul de skyline dans un contexte centralisé ne peuvent être appliqués systématiquement dans un environnement P2P. Aucun de ces algorithmes ne fournit une solution efficace pour l'évaluation des requêtes skyline dans un système pair à pair (à cause de la décentralisation stricte et de la connaissance limitée sur le réseau). Récemment quelques solutions pour résoudre le problème de skyline dans un système P2P ont été proposées dans la littérature [WOT 05] [LTL 06] [HOS 05].

Dans cette partie on va présenter le travail de Hoste [HLS 06] et Zinn [ZIN 05] qui ont proposés des approches pour le traitement des requêtes skyline dans les systèmes P2P. Dans [HLS 06], ainsi que dans [ZIN 05], les auteurs utilisent des index de routage appelés DDS (Distributed Data Summaries). Ces index sont basés sur une structure d'arbre particulier, appelé QTree. Cette structure est une combinaison d'histogramme et d'une autre structure d'arbre (dit R-Tree). La section 3.2.1.1 présente la stratégie de construction d'un index de routage dans le réseau P2P, ensuite l'algorithme proposé dans [ZIN 05] est détaillé.

3.2.1 Algorithme de Zinn

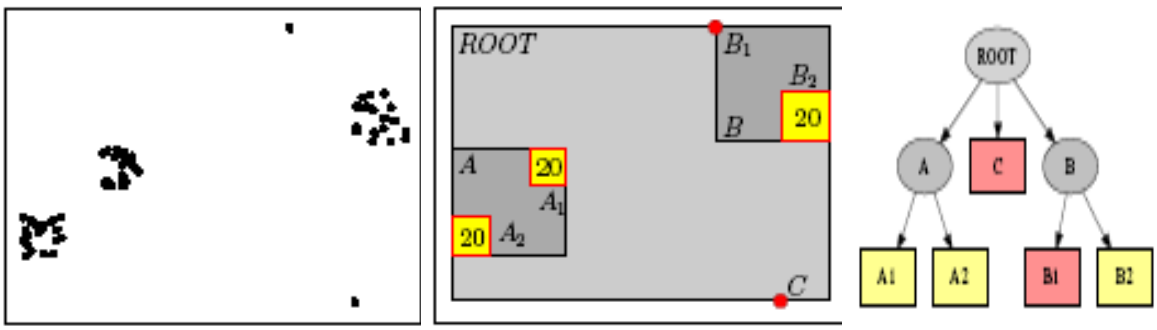
La solution proposée dans [ZIN 05] repose sur deux stratégies essentielles. La première concerne le routage dans le réseau P2P (comment construire un index de routage dans le réseau P2P). La deuxième présente l'algorithme proposé pour l'évaluation d'une requête skyline dans un système P2P.

3.2.1.1 Index de routage dans le réseau P2P

En général, l'index de routage présente des informations sur les données d'un pair (c'est un résumé de ses données). Dans [HLS 06], l'idée de construction de l'index de routage est basée sur une structure d'arbre particulier, appelée QTree. Dans ce qui suit, une description générale de Qtree [ZIN 05] est présentée, puis on présente la procédure de construction d'index dans un réseau P2P.

3.2.1.1.1 Description de Qtree

QTree est une structure d'arbre où chaque nœud correspond à une boîte multidimensionnelle rectangulaire (appelée MBB pour Minimum Bounding Box). Un MBB est le plus petit rectangle contenant de l'information sur les données se trouvant dans le sous-arbre (si le nœud n'est pas une feuille mais un nœud interne) ou dans le nœud (si le nœud est une feuille). Cette information est de nature statistique comme, par exemple, le nombre de données accessibles via le nœud. Chaque MBB d'un nœud fils est subsumé par celui de son parent. Les feuilles sont appelées des "paniers" (buckets). Plus la taille d'un panier est réduite, mieux on peut décider si ses données conviennent pour répondre à une requête posée. Chaque QTree est caractérisé par deux paramètres *fmax* (décrivant le nombre maximal de nœuds fils qu'un nœud interne peut avoir) et *bmax* (représente le nombre maximal des paniers à utiliser dans un QTree). Pour plus de détails voir [ZIN 05].



-a- -b- -c-

Figure 3-2 Exemple de Qtree

La figure 3-2 illustre un exemple de Qtree 2-dimentionnel avec les paramètres suivants : *fmax,=3* et *bmax =5*, la figure 3-2-a montre les données originales, la figure 3-2-b les boites

multidimensionnelle (MBB) de Qtree et la figure 3-2-c montre l'arbre Qtree avec ses buckets et noeuds internes.

Dans la Figure 3-2-b les buckets C et B1 (coloré en rouge) sont des points de données et chacune des buckets A1, A2 et B2 contient l'information qu'il y a 20 points de données dans le domaine couvert par leurs boites multidimensionnelles associées. La structure de données Qtree a été expérimentalement évaluée dans [ZIN 05].

3.2.1.1.2 Construction de l'index

Cette section présente la manière (la procédure) de construction d'un index dans un réseau P2P. Les auteurs de [HLS 06] et [ZIN 05] utilisent un système P2P non structuré DBMS (Data Base Management System). Le réseau est constitué d'un ensemble de pairs où chaque pair (nœud) est relié à un petit ensemble de voisins et peut envoyer des messages à ses voisins. Un réseau P2P sans cycles est supposé. L'idée principale de l'algorithme est présentée via un exemple simple.

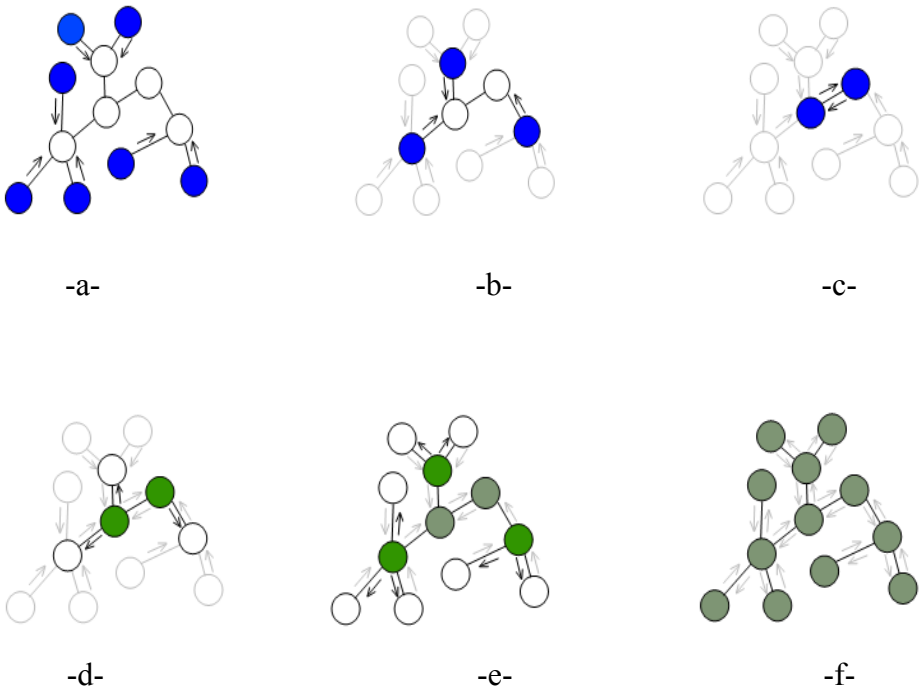


Figure 3-3 Etapes de construction de l'index de routage dans le réseau P2P

La Figure3-3 montre les étapes de construction de l'index dans un réseau P2P contenant 13 pairs. La Figure3-3-a montre la phase d'initialisation: tous les nœuds feuilles (colorés en bleu) envoient l'information sur leurs données locales (compressées à l'aide de la structure Qtree), puis les pairs qui ont reçu les messages de ses voisins sauf un (appelé N) fusionnent leur résumé de données (Qtree) avec toutes les informations reçues pour les envoyer ensuite à N (Figure3-3-b), ceci est fait itérativement jusqu'à ce qu' un nœuds central ait reçu l'information de tous leurs voisins (Figure 3-3-c). La dernière étape consiste à faire l'opération inverse (l'information globale est diffusée dans le même chemin mais de haut en bas), et l'algorithme se termine quand le dernier nœud feuille a reçu l'index correct (Figure 3-3-f).

3.2.1.2 Algorithme de Skyline

Cet algorithme est basé sur l'index de routage présenté dans la section précédente appelé DDS (Distributed Data Summaries). L'idée est de généraliser la relation de dominance $\mathbf{\Omega_S}$ pour qu'elle soit appliquée aux buckets. Par exemple, soient A et B deux buckets, et si

1. $a_{worst} \in A$: le point de données qui a le mauvais score tel que $\forall$ a' $\in$ A : a' $\mathbf{\Omega_S}$ a_{worst}.

2. $b_{best} \in B$: le point de données qui a le meilleur score tel que $\forall$ b' $\in$ B : b_{best} $\mathbf{\Omega_S}$ b'

$$\text{alors } A \ \mathbf{\Omega_S} \ B \Leftrightarrow a_{worst} \ \mathbf{\Omega_S} \ b_{best}.$$

Les points de données peuvent être interprétés comme des buckets sans extension. À l'aide de cette propriété chaque pair peut calculer son skyline de ses données locales, plus le skyline des buckets de son index (DDS).

Tous les buckets contenant des points de skyline font aussi partie du skyline sur les buckets parce que si une buckets B contient un point b qui est dans le skyline (réponse final) alors B ne peut être dominé par aucun autre bucket. L'algorithme de skyline exécuté sur chaque pair est défini comme suit :

- calculer le skyline de ses données locales et tous les buckets de son index de routage.
- Envoyer (router) la requête vers les pairs correspondant aux buckets du skyline
- Combiner les réponses des différents pairs dans une réponse finale.

Dans [HLS 06] une extension de cette algorithme est présentée, elle permet un traitement efficace avec une qualité inférieure de réponse (réponse approximative).

Conclusion

Dans ce chapitre, deux familles d'approches d'évaluation des requêtes skyline dans un contexte distribué ont été présentées. La première famille dédiée au web (modèle client/serveur) où les attributs cibles peuvent être distribués sur plusieurs sites web. Les algorithmes BDS, IDS et PDS font partie de cette catégorie. Il est montré dans [LYL 05] que PDS est le plus performant par apport aux BDS et IDS, il permet de retourner le skyline complet avant que BDS et IDS retourne ces premiers résultats. La rapidité de l'algorithme PDS se traduit par sa stratégie efficace pour l'estimation de rang et par conséquent l'identification de l'objet de terminaison et la structure d'arbre R^* utilisé pour accélérer la vérification de dominance des objets.

