Proposition d'un cadre générique d'optimisation de requêtes dans les environnements hétérogènes et répartis

Tianxiao Liu

Université de Cergy-Pontoise

THÈSE

pour obtenir le grade de
Docteur

discipline : **Sciences et Technologies de l'Information et de la Communication**

présentée et soutenue publiquement

par

Tianxiao Liu

Le 6 juin 2011 sur le sujet

Proposition d'un Cadre Générique d'Optimisation de Requêtes dans les Environnements Hétérogènes Répartis

Thèse CIFRE en collaboration avec : Progress Software - Xcalia

---

JURY

Madame	**Christine Collet**	*Rapporteur*
Monsieur	**Michel Scholl**	*Rapporteur*
Monsieur	**Philippe Rigaux**	*Examinateur*
Monsieur	**Dan Vodislav**	*Examinateur*
Monsieur	**Dominique Laurent**	*Directeur de thèse*
Mademoiselle	**Tuyêt-Trâm Dang-Ngoc**	*Co-encadrante*

Remerciements

J'adresse mes vifs remerciements à Madame Christine Collet et Monsieur Michel Scholl pour avoir accepté d'être rapporteurs de ce travail. Leurs conseils et commentaires ont été constructifs et très utiles pour améliorer le manuscrit.

Je tiens également à remercier Monsieur Philippe Rigaux et Monsieur Dan Vodislav de m'avoir fait l'honneur de participer à ce jury de thèse.

Je tiens à exprimer mes remerciements les plus sincères à Monsieur Dominique Laurent, mon directeur de thèse, et à Mademoiselle Tuyêt Trâm Dang Ngoc, mon encadrante, pour m'avoir conseillé, encouragé et soutenu tout au long de la thèse avec patience et disponibilité, et pour la confiance qu'ils m'ont accordé. Leur support et amitié ont été très émouvants durant ces années de ma thèse.

Je tiens à adresser mes remerciements à Madame Inbar Fijalkow, Directrice du laboratoire des Equipes Traitement de l'Information et Systèmes (ETIS, CNRS UMR 8051), pour m'avoir accueilli dans son laboratoire.

Je souhaite remercier vivement Monsieur Régis Le Brettevillois, Monsieur Eric Samson et Monsieur Marc Van Cappellen, pour m'avoir accueilli dans leur entreprise dans la quelle j'ai réalisé cette thèse CIFRE.

Je tiens à remercier Monsieur Hubert Naacke, qui m'a encadré pendant le stage de Master 2 et m'a recommandé de poursuivre la recherche dans cette thèse.

J'aimerais remercier également les collègues du laboratoire et à l'université, qui m'ont aidé à m'intégrer dans la vie des enseignants et chercheurs : Philippe Laroque, Tao Yuan Ren, Virginie Sans, etc.

Je souhaite remercier mes collègues à l'entreprise : Jérôme Beau, Christophe Boutard, François Huaulmé, Raphaël Lemaitre, pour leur convivialité et amitié durant le travail pour la mise en oeuvre de l'approche dans le produit industriel.

Je remercie de mon coeur mes parents, pour m'avoir permis d'être ce que je suis et m'avoir donner la liberté nécessaire à mon épanouissement. Que cette thèse témoigne de mon respect et de mon amour. Je tiens à remercier également toute la famille derrière moi qui m'ont supporté depuis ces années.

Enfin, merci, Run, mon épouse, pour la patience dont tu fais toujours preuve à mon égard, pour ton attention, pour ta présence dans les bons comme dans les mauvais jours, pour ton encouragement et ton support, et pour ton amour. Je t'aime. *Et tous les autres à qui j'adresse toute ma gratitude.*

Cergy-Pontoise, juin 2011
Tianxiao Liu

Table des matières

Table des figures

Chapitre 1

Introduction

Depuis plus de quarante années, l'accès aux données demeure toujours un sujet important dans le monde informatique. Beaucoup de travaux ont été réalisés pour faciliter l'accès aux diverses données réparties partout dans le monde, d'une manière efficace et stable. Des entreprises ou établissements ont développé les systèmes d'informations pour stocker, interroger et mettre à jour les données. Chacun de ces systèmes dispose d'une façon pour définir le format de données et surtout un mode d'accès aux données permettant aux utilisateurs d'interagir avec les données. Ce type de système d'information est appelé *source de données*. Comment répondre à une question qui nécessite d'accéder à plusieurs sources de données devient une problématique importante depuis une vingtaine d'années. En plus de la diversité des sources de données, l'étendue des sources réparties sur le réseau à différentes échelles (locale, urbaine voire mondiale) amène beaucoup de difficultés dans ce nouveau domaine, notamment en termes de performance.

L'objectif de l'intégration de données est de combiner les données résidant sur différentes sources et de fournir aux utilisateurs une vue unifiée et centralisée de ces données en masquant les détails sur l'hétérogénéité de ces sources. Il s'agit d'une problématique typique lors de la concentration ou de la fusion d'entreprises : faire cohabiter les différents systèmes tout en leur permettant d'interagir d'une manière harmonieuse. Il ne s'agit pas de reconstruire une base répartie à partir des anciennes, mais de conserver les anciennes bases et leur autonomie locale, tout en permettant l'intégration de l'ensemble. Cette solution mène à la problématique du traitement efficace des requêtes des utilisateurs sur des bases de données réparties et hétérogènes.

Dans ce chapitre, nous allons rappeler les principes d'intégration de données et les problématiques pour répondre aux requêtes des utilisateurs dans

ce contexte. Nous introduisons brièvement notre solution proposée dans cette thèse, qui sera ensuite détaillée dans les chapitres suivants.

1.1 Accès aux sources de données hétérogènes réparties

De nos jours, les utilisateurs sont toujours à la recherche des informations dont ils ont besoin sur différentes plate-formes. Ce genre de recherche est formulé par les *requêtes* traitées par les systèmes d'informations. Les informations sont souvent dispersées sur plusieurs sources, il faut donc les synthétiser afin de fournir aux utilisateurs une réponse complète à la requête posée. De plus, les utilisateurs ont potentiellement besoin d'enchaîner leurs requêtes pour avoir des résultats ayant encore plus de sens, qui les aideront à prendre des décisions stratégiques dans différents domaines. Tous ces scénarios sont difficiles pour les utilisateurs car ils doivent connaître la localisation des différentes sources de données, leur contenu (données), les méthodes d'accès à ces données, et enfin le format des résultats temporaires fournis par ces sources. Afin de soulager les utilisateurs des ces tâches longues et complexes, un système de fédération des informations est donc nécessaire.

Dans [Sheth et Larson 1990], les principes d'un système fédéré ont été définis : C'est un système qui contrôle et coordonne la manipulation d'une collection de systèmes de gestion de bases de données autonomes. Chacun des systèmes de base de données appartenant à une fédération, continue à être utilisé indépendamment par un ensemble d'applications. Un système fédéré intègre les bases de données locales en un ou plusieurs schémas fédérés et accepte les requêtes de consultation et de mise à jour.

Les objectifs des systèmes fédérés [Kim 1995] sont :

- supprimer la nécessité de faire migrer les données d'une base à une autre ;
- respecter l'autonomie des bases de données locales ; le système fédéré accède aux bases locales sans privilège particulier par rapport aux autres utilisateurs ;
- permettre une manipulation uniforme de toutes les données intégrées, à l'aide d'un langage unique ;
- masquer l'hétérogénéité des environnements locaux de chaque base de données (ex. machine, réseau) ;

- supporter les transactions contenant des opérations de lecture et de mise à jour sur l'ensemble des bases de fédération ;
- fournir tous les services d'un SGBD : définition de schéma, gestion des transactions, traitement des requêtes, interface interactive et applicative, outils d'administration ;
- offrir des performances qui approchent celles des systèmes de bases de données centralisés et homogènes.

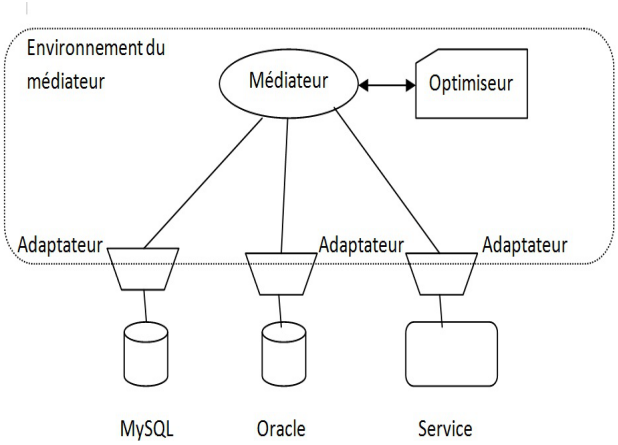

FIGURE 1.1 – Architecture d'un système de médiation

Ces systèmes tentent de proposer à l'utilisateur un accès simple et puissant à l'information. Ils sont capables d'intégrer des sources de données diverses pour offrir à l'utilisateur une vue *centralisée* et *uniforme* des données, en masquant la diversité de localisations, de formats et de modes d'accès. Ils reposent sur l'architecture médiateurs/adaptateurs (figure 1.1). Le médiateur procure à l'utilisateur ou à d'autres médiateurs (intermédiaires) une vue centralisée des données. Il y a un adaptateur par source de données, qui est chargé d'encapsuler la source de données. Il fournit pour les médiateurs un modèle de données et un mode d'accès uniformes. L'architecture médiateurs/adaptateurs, proposée par [Wiederhold 1992], est devenue une solution largement acceptée par la plupart des systèmes de fédération de données hétérogènes réparties.

L'utilisateur consulte le médiateur global en appliquant un mode classique de question/réponse. Le langage pour formuler les requêtes est unique. Le

format des résultats retournés par le médiateur doit être aussi unique. Cela facilite l'utilisation du système du côté de l'utilisateur car une connaissance sur le médiateur concernant sa localisation, la description du contenu et le langage de manipulation des données suffira. En revanche, les médiateurs ont deux tâches importantes à réaliser : d'abord, ils intègrent les informations descriptives des sources fournies par les adaptateurs. Ensuite, ils doivent traiter les requêtes de l'utilisateur. Le traitement des requêtes est réparti sur les médiateurs et les sources de données via les adaptateurs.