La deuxième approche concerne l'évaluation des requêtes skyline dans les systèmes pair à pair. Relativement, peu de solutions sont proposées dans la littérature [WOT 05, LTL 06, ZIN 05]. Dans cette partie, on a présenté la solution proposée par Zinn [ZIN 05]. Elle consiste à construire un index de routage qui permet d'envoyer la requête seulement aux pairs pertinents, ensuite à définir un algorithme pour le calcul de skyline dans ce système.

Le principe de construction de l'index de routage dans [ZIN 05] est utilisé dans l'étude présentée dans le chapitre suivant.

CHAPITRE 4

Vers une Evaluation des requêtes flexibles dans un ED

Introduction

Ce chapitre aborde le problème de l'évaluation des requêtes flexibles dans un contexte distribué, illustré par un système P2P. Récemment, quelques travaux ont été proposés pour tenter de résoudre ce problème, voir par exemple [BHJ 07] [HRV 08]. L'approche proposée comprend deux phases. La première phase permet de construire un index global de routage qui décrit les données des différentes sources, afin de trouver l'ensemble des pairs pertinents pour la requête posée. La seconde phase présente un algorithme de recherche des réponses satisfaisant le mieux la requête.

La procédure proposée pour la construction d'index combine l'algorithme proposée dans [HLS 06] (voir section 3.2.1 du chapitre 3) et le modèle de résumé SaintEtiQ [RNM 02]. La section suivante décrit un exemple d'une requête flexible posée dans un système pair à pair pour trouver les caméras qui sont "*pas trop chères*" et ayant une "*bonne*" qualité. Puis, on présente quelques notions générales sur l'interrogation flexible et le principe de fonctionnement du modèle SaintEtiQ. L'approche pour évaluer de requêtes flexibles dans un contexte distribué est détaillée en section 4.4.

4.1 Exemple de référence

Dans ce qui suit, l'exemple suivant est utilisé pour illustrer notre approche. Il s'agit de 03 magasins de vente de cameras repartis sur trois sites. Chaque site maintient sa propre base de données. Chaque tuple (camera) est décrit par deux attributs (prix, qualité). Dans le tableau 4.1 la colonne modèle est utilisée juste pour l'identification des tuples. La requête flexible Q considérée concerne la recherche des cameras qui sont "*pas trop cher*" et ayant une "*bonne*" qualité.

$$Q = Pas\_Trop\_cher \square Bonne\_qualité.$$

Où "*pas trop cher*" et "*bonne*" qualité correspond à des prédicats graduels modélisés par les ensembles flous donner en Figure 4.1.

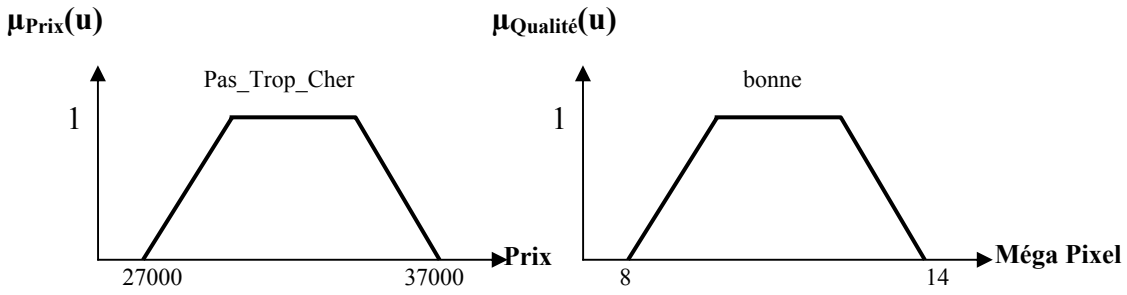


Figure 4-1 Ensembles flous pour les prédicats "pas_trop_chèr" et "bonne qualité"

Tableau 4-1 Exemple d'un ensemble de caméras réparties sur trois sites

Site 1			Site 2			Site 3		
Modèle	Prix	Qualité	Modèle	Prix	Qualité	Modèle	Prix	Qualité
S1	16500	7,2	C1	9000	5	X1	29500	9,3
S2	27500	8,1	C2	26000	6,5	X2	31000	12,8
S3	27000	8,2	C3	37500	7,9	X3	54000	14,1
S4	36900	13,6	C4	5500	4,6	X4	33000	8,1
S5	63500	10,1	C5	41000	14	X5	72000	16,1
S6	35000	10,1	C6	56000	15,3	X6	10000	7,9
S7	29000	9,5	C7	52100	15,1	X7	16000	7,1
S8	39500	8,3	C8	73000	16,5	X8	36000	10,3
S9	41200	14	C9	82000	17	X9	29000	8
S10	53700	14,7	C10	61000	16,5	X10	27000	7,8
S11	30000	12,3				X11	34000	8,9
S12	14900	5,5						

Dans le Tableau 4-1 le prix est exprimé en DA. La qualité concerne la résolution d'une caméra. Elle est mesurée en termes de nombre de pixels.

4.2 Interrogation flexible

Habituellement, lorsqu'un utilisateur sélectionne des données dans une base, il exprime une contrainte booléenne sur les données recherchées. Les systèmes d'interrogation classique distinguent alors deux catégories de données: celles qui satisfont la contrainte et celles qui ne la satisfont pas. Pour certaines applications, cette approche n'est pas appropriée, en effet des problèmes apparaissent lorsque l'utilisateur ne s'exprime pas directement en termes de conditions booléennes, mais au contraire en termes de conditions graduels, en raison de l'existence de préférences au niveau de ses critères de recherche [BLP 98]. Ces problèmes peuvent être résolus en utilisant une interrogation souple dite ***Flexible***. Le principe d'une interrogation flexible vise à étendre le comportement binaire en introduisant des « préférences » dans la requête, ainsi améliorer la capacité d'expression des langages de requêtes. Un élément retourné par une requête sera plus ou moins pertinent suivant les préférences qu'il aura satisfaites.

Définition

Une requête flexible est une requête qui comporte des descriptions imprécises et/ou des termes vagues qui peuvent comporter des préférences.

Définition

D'après Bosc, Motro et Pasi [BMP 01], un système d'interrogation est dit flexible s'il « permet d'exprimer des préférences de manière à ordonner les éléments plus ou moins acceptables qui auront été trouvés en fonction de leur adéquation à la requête.

Le paradigme de l'interrogation flexible décrit par Bosc et al. [BMP 01] assimile un système flexible à un expert humain servant d'intermédiaire entre l'utilisateur et le système d'information. Suivant cette hypothèse, l'expert serait capable d'analyser les demandes des utilisateurs, de déterminer leurs besoins en termes d'information, de localiser les sources d'information adéquates, d'en extraire les éléments de réponse plus ou moins pertinents et de les ordonner. Cependant, il est évident qu'il existe une certaine différence entre l'utilisateur et l'expert; ce dernier ne peut totalement cerner les besoins de l'utilisateur. L'expert ne peut faire qu'une estimation des éléments pertinents. La qualité du système d'interrogation sera

largement influencée par celle de la mise en correspondance entre, d'une part, l'expression de la requête et, d'autre part, les besoins à satisfaire, déduits de la requête, que le système adopte comme objectifs.

4.2.1 Approche basée sur les ensembles flous

Les systèmes d'interrogation flexible se distinguent par les techniques utilisées pour exprimer les préférences et les combinées et pour représenter les réponses. Plusieurs systèmes ont été proposés dans la littérature, il a été montré dans [BOP 92] qu'une approche basée sur la "théorie des ensembles flous" les unifie. Dans les modèles basés sur la théorie des ensembles flous, les requêtes définissent des prédicats flous qui sont plus ou moins satisfaits par les données. Ainsi le résultat d'une requête est un ensemble flou contenant des éléments plus ou moins satisfaisants. Par la suite, on présente et on donne plus de détails sur le langage SQLF basé sur la théorie des ensembles flous (qui permet d'introduire dans SQL des prédicats flous dans les critères de requêtes).

SQLF

Le langage d'interrogation SQLf, proposé par Bosc et Pivert [BOP 95] est une extension du langage SQL permettant à l'utilisateur de construire des requêtes graduelles sur des conditions atomiques définies par des ensembles flous. Chaque condition atomique associe un degré de satisfaction $\mu \in [0, 1]$ à une valeur d'attribut. L'extension, aussi bien sémantique que syntaxique, permet d'exprimer divers éléments d'une requête sous une forme floue. On trouve parmi ces éléments des opérateurs, des fonctions d'agrégation, des modificateurs linguistiques et des quantificateurs, de même que les variables linguistiques et autres prédicats graduels.

Le langage SQLF n'interroge que des bases usuelles contenant des relations strictes, les relations floues sont uniquement le résultat d'une requête floue. La forme générale d'une requête est :

SELECT [n| β | n, β] <liste-attributs>
FROM <relations>
WHERE <conditions-floues>;

Où n est un seuil quantitatif (les 'n' meilleures réponses) et β un seuil qualitatif (les données qui satisfont la requête avec un seuil supérieur à 'β').

SQLf détermine le degré d'appartenance d'un enregistrement à la réponse grâce à une algèbre relationnelle étendue aux constructions floues, généralisation de l'algèbre relationnelle. Cette algèbre étendue opère sur des relations graduelles et fournit des relations graduelles en résultat. Les opérations de l'algèbre relationnelle instanciée par SQL sont alors des cas particuliers des opérations étendues. Il s'agit de la sélection, de la projection, de la jointure, d'opérateurs ensemblistes, et de prédicats graduels (par exemple, les prédicats bonne et chère de la Figure 4.3, définis respectivement sur les attributs qualité et prix).

4.3 Résume de données

Un résumé d'un ensemble de données est une représentation compacte et condensée de cet ensemble. Il permet notamment de mieux exploiter ces données et de les interroger en un temps acceptable.