Le niveau d'abstraction des données et des traitements qu'apporte cette architecture médiateurs/adaptateurs présente l'avantage d'optimiser les requêtes pour améliorer la performance du système. Nous allons voir dans la section suivante, les principales problématiques de l'optimisation de requêtes dans un tel système.

1.2 Optimisation de requêtes

L'utilisation d'un système de médiation permet d'exécuter des requêtes simples et complexes. En conséquence, le traitement de ces requêtes nécessite une grande puissance de calcul du côté du système. Le médiateur global est obligé d'optimiser le processus de traitement de requêtes afin de répondre le plus rapidement possible aux utilisateurs. Les systèmes d'information pouvant traiter les requêtes contiennent toujours un module chargé de l'optimisation de requêtes, appelé *optimiseur*. Ce dernier analyse les requêtes pour déterminer un moyen efficace de les traiter. En général, il existe de nombreuses solutions pour exécuter la même requête, parmi lesquelles certaines sont souvent beaucoup plus rapides que les autres. Comment choisir la solution la plus efficace devient la problématique essentielle d'un optimiseur.

1.2.1 Informations liées au traitement de requêtes

Dans un système de médiation, une requête est formée en se basant sur les informations concernant les données stockées dans différentes sources intégrées. Nous appelons cette requête une *requête répartie*, qui est valide si les données demandées dans la requête existent dans les sources et si les opérations sur ces données demandées sont possibles. Chaque source dispose d'un certain format pour représenter la structure des données d'une manière standard. Cette représentation est appelée *schéma*. Dans ce contexte, le rôle des médiateurs est d'intégrer (d'une manière globale ou locale) ces différents

schémas issus de différentes sources de données pour pouvoir traiter les requêtes réparties.

Pour produire le résultat de la requête posée, le médiateur doit choisir une solution (*plan d'exécution*) qui précise l'ordre des opérations à effectuer. A part les informations liées à l'optimisation de la solution choisie, à la base, les informations concernant la localisation et l'accès aux sources de données sont indispensables pour construire un plan d'exécution valide.

1.2.2 Processus d'optimisation de requêtes

L'optimiseur a deux tâches principales à accomplir en un temps acceptable : d'abord, il faut trouver les solutions alternatives pour une même requête donnée. Ensuite, parmi ces alternatives, il faut choisir, en se basant sur certains critères, la plus performante pour exécuter la requête. Pour mieux comprendre les problématiques existantes dans ce contexte, nous devons nous rappeler le processus général pour traiter une requête.

Une requête est une expression décrivant les informations que l'utilisateur recherche parmi les sources de données. Cette expression est écrite en langage de requêtes défini par le système de médiation. Elle ne décrit que synthétiquement les informations recherchées, sans préciser la façon d'y accéder. Le système doit donc traduire l'expression automatiquement, via son module d'analyse de requêtes, en une séquence d'opérations élémentaires, appelée *plan d'exécution*, dont l'exécution pourra retourner les résultats envisagés. Mais la traduction n'est pas unique. En effet, plusieurs plans d'exécution peuvent être équivalents, c'est à dire donner le même résultat pour la requête. L'ensemble de ces traductions (plans d'exécution) forme un *espace de recherche*. L'optimiseur doit employer un modèle de coût pour prévoir le coût d'exécution de chaque plan d'exécution afin d'en choisir celui avec un coût minimal. Néanmoins, l'exploration de l'espace de recherche est parfois difficile pour les requêtes complexes car il existe tellement de solutions possibles que l'optimiseur doit utiliser une *stratégie de recherche* pour trouver la solution idéale en un temps acceptable, en évitant d'explorer tout l'espace de recherche exhaustivement.

1.2.3 Estimation de coût

Un *coût* quantifie l'exécution d'une requête sur des sources de données. Il peut exprimer le temps d'exécution, la charge de mémoire, le coût financier ou la charge d'énergie (coût énergétique), etc.

Le *modèle de coût* est une fonction qui accepte en paramètre un plan d'exécution et retourne son coût. Le coût du plan est calculé en cumulant le coût des opérateurs élémentaires, de proche en proche selon l'ordre défini par le plan d'exécution lui-même, jusqu'à obtenir le coût total du plan. Pour réaliser cette estimation de coût, une formule de coût est associée à chaque opérateur. La formule de coût dépend des caractéristiques de l'opérateur et aussi de son algorithme d'implémentation.

Dans un système de médiation, les sources de données hétérogènes sont normalement autonomes et ne divulguent pas les informations concernant leurs traitements des opérations. L'estimation de coût des plans pose deux nouveaux problèmes :

1. Les opérateurs dans le plan sont répartis sur les sources (adaptateurs) et les médiateurs ;
2. Les médiateurs ne connaissent pas l'implémentation des opérateurs traités sur les sources, et donc ne possèdent pas les informations de coût précises.

Cependant, de nombreuses sources de données fournissent des informations partielles de coût des opérateurs qu'elles sont capables traiter. Ces informations permettent au système d'avoir une meilleure estimation de coût pour l'optimiseur, qui pourra donc bien choisir un plan d'exécution performant.

1.2.4 Optimisation de requêtes dans un système de médiation

Pour construire l'espace de recherche, l'optimiseur utilise un certain nombre de transformations pour obtenir les alternatives d'exécution d'une requête. Les transformations sont basées sur les propriétés des opérateurs élémentaires. Dans le contexte d'un système de médiation, les transformations sont liées aux aspects de répartition et d'hétérogénéité des sources de données intégrées. L'optimiseur dispose souvent du choix entre exécuter un opérateur sur les sources, l'exécuter sur les médiateurs, ou même de partager l'exécution de l'opérateur entre les deux parties. Certaines sources fournissent une capacité fonctionnelle de traitement restreinte, qui refuse l'exécution de certains opérateurs élémentaires. Il est aussi possible que certaines sources traitent tel ou tel opérateur plus ou moins efficacement. L'optimiseur doit prendre en compte tous ces aspects afin de prévoir le coût le plus précis possible des plans d'exécution.

En plus, au sein du système, la communication des informations liées à l'optimisation (informations de coût, par exemple) doit être réalisée d'une façon performante et stable. Sur les médiateurs, les informations de coût doivent être communiquées sous un format uniforme et conçu pour l'environnement des médiateurs.

1.3 Objectif et contributions de la thèse

Dans cette section, nous présentons le contexte de cette thèse et la solution aux problèmes que nous traitons dans la thèse. Nous résumons brièvement les contributions de la solution proposée.

1.3.1 Objectif et contexte de la thèse

L'objectif de cette thèse est de définir un cadre générique d'optimisation de requêtes, qui permet d'intégrer de différentes techniques d'optimisation de requêtes pour construire efficacement des optimiseurs, dans le contexte d'un système de médiation de sources de données hétérogènes et réparties.

Cette thèse a été effectuée dans le cadre d'un contrat CIFRE. Le travail de la thèse a été mis en oeuvre dans le module d'optimiseur d'un produit d'intégration de données commercialisé par l'entreprise *XCalia - Progress Software Corporation* appelé **DVS** (*Data Virtualization Server*).

L'optimiseur de DVS nécessite une implémentation souple permettant d'intégrer facilement différentes stratégies de recherche pour différents types de requête. L'évolution de l'optimiseur pour l'intégration des éléments d'optimisation de requêtes (règles de transformation, algèbre, etc.) ne doit pas demander que toute l'implémentation soit refaite.

De plus, dans [Travers *et al.* 2006b], un processus de modélisation, optimisation et évaluation de requêtes a été proposé, afin d'évaluer les requêtes hétérogènes semi-structurées répondant aux exigences de la médiation et du traitement de requêtes. Un modèle de présentation, permettant de modéliser toutes les requêtes XQuery non-typées, appelé *Tree Graph View* (TGV) est proposé. Ce modèle TGV est utilisé comme plan d'exécution dans le processus d'évaluation de XQuery et peut être transformé pour des raisons d'optimisation. Ce contexte de travail nécessite toutefois un travail étendu sur l'optimisation de requête, qui fait une partie de l'objectif de cette thèse.

1.3.2 Contributions

Nous introduisons très brièvement ici les principales contributions de cette thèse.

Nous proposons d'abord un modèle générique (GSD) pour la description des sources de données. Ce modèle permet de décrire tous les types d'informations pour tous les types de systèmes gérant des sources. Nous avons défini les différents composants du GSD d'une manière formelle. Ces composants permettent de décrire les informations de source et de les structurer.

En se basant sur la description des schémas et les opérateurs, la description des informations supplémentaires est classifiée par les couche d'annotation, ceci facilite l'extraction des informations pertinentes. Les schémas et les opérateurs peuvent être annotés en n'importe granularité, ceci permet de décrire les informations de n'importe précision.

Avec ce modèle GSD, nous pouvons décrire des informations de coût. Nos travaux autour de l'annotation de coût sur TGV et ont fait l'objectif des articles [Travers *et al.* 2006b] [Travers *et al.* 2007a] [Travers *et al.* 2007c] [Travers *et al.* 2007b]. A partir des informations de coût, nous pouvons calculer le coût des plans d'exécution. Nos travaux sur le modèle de coût ont fait l'objectif des articles [Liu 2007] [Liu *et al.* 2007].

Nous proposons ensuite un cadre générique d'optimisation qui fournit un certain nombre de fonctions unitaires pour réaliser les procédures en commun utilisées par chaque stratégie. Les fonctions unitaires prédéfinies par le cadre permettent de manipuler les règles de transformation afin de faire évoluer le plan d'exécution d'une manière définie par la stratégie de recherche. Pour différentes stratégies de recherche, il suffit de développer l'algorithme qui pilote l'évolution du plan d'exécution en réutilisant les mêmes fonctions unitaires. Ceci permet de concevoir différents optimiseur appliquant différentes stratégies de recherche et différents critères d'optimisation.

Le cadre générique d'optimisation avec le modèle générique de description GSD ont été mis en oeuvres dans différentes versions d'optimiseur du produit DVS, en intégrant différentes stratégies de recherche. Ces optimiseurs montrent sa précision et sa performance dans le contexte de l'optimisation de requêtes hétérogènes réparties.

1.4 Organisation de la thèse

Cette thèse est organisée autour des chapitres suivants :

Chapitre 2 Après ce chapitre Introduction, nous allons dans le chapitre 2, présenter les travaux existants dans le contexte de l'optimisation de requêtes basée sur coût pour un système de médiation. Nous présentons dans un premier temps les travaux liés à la description des informations de sources utilisées pour l'optimisation de requêtes, y compris les modèles de coût hétérogènes. Dans un second temps nous parlons des technologies d'optimisation de requêtes dans un système de médiation.