Un certain nombre de méthodes de construction de résumé réalise une approximation des données en utilisant des termes du langage naturel (modélisés au moyen des ensembles flous). Dans le cas des bases de données, les valeurs des attributs sont décrites par ces termes. Les descriptions obtenues matérialisent les concepts qui existent au sein des enregistrements, par exemple, « les personnes jeunes et bien payées ». Chacun des enregistrements est rattaché à un ou plusieurs concepts suivant l'adéquation entre ses valeurs d'attribut et les termes qui étiquettent le concept (*jeune* et *bien payé*). Ces méthodes s'inscrivent dans le cadre des résumés linguistiques de Yager [YAG 91] (Le modèle SaintEtiQ fait partie de ces méthodes). Les résumés de SAINTETIQ se distinguent en utilisant un même ensemble de termes linguistiques (le *vocabulaire*) sans assignation de niveau. Les descriptions générées sont donc unifiées sur le plan des termes (ou *étiquettes*) linguistiques. Ce modèle permet de produire une représentation compacte d'une base de données (1) en réécrivant les tuples à l'aide de variables linguistiques définies sur chaque attribut; (2) et en les regroupant dans une hiérarchie de résumés.

4.3.1 Approche de résumé SAINTETIQ

SaintEtiQ prend en entrée deux types d'informations : les données à résumer, et les données relatives au domaine (appelées des connaissances sur le domaine). Les données à résumer sont des données relationnelles au sens des bases de données. À ce titre, elles sont organisées en enregistrements (ou « tuples ») qui suivent le schéma d'une relation R définie sur un ensemble d'attributs $A = (A_1; A_2; \ldots ; A_n)$. Chaque attribut A_i est défini sur un domaine, noté $D(A_i)$, qui peut être numérique ou symbolique. Ainsi, un enregistrement t est un tuple formé d'une suite de valeurs suivant l'ordre prédéfini des attributs A_i. Une autre contrainte sur les données est leur complétude : toutes les valeurs d'attributs doivent être présentes. Pour tout enregistrement t d'une relation R, la valeur $t.A_i$ est nécessairement connue, élémentaire, précise et certaine. Les données incomplètes, incertaines ou mal connues ne sont donc pas traitées.

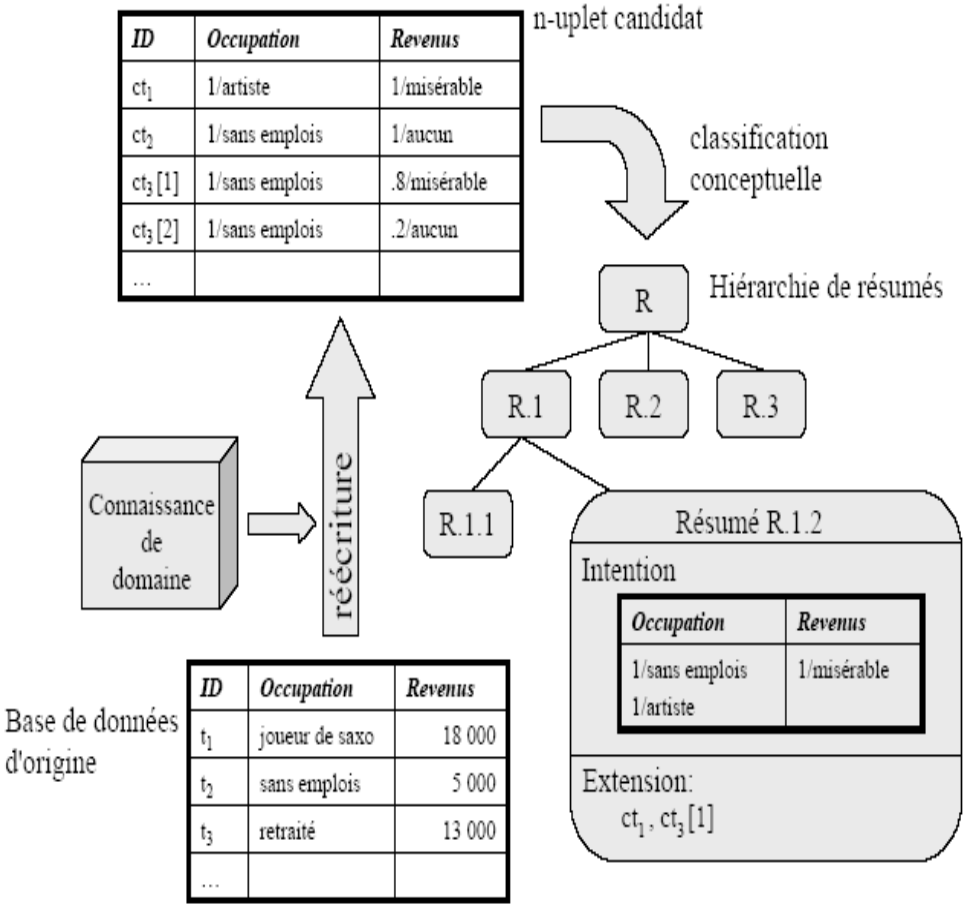


Figure 4-2 Principe du système SAINTETIQ [RNM 02]

Les connaissances sur le domaine fournissent le vocabulaire d'expression des résumés. Ces connaissances sont constituées essentiellement de variables linguistiques définies sur les domaines d'attributs de la relation considérée (voir par exemple la Figure 4.3 pour les attributs "Prix" et "Qualité"). Elles sont données par l'utilisateur ou un expert afin de définir un langage de description des données dont la sémantique soit la plus proche possible de l'utilisateur. D'une manière générale, les connaissances de domaine permettent de faire la correspondance entre les valeurs de domaines d'attribut et un vocabulaire d'expression des résumés de données.

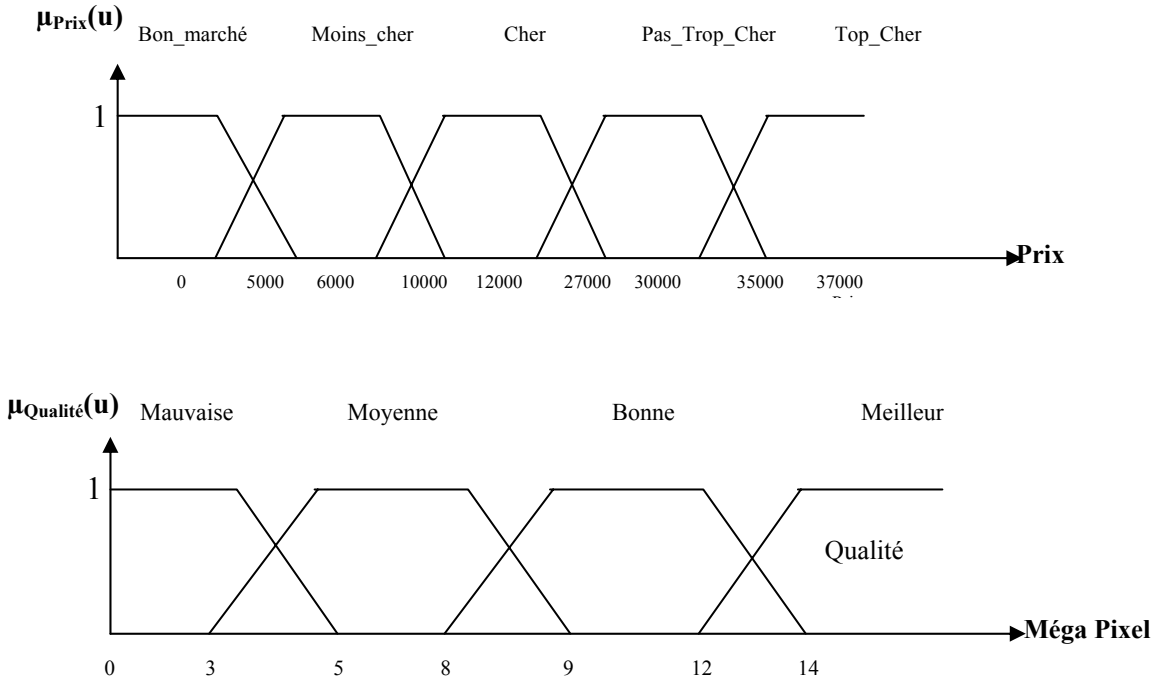


Figure 4-3 Variables linguistiques décrivant les prédicats "Prix" et "Qualité"

4.3.2 Construction de résumé SAINTETIQ

SaintEtiQ prend en entrée les enregistrements de la base de données et fournit une forme de connaissance en sortie. La Figure 4.2 donne une vue d'ensemble de l'architecture du système. L'acte de résumer est alors conçu comme un processus de découverte de connaissances à partir des données. Le modèle SaintEtiQ peut se décomposer en deux grandes phases : (i) la réécriture; et (ii) l'organisation du résumé.

4.3.2.1 Phase de réécriture

Cette étape permet au système de réécrire les tuples de la base de données avant que ceux-ci ne soient exploités par le service du résumé (un tuple réécrit est appelé tuple candidat). Cette étape donne naissance à un ou plusieurs tuples candidats qui peuvent être considérés comme des représentations linguistiques d'un tuple de la base.

Par exemple dans le Tableau 4.2, la valeur 36900 pour le prix de la caméra S4(36900, 13.6) peut être interprétée comme *Pas_trop_cher* (*P_T_Ch*) (avec un degré de 0,1) et *Trop_cher* (*T_Ch*) (avec un degré de 0,9). Concernant la "Qualité", la valeur 13.6 représente une meilleure qualité (avec 1 comme degré d'appartenance à l'ensemble floue *Meilleur,* dénoté par *Meil*). Ainsi, le tuple S4 est réécrit en deux tuples candidats $Ct_{4,1}$ *(0.1/P_T_Ch, 1.0/Meil) et* $Ct_{4,2}$ *(0.9/T_Ch, 1.0/Meil).* Le Tableau 4.2 donne les tuples candidats du site1 (décrit dans le Tableau 4.1) réécrits à l'aide des variables linguistiques définies sur le "Prix" et la "Qualité" (Figure 4.3), l'attribut modèle sert juste pour l'identification des caméras.

4.3.2.2 Phase d'organisation du résumé

Ce service consiste à organiser les résumés au sein d'une hiérarchie de manière à ce que le résumé le plus général soit placé au sommet de cette hiérarchie et les résumés les plus spécifiques au niveau des feuilles. Le résumé racine décrit ainsi l'intégralité du jeu de données tandis que les feuilles ne résument qu'une partie plus limitée de la base. Les résumés "feuille" contiennent les tuples candidats, ils sont vus comme des classes de tuples similaires. Plus de détails dans [RNM 02].