Chapitre 3 Dans le chapitre 3, nous proposons un modèle générique de description de sources (GSD) qui permet de décrire n'importe quel type d'information sur n'importe quel type de source de données (ou médiateur). Ce modèle structure les informations à décrire : Les éléments de base comme les schémas et les opérateurs sont d'abord décrits, et les informations supplémentaires (notamment concernant le coût) sont ensuite associées à ces éléments de bases. Chaque élément du modèle est défini en considérant ses caractéristiques et ses relations avec les autres éléments. Ce modèle permet l'extraction des informations pertinentes demandées par l'optimisation de requêtes.

Chapitre 4 Dans le chapitre 4, notre cadre générique d'optimisation est proposé. Nous discutons de l'espace de recherche en analysant les caractéristiques des règles de transformation équivalentes. Nous présentons notre cadre générique d'optimisation avec ses fonctions unitaires. Pour montrer la généricité du cadre, l'application de certains algorithmes de stratégie de recherche dans notre cadre générique d'optimisation est montrée, allant des algorithmes naïfs comme *Exhaustif* aux algorithmes plus complexes comme *Génétique*. Nous expliquerons également comment utiliser notre cadre générique pour construire des optimiseurs ayant différents critères d'optimisation.

Chapitre 5 Le chapitre 5 montre les résultats de validation de notre approche. Nous introduisons le système de médiation utilisé pour notre validation ; ensuite, nous montrons la validation de l'estimation de coût basée sur GSD ; Enfin, les résultats de l'application des différentes stratégies de recherche dans notre cadre générique d'optimisation sont présentés. Nous présentons aussi l'intégration et mise en oeuvre de notre approche dans le

produit d'intégration de données commercialisé de l'entreprise XCalia - Progress Software Corporation.

Chapitre 6 Nous concluons cette thèse et présentons des perspectives de recherche à court et long termes dans le chapitre 6.

Chapitre 2

État de l'Art

2.1 Introduction

Dans le chapitre précédent, nous avons introduit le contexte de travail de cette thèse et la problématique principale à résoudre : Il s'agit de définir un cadre d'optimisation pour concevoir efficacement différents optimiseurs intégrant les technologies d'optimisation de requêtes dans un système de médiation. Dans ce chapitre, nous allons résumer les travaux existants dont notre travail est inspiré. Nous commençons, dans cette section d'introduction, par expliquer les termes qui sont utilisés dans cette thèse pour raison de clarification.

2.1.1 Médiation des sources de données hétérogènes réparties

La notion de système de médiation a été introduite dans [Wiederhold 1992] entre les applications et les sources de données. Les deux principaux caractères d'un système de médiation sont l'hétérogénéité et la répartition. Ces systèmes permettent d'intégrer différentes sources de données autonomes, qui ont des modèles de données très variés : modèle relationnel, modèle orienté-objet, modèle semi-structuré, modèle services Web, etc. Les sources de données intégrées peuvent être réparties sur des réseaux à plus ou moins grande échelle : du réseau local jusqu'à Internet.

Sources de données hétérogènes

Afin de comprendre l'hétérogénéité des sources de données, nous regardons tout d'abord la définition de l'homogénéité des sources dans [A. Elmagarmid et Sheth 1998] : *Un système d'information est qualifié d'homogène si le logiciel*

qui crée et manipule les données est identique pour toutes les sources, et si toutes les données ont le même format (ou modèle). Ainsi, un système est *hétérogène* s'il ne satisfait pas à toutes les conditions d'un système homogène. Par exemple, la différence de modèle de données, de langage pour créer et manipuler les données. Plus les dissimilitudes sont grandes, plus il est difficile de masquer cette hétérogénéité aux yeux de l'utilisateur.

Chaque source de données intégrée dans le système dispose d'un certain degré d'autonomie, qui indique le degré d'indépendance au niveau de traitement des données. Dans [Du et Elmagarmid 1989], trois aspects d'autonomie des sources sont décrits :

1. *autonomie de conception* : chaque source de données utilise son propre modèle de données, qui, normalement, ne peut pas être changé par l'utilisateur ;
2. *autonomie de communication* : chaque source de données ne divulgue qu'une partie des informations concernant ses données et le système gérant la manipulation de ces données
3. *autonomie d'exécution* : chaque source de données traite les requêtes selon son propre modèle de données.

Répartition (distribution) des sources

Les données peuvent être stockées sur des dispositifs locaux ou répartis géographiquement. Les données réparties ont deux possibilités en terme de relation. Premièrement, les données réparties ont un sens relationnel, ce qui nécessite en général des jointures inter-site ; et deuxièmement, les données peuvent être dupliquées de manière homogène.

Dans le suite de cette thèse, nous allons utiliser indifféremment les termes *Répartition* ou *Distribution* pour désigner la caractéristique précédemment citée des sources données par opposition aux sources de données *centralisées*.