Pour le site 1 par exemple, après que tous les tuples candidats sont produits (Tableau 4.2), ils sont passés au service de résumé qui permet de grouper et classifier les tuples candidats dans une hiérarchie comme il est montré dans le Tableau 4.3. Pour un résumé (e.g. Z_{f1}) son degré de satisfaction pour un attribut (e.g. Prix) est égal au maximum des degrés de satisfaction de ses tuples candidats. Z_{f1} couvre les tuple $\{Ct_1, Ct_{2,1}, Ct_{3,1}, Ct_{12,2}\}$ et donc le degré de Z_{f1} pour le prix vaut:

$$\text{Max } (1, 0.7, 1, 1) = 1.$$

Tableau 4-2 Exemples de réécriture (Moy : Moyenne, Mauv : Mauvaise)

Modèle	Prix	Qualité	Tuples candidats
S1	16500	7,2	Ct_1 (1.0/Ch, 1.0/Moy)
S2	27500	8,1	$Ct_{2,1}$ (0.7/Ch, 1.0/Moy);
			$Ct_{2,2}$ (0.3/P_T_C, 1.0/Moy)
S3	27000	8,2	$Ct_{3,1}$ (1.0/Ch, 0.8/Moy);
			$Ct_{3,2}$ (1.0/P_T_C, 0.2/Bon)
S4	36900	13,6	$Ct_{4,1}$ (0.1/P_T_C, 1.0/Meil);
			$Ct_{4,2}$ (0.9/T_C, 1.0/Meil)
S5	63500	10,1	$Ct_{5,1}$ (1.0/T_C, 0.2/Bon);
			$Ct_{5,2}$ (1.0/T_C, 0.8/Meil)
S6	35000	8,9	$Ct_{6,1}$ (1.0/P_T_C, 0.1/Moy);
			$Ct_{6,2}$ (1.0/P_T_C, 0.9/Bon)
S7	29000	9,5	$Ct_{7,1}$ (0.2/Ch, 1.0/Bon);
			$Ct_{7,2}$ (0.8/P_T_C, 1.0/Bon)
S8	39500	8,3	$Ct_{8,1}$ (1.0/T_C, 0.7/Moy);
			$Ct_{8,2}$ (1.0/T_C, 0.3/Bon)
S9	41200	14	Ct_9 (1.0/T_C, 1.0/Meil)
S10	53700	14,7	Ct_{10} (1.0/T_C, 1.0/Meil)
S11	30000	12,3	$Ct_{11,1}$ (1.0/P_T_C,0.1/Moy);
			$Ct_{11,2}$ (1.0/P_T_C,0.9/Bon)
S12	14900	5,5	$Ct_{12,1}$ (1.0/Ch, 0.1/Mauv);
			$Ct_{12,2}$ (1.0/Ch, 0.9/Moy)

D'une manière similaire, on peut obtenir les hiérarchies de résumés Z2 et Z3 pour les sites 2 et 3 respectivement :

Z2 = (0.5/B_Marché, 1.0/M_Ch, 0.5/Ch, 1.0/T_Ch, 1.0/Moyen, 1.0/Meil).
Z3 = (1.0/M_Ch, 1.0/Ch, 1.0/P_T_C, 1.0/T_C, 1.0/Moy, 1.0/Bon, 1.0/Meil).

Tableau 4-3 Classification des tuples candidats

Résumé Z	Intension	Tuple couvert par Z
Z_{f1}	(1.0/Ch, 1.0/Moy)	Ct_1, $Ct_{2,1}$, $Ct_{3,1}$, $Ct_{12,2}$
Z_{f2}	(1.0/P_T_C, 1.0/Moy)	$Ct_{2,2}$, $Ct_{11,1}$
Z_{f3}	(1.0/Ch, 1.0/Bon)	$Ct_{3,2}$, $Ct_{7,1}$
Z_{f4}	(0.1/P_T_C, 1.0/Meil)	$Ct_{4,1}$
Z_{f5}	(1.0/T_C, 1.0/Meil)	$Ct_{4,2}$, $Ct_{5,2}$, Ct_9, Ct_{10}
Z_{f6}	(1.0/T_C, 0.3/Bon)	$Ct_{5,1}$, $Ct_{8,2}$
Z_{f7}	(1.0/P_T_C, 1.0/Bon)	Ct_6, $Ct_{7,2}$, $Ct_{11,2}$
Z_{f8}	(1.0/T_C, 0.7/Moy)	$Ct_{8,1}$
Z_{f9}	(1.0/Ch, 0,1/Mauv)	$Ct_{2,1}$
Z_{11}	(1.0/Ch, 1.0/P_T_C, 1.0/T_C , 1.0/Moy)	Z_{f1}, Z_{f2} , Z_{f8}
Z_{12}	(1.0/Ch, 1.0/P_T_C, 1.0/T_C , 1.0/Bon)	Z_{f3}, Z_{f6} , Z_{f7}
Z_{13}	(1.0/P_T_C, 1.0/T_C , 1.0/Meil)	Z_{f4}, Z_{f5}
Z'_1	(1.0/Ch, 1.0/P_T_C, 1.0/T_C , 0,1/Mauv , 1.0/Moy, 1.0/Bon,)	Z_{11} , Z_{12} , Z_{f9}
Z_1	(1.0/Ch, 1.0/P_T_C, 1.0/T_C, 0,1/Mauv, 1.0/Moy, 1.0/Bon, 1.0/Meil)	Z_{13} , Z'_1

La Figure 4.4 montre l'hiérarchie de résumés de la relation caméras.

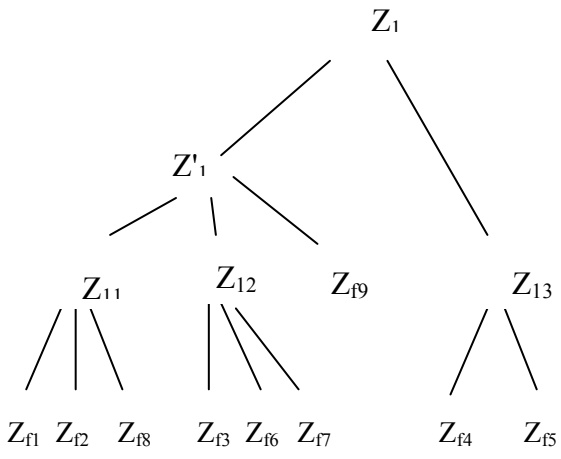


Figure 4-4 Hiérarchie des résumés.

4.3.3 Représentation des résumés

Un résumé z peut être représenté par une paire (I_z, R_z) où I_z (resp. R_z) est appelé l'intension (resp. extension) de z. Le sous-ensemble des tuples de la base de donnée impliqué dans le résumé z est habituellement appelé *l'extension*, alors que la description linguistique de ces tuples est appelée *l'intention*. Par exemple, l'intension du résumé Z_{f1} ((1.0/Ch, 1.0/Moy)) est I_{Zf1}= *{Cher, Moyenne}* et son extension R_{Zf1}= $\{Ct_1, Ct_{2,1}, Ct_{3,1}, Ct_{12,2}\}$.

4.3.4 Interrogation des résumés

Dans [VRU 06], un algorithme pour l'interrogation d'une hiérarchie de résumés SaintEtiq est proposé. L'idée de base consiste à parcourir l'arbre des résumés afin de trouver les réponses satisfaisant la requête. Le résultat de ce processus est donc un ensemble de résumés. Dans cette section, la manière dont la requête est formulée et l'algorithme de sélection des résumés sont présentés.

Soit la relation R (décrivant des caméras, voir Tableau 4.1) et la requête flexible Q = "Pas_Trop_Cher $\wedge$ Bonne" qui signifie que la recherche de l'utilisateur concerne les caméras qui ne sont pas trop_chères et de bonne qualité. L'ensemble des étiquettes linguistiques apparaissant dans une requête sur l'attribut A_i sont appelées caractères requis, dénotées par C_{Ai}. L'ensemble des C_{Ai}, désigné par C, est appelé caractérisation initiale liée à la requête posée. Par exemple, C = [Pas_trop_Cher, Bonne] pour la requête Q. Le test de correspondance entre les caractères requis C_{Ai} des attributs de la requête et les descripteurs $L_{Ai}(Z)$ extraits d'intension de Z, est de trois types :

Correspondance nulle : il existe au moins un attribut A_i pour lequel Z ne montre aucun des caractères requis pour A_i.

Correspondance exacte : pour tous les attributs de la requête, le résumé Z présente uniquement les caractéristiques recherchées, il est considéré comme résultat si Z est une feuille.

Correspondance par excès : cette situation se présente lorsque le résumé Z dispose sur un ou plusieurs attributs plus de descripteurs que ceux de la requête, donc l'exploration du sous-arbre de racine Z est nécessaire.

La sélection des résumés est réalisée par la fonction Recherche de l'algorithme 1. Cette fonction est récursive, elle prend en entrée un résumé Z et la caractérisation initiale C de la requête. Elle retourne en sortie une liste de résumés Lres.
Si on applique l'algorithme 1 à l'hiérarchie de résumés du Site 1 (Figure 4.4) et la requête Q, on obtient le résultat suivant $\{Z_{f7}\}$ qui couvrent les tuples $\{Ct_{61}, Ct_{7,2}, Ct_{11,2}\}$.

Algorithme 1. Fonction Recherche (z, C)

Entrée : Résumé z et la caractérisation C

1. Lres := ∅ /* la liste est vide */
2. **Si** Corr(z, C) = excès **Alors**
3. **Pour** chaque nœud fils zfils de z **Faire**
4. Lres := Lres + Recherche (zfils, C)
5. **Fin pour**
6. **Sinon**
7. **Si** Corr(z, C) = exacte **Alors**
8. **Si** z est une feuille **Alors**
9. ajouter (z, Lres)
10. **Sinon**
11. Lres := Lres + Recherche (z, C)
12. **Fin si**
13. **Fin si**
14. **Fin si**

Résultat : Lres /* liste de résumé retournée */

4.4 Evaluation de requêtes flexibles dans un contexte distribué : Une approche hybride

Dans cette partie, le problème considéré concerne l'évaluation des requêtes flexibles [BLP 04] dans un contexte de bases de données distribuées. Ces requêtes permettent aussi d'exprimer des préférences au moyen de prédicats flous (comme "jeune", "cher", etc.) dont la satisfaction est graduelle. Dans ce cadre, le résultat d'une requête n'est plus un ensemble "plat" d'éléments mais un ensemble où chaque élément est affecté d'un degré de satisfaction.

Pour autant que nous le sachions, il n'existe pas de travaux sur cette problématique dans la littérature, excepté ceux proposés dans [BHJ 07] [HRV 08] où l'aspect évaluation des requêtes n'a pas été suffisamment traité. L'approche proposée combine l'algorithme décrit dans [ZIN 05] pour la construction d'index de routage et le modèle de résumé présenté dans [RNM 02] pour la définition du contenu de l'index. En particulier, un algorithme d'évaluation de requêtes flexibles dans un environnement P2P est proposé. Il permet de renvoyer les réponses qui satisfont au mieux la requête posée. Un exemple illustratif est utilisé dans ce chapitre. Il s'agit d'une base de données répartie sur trois sites et décrivant un ensemble de caméras (voir Tableau 4.1).

La base de données est supposée décrite par le schéma Caméra (Modèle, prix, Qualité). Ce schéma (global) est celui de chaque instance de chaque site (les problèmes soulevés par la médiation de données ne sont pas considérés ici).

4.4.1 Procédure de construction de l'index de routage

Cette section est consacrée à la démarche proposée pour la construction d'un index global permettant le routage dans le réseau P2P. Il est montré dans [VRU 06] qu'un résumé SaintEtiQ peut être considéré comme un index multidimensionnel. L'idée suggérée ici est de construire une hiérarchie de résumés (résumé global) qui décrit toutes les données du réseau P2P. Rappelons tout d'abord que le modèle P2P considéré est un système pair à pair décentralisé non structuré où la recherche se fait à l'aide des index de routages. Les hypothèses de base suivantes sont supposées vérifier:

- Les données sont arbitrairement distribuées sur les pairs.
- Chaque pair stocke et partage sa base de données locale.

- Tous les attributs impliqués dans la requête figurent dans les différentes bases de données distribuées.
- Chaque pair maintient un résumé local de sa propre base. Tous les pairs coopèrent pour construire un résumé global décrivant l'ensemble des données du réseau. On suppose aussi que les mêmes connaissances du domaine sont utilisées pour résumer les données de toutes les sources.
- Le problème d'hétérogénéité (et l'intégration de schémas) n'est pas considéré ici. Un schéma global est supposé.

Dans un mécanisme de routage efficace, la requête est envoyée uniquement vers les pairs qui sont susceptibles de répondre à la requête. On considère ici qu'une hiérarchie de résumés sert comme un index multidimensionnel, on ajoute un troisième terme, Pz, à la définition d'un résumé. Ce terme, dit Peer-extent [HRV 08], fournit l'ensemble des pairs qui ont des données décrites par le résumé z. On suppose aussi que dans la phase de réécriture, chaque tuple candidat contient un identificateur de son tuple original.

Le processus de construction de l'index est le même que celui présenté dans [ZIN 05], la seule différence réside dans la manière de résumer les données. Dans notre cas, l'ensemble des données locales pour chaque pair est résumé à l'aide du modèle SaintEtiQ. La procédure de construction de l'index global (résumé global) qui décrit toutes les données du réseau P2P est illustrée par les étapes suivantes:

1- Une phase d'initialisation est faite sur chaque pair. L'ensemble des données locales est résumé à l'aide du modèle SaintEtiQ et un index local est créé pour chaque pair.
2- Tous les nœuds feuilles envoient une copie des informations sur leurs données (synthétisées sous forme d'une hiérarchie de résumés) vers ses voisins dans le réseau de P2P (Figure 3.3-a).
3- Tous les pairs qui ont reçu des messages depuis leurs voisins sauf un (appelé N), fusionnent leurs informations locales (disponibles sous forme de résumé) avec toutes les informations reçues des différents voisins à l'exception du nœud N. Puis, ils envoient le résumé fusionné à N (Figure 3.3-b). L'opération de fusion suit exactement le modèle SaintEtiQ comme expliqué en Section 4.3.2 et elle est illustrée dans l'exemple 1 ci-dessous.

4- L'étape 3 est répétée itérativement jusqu'à ce que le nœud central reçoive des informations de tous leurs voisins (Figure3.3-c).
5- Le nœud central produit un résumé global (après la fusion de tous les résumés de ses voisins avec son résumé local). Ce résumé décrit toutes les données disponibles dans le réseau P2P, il sert comme un index sur les données du réseau (Figure 3.3-d).
6- La dernière étape consiste à faire l'opération inverse (le résumé global est diffusé dans le même chemin mais de haut en bas), et l'algorithme se termine quand les nœuds feuilles reçoivent l'index de routage global (Figure 3.3- e et f).

Exemple 1. Reprenons notre exemple des trois sites décrivant des caméras (voir Tableau 4.1). Supposons que les trois sites forment un réseau P2P avec le site 1 et le site 3 comme nœuds feuilles et le site 2 comme nœud central. Les pairs 1, 2 et 3 résument leurs données locales par les résumés Z1, Z2 et Z3, voir Section 4.3.2. Les deux pairs feuilles (site 1, site 3) envoient leurs résumés de données (index locaux) vers leur voisin (le nœud central : site 2). Le pair 2 fusionne les résumés reçus de ses voisins avec son propre résumé et délivre un résumé Z global (index global) qui décrit l'ensemble des données du réseau. Enfin, l'index Z, voir Figure 4.5, est renvoyé aux nœuds feuilles. Le Tableau 4.4 présente l'index local de chaque pair ainsi que l'index global construit.

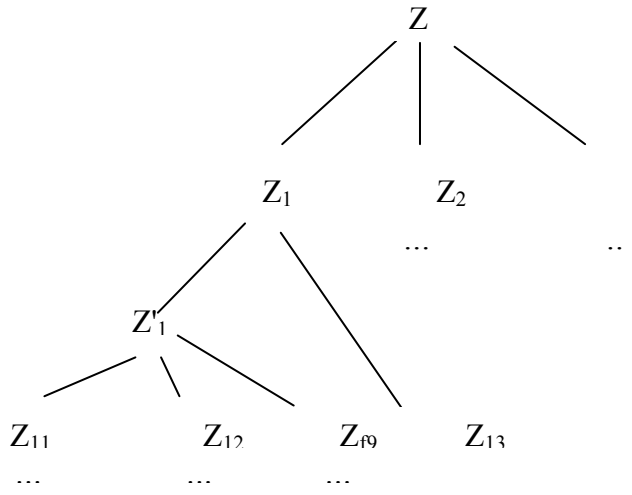


Figure 4-5 Hiérarchie de résumés pour le réseau P2P

Tableau 4-4 Résumé global pour le réseau P2P

Site 1 (Z_1)	(1.0/Ch, 1.0/P_T_C, 1.0/T_C, 0,1/Mauv, 1.0/Moy, 1.0/Bon, 1.0/Meil)
Site 2 (Z_2)	(0.5/B_Marché, 1.0/M_Ch, 0.5/Ch, 1.0/T_Ch, 1.0/Moyen, 1.0/Meil)
Site 3 (Z_3)	(1.0/M_Ch, 1.0/Ch, 1.0/P_T_C, 1.0/T_C, 1.0/Moy, 1.0/Bon, 1.0/Meil)
Résumé global (Z)	({0.5/B_Marché,1.0/M_Ch,1.0/Ch, 1.0/P_T_C, 1.0/T_C}, {0.1/Mauv, 1.0/Moy, 1.0/Bon, 1.0/Mei}

4.4.2 Stratégie d'Evaluation

Dans cette section, on présente une stratégie efficace pour localiser les pairs pertinents à une requête posée. Cette localisation des pairs pertinents est réalisée en interrogeant l'hiérarchie des résumés (l'index global). La solution proposée consiste à adapter l'algorithme 1 dans le sens où la réponse retournée n'est plus un ensemble de résumés, mais un ensemble de pairs. Ensuite, un algorithme de recherche des meilleurs tuples satisfaisant la requête est proposé. Cette recherche se fait en interrogeant la base de données originale par le biais d'un index local.

4.4.2.1 Localisation des pairs

Lorsqu'une requête est posée sur un pair, l'index (hiérarchie globale) est exploré à l'aide de l'algorithme 2 (une version adaptée de l'algorithme 1) donné ci-dessous. Rappelons que chaque résumé z contient dans sa description les pairs (P_z) qui sont décrits par ce résumé. Soit P_Q l'ensemble des pairs pertinents pour la requête. Le principe de l'algorithme est : pour chaque nœud z du résumé qui correspond exactement à la requête, P_z est ajouté à l'ensemble P_Q. Puis, la requête est propagée vers l'ensemble des pairs contenus dans P_Q. Chaque pair qui a reçu le message, doit vérifier la satisfaction de la requête par rapport à ses données stockées localement (voir la section suivante) et ainsi identifier les meilleurs tuples. Ensuite, il renvoie la réponse vers l'initiateur de requête. Lorsque le pair initiateur reçoit tous les résultats des

pairs de l'ensemble P_Q, il filtre les résultats et renvoie seulement les réponses les plus satisfaisantes.

Algorithme 2 . Fonction SelectPair (z, C)

Entrée : Résumé z et la caractérisation C

1 . $P_Q := \emptyset$ /* la liste est vide */
2 . **Si** Corr (z, C) = excès **Alors**
3 . **Pour** chaque nœud fils z_{fils} de z **Faire**
4 . $P_Q := P_Q$ + SelectPair (z_{fils}, C)
5 . **Fin pour**
6 . **Sinon**
7 . **Si** Corr (z, C) = exacte **Alors**
8 . **Si** z est une feuille **Alors**
9 . ajouter (P_z, P_Q)
10 . **Sinon**
11 . $P_Q := P_Q$ + SelectPair (z, C)
12 . **Fin si**
13 . **Fin si**
14 . **Fin si**
Résultat : P_Q /* la liste des pairs pertinents pour Q */

4.4.2.2 Evaluation de la requête sur chaque pair

Après que les pairs contribuant à la réponse d'une requête aient été identifiés, il est nécessaire d'évaluer la requête flexible sur les données locales de chaque pair et donc l'hiérarchie des résumés des données du pair (l'index local) est explorée pour trouver les résumés qui satisfont la requête. Comme les résumés feuilles regroupent les tuples candidats qui sont réécrits par les mêmes descripteurs linguistiques, on étend l'algorithme 1 pour qu'il retourne les enregistrements initiaux (tuples) ayant les degrés de satisfaction les plus élevés. L'algorithme 3 illustre cette procédure.