Principes des systèmes de médiation

Comme illustré dans la figure 2.1, un système de médiation repose en général sur l'architecture médiateur-adaptateurs. Le *médiateur* est chargé de fournir aux utilisateurs une vue centralisée des données et un langage unique pour manipuler les données réparties sur différentes sources. Les *adaptateurs*, dont chacun est associé à une source de données, encapsulent la source et fournissent au médiateur un modèle de données et un mode d'accès uniformes.

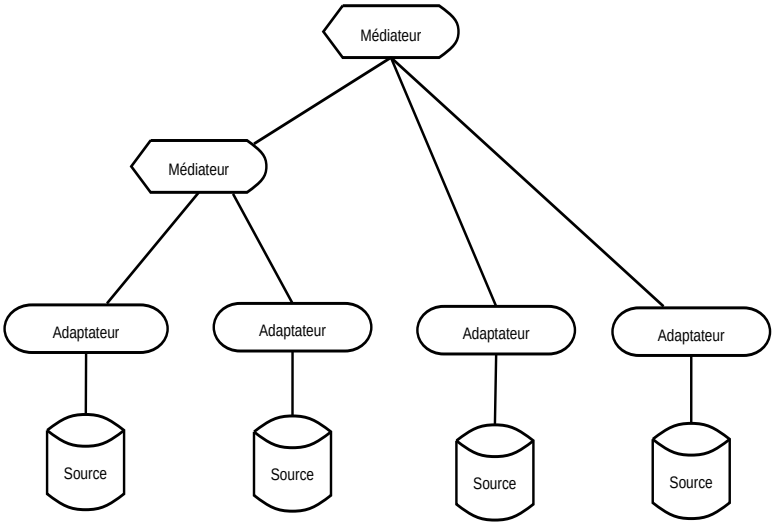

FIGURE 2.1 – Architecture d'un système de médiation

Dans cette architecture, l'hétérogénéité des sources de données est masquée par les adaptateurs et la distribution par le médiateur.

2.1.2 Optimisation de requêtes basée sur le coût dans un système homogène centralisé

La plupart des systèmes de gestion de bases de données fournissent des langages non procéduraux (par exemple, SQL) qui facilitent le développement d'applications et augmentent la productivité de l'utilisateur. Ces langages sont de nature déclarative, c'est-à-dire qu'ils permettent d'exprimer des requêtes complexes d'une manière simple et concise, en masquant les détails concernant les traitements physiques des données. L'utilisateur du langage n'est pas obligé de spécifier le processus à effectuer pour obtenir le résultat de la requête. En revanche, le système est chargé de trouver la meilleure solution pour exécuter la requête, par un module appelé *optimiseur de requêtes*. Ce module soulage l'utilisateur de l'optimisation de requêtes, tâche longue et complexe qui exploite de nombreuses informations pertinentes au sujet des données et du système. La plupart des stratégies d'optimisation peuvent être automatisées.

Dans [Florescu 1996], les principes généraux de l'optimisation de requêtes sont décrits(figure 2.2). Cette procédure se déroule pour traduire la requête initiale dans un langage de plus en plus proche du système d'exécution.

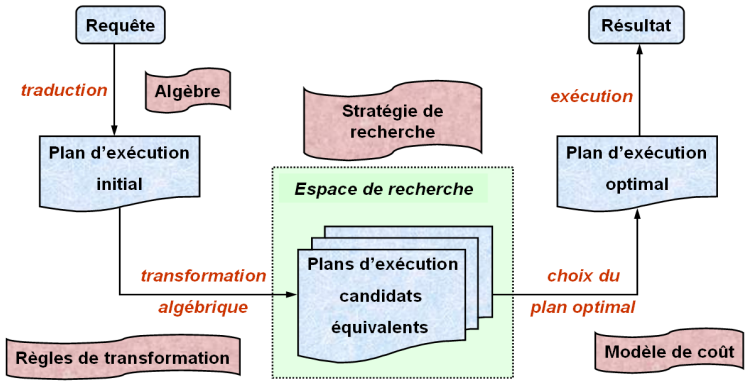

FIGURE 2.2 – Processus d'optimisation de requête basée sur le coût

Algèbre. Une algèbre utilisée par l'optimiseur est constituée d'opérateurs relationnels simples (ex. sélection, projection, jointure, union, etc.) et d'un opérateur de communication entre les sources et les autres composants du système. Les opérateurs de l'algèbre acceptent en opérande des relations composées de tuples, et produisent des relations en tant que résultat de l'opérateur. Dans les systèmes relationnels, l'algèbre a été définie par Codd [Codd 1970]. Chaque requête est traduite dans les opérateurs de l'algèbre. Il existe d'autres algèbres pour les systèmes non-relationnels, par exemple, une algèbre LORE a été proposée pour XML dans [McHugh et Widom 1999a].

Plan d'exécution. La requête peut être présentée en utilisant l'algèbre de l'optimiseur, par un arbre d'opérations dont les noeuds sont des opérateurs de l'algèbre, et les feuilles représentent les données des sources (opérandes). Cet arbre est appelé le *plan d'exécution logique* de la requête parce qu'il spécifie la méthode pour exécuter la requête, en indiquant l'ordonnancement de l'exécution des différents opérateurs, et la communication des données entre les sources et le système qui traite la requête. Si nous ajoutons les informations concernant l'exécution des opérateurs sur la couche physique du système, par exemple, l'algorithme utilisé pour effectuer les opérateurs binaires, ce plan d'exécution devient le plan d'exécution physique, qui est prêt à être exécuté par le système.

Règles de normalisation. L'optimiseur commence par arranger la requête sous une forme normalisée, plus apte à être simplifiée. Pour cela, il suit des règles de normalisation. Ces règles spécifient la manipulation des connec-

teurs logiques et des quantificateurs universels et existentiels. La forme normalisée peut être exprimée dans le langage de requête ou déjà dans l'algèbre.

Règle de transformation. Les équivalences supportées par les opérateurs induisent des réécritures algébriques conservant la sémantique de la requête mais potentiellement plus simples à évaluer. Ces équivalences sont décrites par des règles de transformations. Certaines règles ont pour but de reformuler directement la requête en un plan d'exécution. D'autres utilisent un plan d'exécution initial pour générer un ensemble de plans équivalents.

Espace de recherche L'ensemble des plans d'exécution équivalents constitue l'ensemble des possibilités d'exécution d'une requête. Le nombre de possibilités peut s'avérer rédhibitoire. Pour résoudre ce problème, l'optimiseur a besoin d'une stratégie de recherche.

Modèle de coût Le but d'un modèle de coût est d'estimer le coût d'exécution des plans. Il permet ainsi de choisir le meilleur plan d'exécution ayant le moindre coût d'exécution. Le modèle de coût contient, d'une part, des statistiques sur les données et sur le système SGBD et, d'autre part, des formules pour évaluer la taille des résultats intermédiaires et le coût des plans. Ces formules reposent en général sur un certain nombre de paramètres et d'hypothèses simplificatrices. L'unité de mesure du coût dépend de l'objectif d'optimisation. Le coût peut être mesuré :

- en *unité de temps* si l'objectif est de réduire le temps de réponse du plan ;
- en *unité de travail* si l'objectif est de réduire la consommation de ressources du système ;
- en *nombre de connexions* si l'objectif est de réduire les appels aux sources de données ;
- en *unité monétaire* si l'objectif est de réduire le prix d'exécution de la requête, dans le cas certaines accès aux sources sont coûteux ;
- en *unité de flux de données* si l'objectif est de réduire la taille de données circulées dans le réseau à un moment donné.

Stratégie de recherche La stratégie de recherche explore l'espace de recherche pour trouver le meilleur plan d'exécution en utilisant le modèle de coût. Cette stratégie définit quels sont les plans explorés parmi l'ensemble des plans de l'espace de recherche, et l'ordre dans lequel ces plans sont explorés. La programmation dynamique est la stratégie déterministe la plus couram-

ment utilisée pour construire les plans qui sont des arbres de jointures (cf. section 2.4.1).

Optimisation statique et dynamique Les systèmes de gestion de bases de données relationnelles disposent de techniques d'optimisation qui ont déjà fait leur preuve [Levy *et al.* 1996b]. La plupart des techniques d'optimisation proposées ci-dessus sont de nature statique : c'est-à-dire que le processus d'une optimisation précède l'exécution de la requête. Ces stratégies reposent sur estimation, *a priori*, des caractéristiques des sources de données. Par conséquent, si les estimations manquent de précision, le plan d'exécution choisi est loin d'être optimal. Pour contourner ce problème, des techniques d'optimisation dynamique, pendant l'exécution du plan, ont été proposées (cf. section 2.4.3).

2.1.3 Optimisation de requêtes dans les systèmes de médiation

Dans un système de médiation intégrant des sources de données hétérogènes et réparties, le problème d'optimisation de requêtes est similaire à celui rencontré dans un contexte centralisé et homogène. Cependant, il est intéressant d'exploiter les capacités respectives des différentes sources vers lesquelles les données sont distribuées, dans l'objectif de réduire le coût global des requêtes.

Estimation du coût

Dans un environnement hétérogène, les sources sont autonomes car elles ne se conforment pas à un modèle de données unique et le coût de traitement est inconnu. Les adaptateurs n'ont pas le contrôle de la source. De plus, le degré d'autonomie d'une source n'est pas le même pour toutes les sources. Les sources les plus autonomes masquent entièrement le modèle de traitement des opérateurs. A l'autre extrémité, les sources les moins autonomes fournissent toutes les informations permettant d'estimer le coût.

Dans ce contexte, la problématique de l'estimation de coût est la suivante : comment le médiateur peut-il estimer le coût d'un plan d'exécution sachant que

- les opérations du plan d'exécution sont distribuées sur les adaptateurs et le médiateur ;
- le médiateur ne connaît pas l'implémentation des opérateurs sur les sources par l'intermédiaire des adaptateurs ;

– certaines sources proposent des informations partielles de coût.

Les sources étant autonomes, elles peuvent implémenter les opérations de diverses façons. Si le médiateur adopte l'hypothèse minimale supposant que la source ne fournit aucune information de coût, il ne peut pas estimer le coût d'un plan avec précision. Cependant, de nombreuses sources offrent une description partielle du coût des opérations qu'elles sont capables de traiter. Il est avantageux pour le médiateur de disposer d'un moyen pour obtenir les éventuelles informations de coût des sources. Ces informations, bien que partielles, fournissent une meilleure estimation de coût à l'optimiseur qui, par conséquent, choisit un plan plus performant.