Algorithme 3 . Fonction Recherche_Tuples(z, C)

Entrée : Résumé z et la caractérisation C

1 . L_{Ct} := ∅ /* la liste des tuples candidats est vide */
2 . L_{meil_Ct} := ∅ /* la liste des meilleurs tuples candidats est vide */
3 . $L_{réponse}$:= ∅ /* la liste des réponses est vide */
4 . **Si** Corr (z, C) = excès **Alors**
5 . **Pour** chaque nœud fils z_{fils} de z **Faire**
6 . L_{Ct} := L_{Ct} + Recherche_Tuples (z_{fils}, C)
7 . **Fin pour**
8 . **Sinon**
9 . **Si** Corr (z, C) = exacte **Alors**
10 . **Si** z est une feuille **Alors**
11 . **Pour** un tuple candidat Ct de z **Faire**
12 . ajouter(Ct, L_{Ct})
13 . **Fin pour**
14 . **Sinon**
15 . L_{Ct} := L_{Ct} + Recherche_Tuples (z, C)
16 . **Fin si**
17 . **Fin si**
18 . **Fin si**
19 . L_{meil_Ct} := BNL (L_{Ct})
20 . **Pour** chaque Ct de L_{meil_Ct} **Faire**
21 . retourner le tuple original t
22 . ajouter (t, $L_{réponse}$)
23 . **Fin pour**
Résultat : $L_{réponse}$

La recherche est basée sur un test de correspondance comme dans l'algorithme 1, mais dans ce cas les résumés sont des index sur les données recherchées, quand un résumé feuille z correspond exactement à la requête, l'ensemble des tuples candidats couverts par z est ajouté à la liste des tuples candidats(L_{Ct}). Ensuite, la fonction BNL(Block Nested Loop algorithm) [BKS 01] retourne les meilleurs tuples L_{meil_Ct} parmi la liste L_{Ct}.

Supposons que l'ensemble L_{Ct} contient seulement les deux tuples candidats $Ct_{11,2}$ = (1.0/P_T_C,0.9/Bon) et $Ct_{2,1}$ = (1.0/P_T_Ch, 0.6/Bon). La fonction BNL appliquée sur L_{Ct} retourne comme résultat le tuple candidat $Ct_{11,2}$. En effet, les deux tuples ont le même degré de satisfaction pour le "prix P_T_Ch", mais $Ct_{11,2}$ satisfait le critère "bonne qualité" avec un degré supérieur au degré associé à $Ct_{2,1}$.

Une connexion à la base de données est nécessaire pour chaque tuple candidat Ct de la liste L_{meil_Ct}. Elle permet d'identifier les tuples de la base qui lui correspond. Le résultat final de l'algorithme est donc l'ensemble des tuples satisfaisant au mieux la requête.

Exemple 2 (suite). Considérons la requête floue Q = "Pas_Trop_Cher ∧ Bonne" sur le pair 2 de l'exemple 1. Supposons que l'index de routage est construit, voir section 4.4.1. Le pair 2 commence par exécuter l'algorithme 2 pour déterminer les pairs pertinents. La hiérarchie (l'index global) est explorée à l'aide de l'algorithme 2 pour retourner les pairs pertinents. Premièrement la racine Z correspond la requête par excès, il faut tester ses nœuds fils, le nœud Z_2 à une correspondance nulle par rapport à la requête, donc le sous arbre de racine Z_2 est éliminé et ne peut être considéré par la suite, alors que les sous arbres de racines Z_1 et Z_3 (respectivement) sont considérés et ainsi de suite. Pour Z_1, le résumé (nœud) Z_{f7} correspond exactement à la requête et son pair P_z= {site1}, alors que pour Z_3, P_z= {site3}. Le résultat final de l'algorithme est P_Q = {pair1, pair3}. Maintenant, la requête est routée vers les pairs 1 et 3. Chaque pair exécute l'algorithme 3. Le résultat de cet algorithme sur le pair 1 est le résumé Z_{f7} qui couvre les tuples candidats :

{Ct_{61}, $Ct_{7,2}$, $Ct_{11,2}$} avec $Ct_{6,1}$ = (1.0/P_T_C, 0.9/Bon), $Ct_{7,2}$ = (0.8/P_T_C, 1.0/Bon) et $Ct_{11,2}$ = (1.0/P_T_C, 0.9/Bon).

Les tuples de la base retournés sont donc {S6, S7, S11}.

De la même façon, on peut vérifier que l'application de l'algorithme 3 sur le site 3 retourne les tuples candidats :

{$Ct_{1,2}$, $Ct_{11,2}$} avec $Ct_{1,2}$ = (0.83/P_T_C, 1.0/Bon) et $Ct_{11,2}$ (1.0/P_T_C,0.9/Bon).

Les tuples de la base retournés sont donc {X1, X11}.

Ensuite les deux pairs envoient leurs réponses au site initiateur de la requête, le site 2, qui fusionne ces résultats et délivre la réponse finale à la requête sous forme d'un ensemble flou :

Réponse = {*min (1.0, 0.9)/S6; min (1.0, 0.9)/S11; min (0.83, 1.0)/X1; min (1.0, 0.9)/X11*}.

= {*0.9/S6; 0.9/S11; 0.83/X1; 0.9/X11*}.

Conclusion

Dans ce chapitre, nous avons abordé le problème d'évaluation des requêtes flexibles dans un système P2P. La solution proposée permet de combiner le principe de l'index de routage global et distribué et le concept de résumé de données pour évaluer efficacement des requêtes floues.

En résumé, les apports principaux de ce chapitre pour l'évaluation des requêtes flexibles sont les suivants:

- Une démarche est proposée pour la construction d'index de routage (qui décrit l'ensemble des données du réseau). Elle est fondée sur la méthode de Zin [ZIN 05] pour la construction d'index et le modèle de résumé de données SaintEtiQ pour la définition du contenu d'index.

- Un premier algorithme est proposé et permet la localisation des pairs pertinents pour la requête en interrogeant l'index global (l'hiérarchie des résumé).

- Le second algorithme, exécuté au niveau de chaque pair pertinent, retourne les réponses (tuples ou enregistrement) contenues dans les sources associées aux pairs du système, qui satisfont au mieux la requête.

Conclusion Générale

Habituellement, l'interrogation classique de bases de données est qualifiée d'interrogation booléenne (ou rigide) dans le sens où l'utilisateur lorsque il formule une requête, avec SQL par exemple, obtient une réponse ou rien du tout. Ce type d'interrogation s'avère inadéquate et inefficace pour certaines applications, qui supportent des données vagues, imprécises et/ou des préférences dans les attributs. L'interrogation flexible constitue alors une alternative à l'interrogation booléenne pour ces types d'applications.

Dans ce mémoire, nous avons étudié les requêtes à préférences sous l'angle de l'interrogation flexible. L'objectif principal de cette étude est de présenter une méthode adéquate pour l'évaluation des requêtes flexibles dans un environnement distribué, illustré par un système P2P.

Dans le chapitre 1, nous avons décrit le principe des requêtes à préférences et sa modélisation dans le cadre des SGBD relationnel. Nous avons également étudié l'approche des requêtes Skyline, qui représente un des formalismes les plus utilisés pour représenter les préférences des utilisateurs dans les requêtes. Dans le chapitre suivant, nous avons présenté les caractéristiques principales des systèmes pair à pair (P2P) et abordé le problème de localisation des données pertinentes dans ces systèmes. Plus particulièrement la technique des index de routage (Rounting indexes) est détaillée. Le chapitre 3 explique quelques approches pour évaluer les requêtes Skylines dans un cadre décentralisé. Par exemple, le travail de Zinn qui représente la base de notre proposition. En fin nous avons donné les notions de base de l'interrogation flexible et expliqué notre solution.

A l'issu de cette étude, nous avons dans un premier temps proposé une démarche (pour la localisation des pairs pertinents) qui combine l'approche de Zinn pour la construction d'index de routage et le modèle de résumé SaintEtiQ pour la définition du contenu de l'index.

Ensuite, nous avons exposé les étapes de base pour l'évaluation des requêtes flexibles dans un environnement P2P. L'algorithme fourni est illustré de manière détaillée via un exemple.

Plusieurs perspectives futures s'imposent. Nous citons quelques unes qui nous semblent les plus importantes :

- La maintenance de l'index de routage construit, en termes d'insertion, suppression et modification.
- Etude des performances des algorithmes développés en termes de temps de réponse et de nombre de messages échangés dans le réseau.

Bibliographie

[BEZ 93] J. C. Bezdek. Fuzzy Models – What Are They, and Why?. IEEE Transactions on Fuzzy Systems,Vol.No.1,February 1993.

[BGZ 04] Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. Efficient distributed skylining for web information systems. In EDBT, pages 256–273, 2004. 3.2.4.

[BHJ 07] P. Bosc, A. Hadjali, H. Jaudoin, O. Pivert, Flexible querying of multiple data sources through fuzzy summaries, *Proc. of FlexDBIST'07*, in conjunction with *DEXA'07*, Regensburg, Germany, September 3-7, pp. 350-354, 2007.

[BKS 01] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In IEEE Conf. on Data Engineering, pages 421–430, Heidelberg, Germany, 2001.

[BLP 98] P. Bosc, L.Liétard, O.Pivert. "Bases de données et Flexibilité: Les requêtes Graduelles". Techniques et sciences Informatiques 17 (3), PP. 355-378, 1998.

[BLP 04] P. Bosc, L. Liétard, O. Pivert, D. Rocacher, Gradualité et imprécision dans les bases de données, Paris, Editions Ellipses, 2004.

[BMP 01] P.Bosc,A.Motro et G. Pasi. Report on the fourth international conference on flexible query answering systems. SIGMOD Record, 30 (1) :66–69.2001.

[BOP 92] P. Bosc, O.Pivert. "Somme Approches for relationnal Database Flexible Querying". International Journal of Intelligent Information Systems,1, PP.323-354. 1992.

[BOP 95] P. Bosc, O.Pivert. SQLf: a relational database language for fuzzy querying. IEEE Transactions on Fuzzy Systems, 3(1):1–17,1995.

[BRU 00] Les technologies pair à pair, Bruno Richard, 2000, Hewlett packard (Laboratoire Grenoble).

[CGM 02] A. Crespo, H. Garcia-Molina, Routing Indices For Peer-to-Peer Systems, Inter. Conference on Distributed Computing Systems, 2002.

[CHO 03] J. Chomicki, Preference formulas in relational queries, ACM Transactions on Database Systems, 27, 2003, pp. 153-187.

[CON 99] Connan F., Interrogation flexible de bases de données multimédia, Thèse de doctorat, Université de Rennes I, 1999.

[CRO 03] Cross V., Defining fuzzy relationships in object models: abstraction and interpretation ,Fuzzy Sets and Systems, 140, (1) 16, 2003, p. 5-27.

[DHP 80] D. Dubois and H. Prade. Fuzzy Sets and Systems : Theory and Applications. Academic Press, London, 1980.

[Emule] http://www.emule-project.net.

[Fas] http://en.wikipedia.org/wiki/FastTrack. 2.2.1.

[FNet] http://freenet.sourceforge.com.

[Gnu] http://www.gnutella.com.

[HAC 99] J. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Rotha, and P. Haas. Interactive data analysis: The control project. IEEE Computer, 32(8):51–59, 1999.

[HKD 08] A. Hadjali, S. Kaci, H. Prade, Database preferences queries – A possibilistic logic approach with symbolic priorities, Proc. of the 5th Inter. Symposium on Foundations of Information and Knowledge Systems (FoIKS'08), February 11-15, Pisa (Italy), LNCS 4932, pp. 291–310, 2008.

[HLS 06] K. Hose, C. Lemke, K. Sattler, Processing relaxed skylines in PDMS using distributed data summaries, In Proc. CIKM'06, USA, 2006.

[HOS 05] K. Hose, Processing Skyline queries in P2P systems, VLDB 2005 PhD Workshop, Trondheim, 2005.

[HRV 08] R. Hayeky, G. Raschiay, P. Valduriezz and N. Mouaddib: Summary Management in P2P Systems, EDBT'08, 2008, Nantes, France.

[Ka] http://www.kazaa.com.

[KRR 02] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for Skyline queries. In Proceedings of the International Conference on

Very Large Data Bases (VLDB), pages 275–286, Hong Kong, China, August 20-23, 2002.

[LTL 06] H. Li, Q. Tan W. C. Lee: Efficient Progressive Processing of Skyline Queries in Peer-to-Peer Systems, International Conference on Scalable Information Systems (INFOSCALE'06), 2006.

[LYL 05] E.Lo , K. Y. Yip, K. Lin, D. W. Cheung Progressive skylining over Web-accessible databases, Data & Knowledge Engineering 57 (2006) 122–147.

[MBG 04] A.MARIAN, N.BRUNO et L.GRAVANO: Evaluating Top-k Queries over Web-Accessible Databases in ACM Transactions on Database Systems, Vol. 29, No. 2, June 2004, Pages 1–44.

[MDM 02] Petar Maymounkov and David Mazires. Kademlia: A Peer to-peer Information System Based on the XOR Metric. In The 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), 2002.

[MNR 02] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In In Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02), August 2002.

[Nap] http://www.napster.com.

[PTF 03] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for Skyline queries. In Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003.

[PTF 05] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. ACM Transactions on Database Systems (TODS), 30(1):41–82, 2005.

[RFH 01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. In Proceedings of the ACM SIGCOMM' 01 Conference, Berkeley, CA, August2001.

[RNM 02] G.Raschia and N.Mouaddib. A fuzzy set- based approach to database summarization. Fuzzy sets and systems 29(2), pp. 137– 162, 2002.

[RPD 01] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 329-350 2001.

[SMN 01] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. IEEE Transactions on Networking, 11, 2003.

[TEO 01] K.-L. Tan, P.-K. Eng, and B.C. Ooi. Efficient progressive Skyline computation. In Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 301–310, Roma, Italy, September 11-14, 2001.

[VRU 06] W.A. Voglozin· G.Raschia· L.Ughetto· N.Mouaddib: Querying a summary of database. J Intell Inf Syst (2006)26:59–73.

[WOT 05] S. Wang1, B.C.Ooi , A. K. H. Tung, L. Xu: Efficient Skyline Query Processing on Peer-to-Peer Networks, National University of Singapore 2005.

[YAG 91] Ronald R. YAGER. On linguistic summaries of data. Knowledge Discovery in Databases, pages 347–366. MIT Press,1991.

[ZAD 65] L. Zadeh. Fuzzy sets. Information and Control, 8:338–353. 1965.

[ZAD 76] L. Zadeh. Concept of a linguistic variable and its application to approximate reasoning-III. Information Systems, 9:43–80, 1976

[ZIN 05] D. Zinn, Skyline Queries in P2P Systems. Master's Thesis, TU Ilmenau, Germany, 2005.

[ZKJ 00] BenY.Zhao, John D.Kubiatowicz, and Anthony D.Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB//CSD-01-1141, U. C. Large Data Bases (VLDB), pages 275–286, Hong Kong, China, August 20-23, 2002.Berkeley, April 2000.

Liste des tableaux

Liste des figures

Annexe A

Théorie des Ensembles Flous

Cette annexe est limitée à la présentation des notions clés indispensables à la lecture du chapitre 4.

L'idée sous-jacente à la notion d'ensemble flou [ZAD 65] est de pouvoir représenter des catégories ou classes d'éléments dont la frontière entre appartenance et exclusion n'est pas brutale mais au contraire *graduelle*. Cette démarche est particulièrement adaptée pour traiter de nombreux concepts de la vie courante (et de termes utilisés en langage naturel) pour lesquels il existe une transition progressive entre *appartenance totale* et *non appartenance* en ce sens que le concept décrit par l'ensemble (e.g. température *élevée*, prix *bon marché*, couleur *claire*, etc.) est plus ou moins observé. Si l'interprétation de ce type de terme reste contextuelle, elle se représente mal par un ensemble classique tel que celui représenté par la Figure A.1.

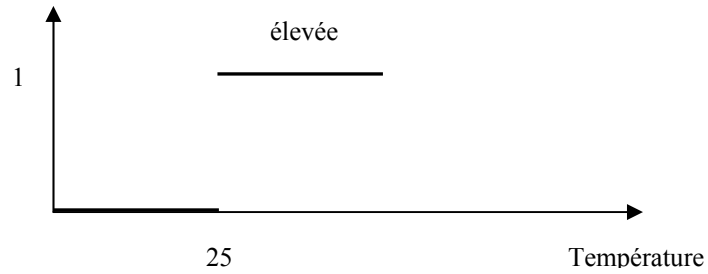


Figure A.1: Le concept "température élevée" représenté par un ensemble usuel

En effet, on y observe une *transition brutale* entre les valeurs correspondant au concept de "température *élevée*" et celles qui y sont étrangères. Ainsi, une température de 25 degrés n'est

pas du tout élevée, alors qu'une autre de 25,1 degrés l'est complètement, ce qui n'est pas conforme au sens commun.

A.1. Ensemble flou

Dans un ensemble de référence X (appelé aussi univers de discours), un sous-ensemble flou E de X est caractérisé par une fonction d'appartenance μ de E dans l'intervalle des nombres réels [0, 1] (degré d'appartenance qui est l'extension de la fonction caractéristique d'un sous-ensemble classique). En fait un sous-ensemble flou (nous dirons plus brièvement un ensemble flou) est formellement défini par l'application μ, La valeur $\mu_E(x)$ exprime dans quelle mesure (à quel point) l'élément x de X appartient à l'ensemble flou E :

$$\mu_E : X \rightarrow [0, 1]$$
$$x \rightarrow \mu_E(x).$$

Quand $\mu_E(x)$ est nul, x n'appartient pas du tout à E et quand il vaut 1, x est complètement dans E (x est aussi dit élément typique de E). Plus $\mu_E(x)$ est proche de 1 (resp. 0), plus (resp. moins) x appartient à E. Un ensemble usuel peut être vu comme un cas particulier d'ensemble flou dont la fonction d'appartenance ne prend que les valeurs 0 et 1.

A.2. Caractéristiques d'un ensemble flou

Les caractéristiques d'un ensemble flou [BEZ 93] détaillées par la suite, sont présentées dans la figure A.2

Support

Le support d'un ensemble flou E, noté supp(E), correspond à l'ensemble usuel des éléments appartenant quelque peu à E (c'est la partie X sur lequel la fonction d'appartenance n'est pas nulle), soit :

$$\text{Supp}(E) = \{x \mid x \in X \text{ et } \mu_E(x) > 0\}.$$

Noyau

Le noyau d'un ensemble flou E, noté noy(E), se définit comme l'ensemble usuel des éléments appartenant complètement à E (l'ensemble des éléments pour lequel la fonction

d'appartenance de E vaut 1), donc :

$noy(E) = \{x \mid x \in X \text{ et } \mu_E(x) = 1\}$.

Hauteur

La hauteur d'un ensemble flou E est donnée par le degré de l'élément d'appartenance maximale à E. E est dit normalisé si sa hauteur vaut 1, soit:

$H(E) = \max_{x \in X} \mu_E(x)$.

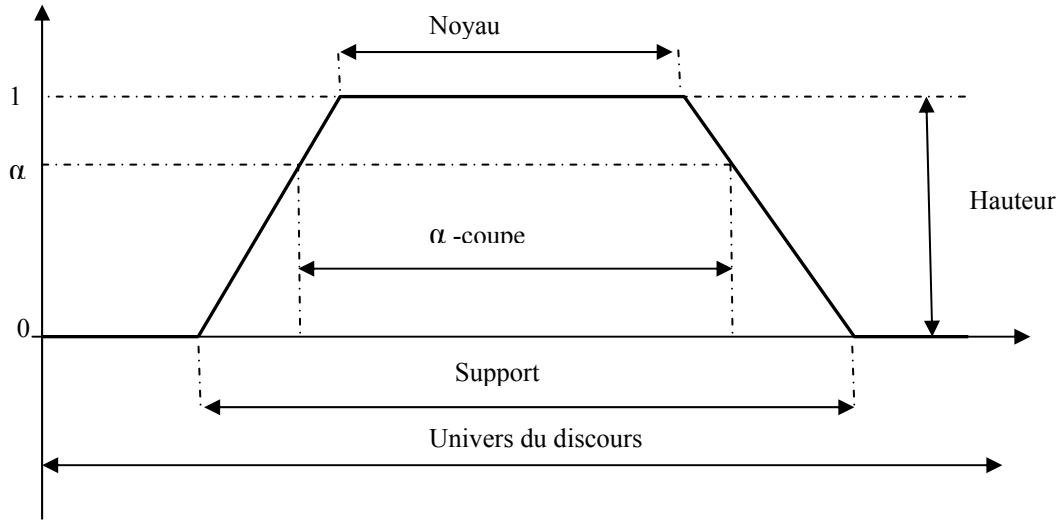


Figure A.2: Support, Noyau, Hauteur et α-coupe d'un ensemble flou

Cardinalité

La cardinalité d'un ensemble usuel est étendue naturellement aux ensembles flous par :

$card(E) = \sum_{x \in X} \mu_E(x)$.

Coupe de niveau α (α-coupes)

Une alpha-coupe est un ensemble strict composé des éléments dont le degré d'appartenance est supérieur ou égal à un seuil "α".

$E_\alpha = \{x \mid x \in X \text{ et } \mu_E(x) \geq \alpha\}$

Exemple A.2.1

Soit X = {a, b, …, j, k} et L = {1/a + 0.7/c + 0.2/e + 0.3/f + 0.6/h + 0.7/j + 0.9/k}. La hauteur de L vaut 1 (L est normalisé) et on a :

• $L_{0.3}$ = {a, c, f, h,j, k} ; $L_{0.5}$= {a, c, h,j, k} ;
• $L_{0.6}$= {a, c, h, j, k} ; $L_{0.8}$= {a, k} ;
• supp(L) = {a, c, e, f, h, j, k} ;
• noy(L) = L_1 = {a} ;
• card(L) = 4.4.

A.3. Opérations sur les ensembles flous

Plusieurs opérateurs d'union, intersection et de complément d'ensembles flous sont définis en fonction des "normes-conormes triangulaires". Nous commençons par définir les notions de normes et conormes triangulaires qui seront utilisées dans les opérations de base sur les ensembles flous.

Définition 1

Une norme triangulaire T est une opération binaire sur l'intervalle [0,1]. Cette opération est associative, commutative, monotone et telle que T (a, 1)=a.

Définition 2

Une conorme triangulaire $\perp$ est une opération binaire sur l'intervalle [0,1]. Cette opération est associative, commutative, monotone et telle que $\perp$ (a, 0)=a.

Il existe plusieurs interprétations de ces normes-conormes. Parmi les normes-conormes les plus utilisées dans la littérature, la norme-conorme "min-max" de Zadeh préserve la plupart des propriétés habituelles des ensembles classiques [DHP 80]. Avec cette norme-conorme, les opérateurs [BEZ 93] suivant sont définis:

Soient E et F deux ensembles fous définis sur l'univers X

1. Inclusion : A $\subseteq$ B définie par $\forall x \in X$, $\mu_E(x) \leq \mu_F(x)$.
2. L'égalité : Si $\forall x \in X$ on a $\mu_E(x) = \mu_F(x)$ alors on dira que A et B sont égaux.

$E=\overline{F}$.

3. Complémentarité : $\forall x \in X\ \mu_E(x) = 1-\mu_F(x)$: notons

4. Intersection: $\mu_{E\cap F}(x) = \min(\mu_E(x), \mu_F(x))$.

5. Union: $\mu_{E\cup F}(x) = \max(\mu_E(x), \mu_F(x))$.

A.4. Variable linguistique

La description d'une situation, d'un phénomène ou d'un procédé contient en général des expressions floues comme : quelque, beaucoup, souvent, chaud, froid, rapide, lent ...etc.

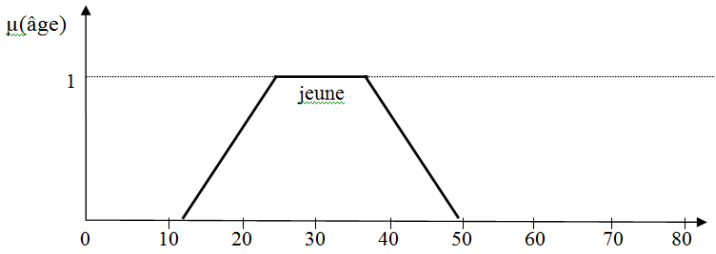


Figure. A.3 : Fonction d'appartenance du sous-ensemble jeune.

Une variable linguistique [ZAD 76] est une variable dont les valeurs sont définies en termes linguistiques, elle est caractérisée par un triplet (X, T(X), U) où X est le nom de la variable, T(X) l'ensemble des termes primaires affectés à la variable, chaque terme est un sous-ensemble défini sur le référentiel U.

Exemple : si on a le mot « **âge** » représenté par une variable linguistique, il aura comme ensemble des termes : T (âge) = {enfant, jeune, vieux} ou chaque terme primaire est l'étiquette d'un sous-ensemble flou dans le référentiel (Figure.A.3).

Remarque : Toutes les fonctions d'appartenance utilisées sont normalisées et sont de type trapézoïdale. C'est-à-dire toute fonction d'appartenance est représentée par le quadruplet (A, B, a, b) où [A, B] (resp. [A – a, B + b]) est le support (resp. le noyau) de la fonction.

ENSSAT
LANNION

ECOLE NATIONALE SUPÉRIEURE
DES SCIENCES APPLIQUÉES ET DE TECHNOLOGIE

Allel HADJALI
Doctorat d'État
Maître de Conférences
Tél : 02 96 46 91 43
Fax : 02 96 37 01 99
E-mail : hadjali@enssat.fr

Rapport sur le Mémoire de Magister Intitulé "Contribution à l'Étude de Requêtes à Préférences dans un Contexte Distribué" Présenté par M. Abdelkader ALEM

Le mémoire présenté par M. Abdelkader ALEM s'inscrit dans un cadre de recherche qui a suscité l'intérêt d'un très grand nombre de chercheurs dans la communauté des bases de données et des systèmes de recherche d'information. Il s'agit de l'intégration des préférences dans les requêtes utilisateurs et la résolution de ce type de requête dans le cas où les sources de données interrogées sont distribuées. Le contexte scientifique du mémoire est donc à l'intersection de deux thèmes de recherche, particulièrement, à la mode et d'actualité. La problématique abordée par l'auteur concerne l'évaluation de requêtes flexibles, où les préférences sont exprimées au moyen de prédicats graduels modélisés à l'aide des ensembles flous, dans un cadre distribué représenté par un système P2P.

Le mémoire est organisé en quatre chapitres :

Le chapitre 1 constitue un rappel sur les requêtes à préférences. Une famille particulière de ces requêtes, dites les Requêtes Skyline, est explicitement étudiée. Les différentes approches proposées pour le traitement des requêtes Skyline, dans un contexte centralisé, sont également examinées et comparées.

Dans le chapitre 2, une synthèse sur les systèmes P2P est présentée. Des notions telles que les index de routage nécessaires à cette étude sont décrites dans un style très intelligible.

Le chapitre 3 est consacré à l'étude des requêtes Skyline dans un contexte distribué. Deux grandes classes d'algorithmes sont discutées. La première classe d'algorithmes concerne le traitement des requêtes Skyline sur le Web en général. La seconde classe s'intéresse aux algorithmes dédiés au traitement des requêtes Skyline dans un système P2P. Malgré la difficulté de la tâche, l'auteur a su bien mener l'étude des différents algorithmes et à illustrer leurs principes via des exemples faciles à comprendre. Ce chapitre constitue le point de départ pour le chapitre suivant.

ÉCOLE NATIONALE SUPÉRIEURE
DES SCIENCES APPLIQUÉES
ET DE TECHNOLOGIE
6 RUE DE KERAMPONT
BP 80518
F-22305 LANNION CEDEX

TÉL. 33 (0)2 96 46 90 00
FAX 33 (0)2 96 37 01 99

www.enssat.fr
www.univ-rennes1.fr

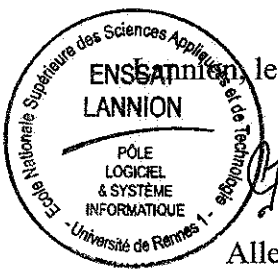
UNIVERSITÉ DE RENNES 1

Le chapitre 4 de ce manuscrit présente la contribution de l'auteur pour introduire une approche hybride d'évaluation de requêtes flexibles dans un système P2P. L'approche proposée combine la technique (de Zinn) pour la construction d'index distribué et la méthode du modèle SaintEtiq pour la construction d'un résumé d'une source de données.
Dan un premier temps, l'auteur propose une démarche permettant de construire un index global de routage qui décrit les données des différentes sources, afin de trouver l'ensemble des pairs pertinent pour la requête posée. Dans un second temps, un algorithme de recherche des réponses satisfaisant le mieux la requête, est discuté.
Il est à noter que l'implémentation et l'intégration de cette approche, dans un système d'interrogation opérationnel, ne sont pas réalisées. Cela est dû notamment au fait que l'auteur ne dispose pas d'environnement adéquat et d'outils de développement nécessaires au sein de son lieu de travail.

Il s'agit d'un travail sérieux et maitrisé sur un sujet assez délicat et d'actualité combinant le paradigme des requêtes à préférences (modélisées en termes de prédicats flous) et des systèmes distribués (illustrés par les systèmes P2P). L'auteur s'est appuyé sur une bibliographie riche et récente ; ce qui démontre sa bonne connaissance de la littérature. Pour autant que nous le sachions, il n'existe pas de travaux antérieurs sur le sujet et que cette étude constitue une première réflexion sur le traitement des requêtes flexibles dans un système distribué. Ce travail peut être poursuivi selon plusieurs directions (comme il a été clairement mentionné en conclusion du manuscrit). Une des directions qui me semble la plus pertinente est la mise en œuvre des algorithmes proposés pour valider l'approche et estimer sa complexité et son coût.

M. ALEM a un article (intitulé : Vers une approche basée sur les résumés distribués pour le traitement des requêtes flexibles dans un système P2P) accepté aux Journées Doctoriales organisées par l'école doctorale STIC, ESI (du 8 au 9 décembre 2009, M'sila). Une version révisée et étendue de cet article sera aussi soumise au Colloque sur l'Optimisation et les Systèmes d'Information (COSI'2010), du 18 au 20 avril 2010, Ouargla.

Par conséquent, M. Abdelkader ALEM peut donc être autorisé sans aucun doute à soutenir son mémoire de Magister.



Lannion, le 14 novembre 2009

Allel HADJALI

Directeur du Mémoire
ENSSAT/IRISA, Université Rennes 1