Adaptation de l'espace de recherche

Dans un environnement hétérogène distribué, pour traduire une requête, certaines règles de transformation sont plus particulièrement liées à l'aspect distribué et hétérogène des sources de données. Ces règles sont utilisées par l'optimiseur pour générer plus de plans d'exécution candidats, ce qui élargit l'espace de recherche.

Nouvelles transformations liées à la distribution L'optimiseur exploite l'aspect distribué du système afin de répartir le traitement de la requête entre le médiateur et les sources. En effet, certaines opérations élémentaires peuvent être traitées indifféremment par le médiateur ou par la source via l'adaptateur. Par exemple, pour filtrer une collection de données, deux solutions s'offrent à nous : soit l'adaptateur effectue le filtrage et transmet les données filtrées, soit l'adaptateur transmet la collection complète qui sera filtrée par le médiateur. Cet aspect est appelé la *localisation de traitement d'opération*. L'optimiseur modifie aussi l'ordre d'accès aux sources, il réordonne les opérations qui combinent plusieurs sources. Pour cela, il prend en compte l'associativité et la commutativité des opérations (souvent les opérations binaires : jointure, union, etc.) entre plusieurs sources de données.

Nouvelles Transformations liées à la répartition des sources Une des particularités du système de médiation est la redondance des données entre plusieurs sources. L'optimiseur exploite cette particularité pour construire plusieurs alternatives d'accès à des sources distinctes dont le contenu est identique. Les sources sont hétérogènes dans la mesure où elles n'ont pas toutes été conçues avec les mêmes objectifs. Certaines sources sont très performantes pour traiter des opérations particulières, au détriment des autres opérations.

2.1.4 Organisation du chapitre

La suite du chapitre est organisé comme suit : dans la section 2.2, nous résumons les travaux liés à la description des sources de données dans le cadre du traitement de requêtes hétérogènes réparties ; Ensuite, dans la section 2.3, nous décrivons différents modèles de coût pour les sources de données homogènes ou hétérogènes. Après, nous présentons dans la section 2.4, de différentes techniques d'optimisation de requêtes avec leurs stratégies de recherche pour explorer l'espace de recherche constituant de plans d'exécution candidats et nous introduisons les exigences d'un cadre générique d'optimisation. Nous résumons les travaux référencés dans ce chapitre dans la section 2.5.

2.2 Description des sources de données pour traitement de requêtes dans le cadre de médiation

Dans un système de sources de données hétérogènes réparties, la description des sources de données est primordiale dans le processus de traitement de requêtes. Toutes les informations liées à l'exécution et à l'optimisation de requête font partie de cette description. Dans cette section, nous résumons les travaux existants pour cette problématique dans le contexte de traitement de requêtes dans un système de médiation.

Le médiateur expose à ses utilisateurs un schéma sur les données intégrées, ce qui permet de formuler les requêtes interrogeant ces données. Ce schéma est appelé le schéma du médiateur (ou schéma global). Pour l'exécution des requêtes, le médiateur doit intégrer les schémas de chaque source de données intégrée afin de construire le schéma global.

2.2.1 Aspects globaux

Pour la description des schémas, différents systèmes de médiation se distinguent par [Hacid et Reynaud 2004] :

- La façon dont est établie la correspondance entre le schéma global et les schémas des sources de données à intégrer.
- Les langages utilisés pour modéliser le schéma global, les schémas des sources de données à intégrer et les requêtes des utilisateurs.

GAV et LAV

Pour la correspondance entre le schéma global et les schémas des sources de données à intégrer, nous distinguons deux approches : l'approche GAV (*Global As View*) qui définie le schéma global comme une vue intégrante sur les schémas locaux des sources de données à intégrer, et l'approche LAV (*Local As View*) qui est une approche descendante depuis le médiateur vers les sources de données.

Systèmes de médiation utilisant GAV

Le système de médiation Xlive [Dang-Ngoc *et al.* 2005], utilise l'approche GAV dans la quelle les adaptateurs fournissent au médiateur un accès uniforme aux sources de données hétérogènes basées sur XML .

Dans le système de médiation HERMES [V.S. Subrahmanian et Ward 1995], les adaptateurs se limitent dans leur fonctionnement à :

- assurer la correspondance du langage du médiateur et celui des sources de données ;
- proposer au médiateur un modèle de coût lors de l'exécution d'une requête basé sur les appels précédents aux sources de données
-

L'intégration des schémas des sources de données hétérogènes n'est pas prise en compte pour les adaptateurs du système médiateur HERMES qui utilise l'architecture GAV dans son objectif d'intégration.

Systèmes de médiation utilisant LAV

Plusieurs systèmes utilisent l'approche LAV :

- Information Manifold [Levy *et al.* 1996a] est un système de médiation qui essaie de généraliser les travaux dans [C. Collet et Shen 1996], [Y. Papakonstantinou et Ullman 1995] et [V.S. Subrahmanian et Ward 1995]. Il fournit un mécanisme pour décrire les schémas et la capacité fonctionnelle de traitement des sources, indépendamment du langage utilisé pour formuler les requêtes envoyées au médiateur. Cette indépendance permet au médiateur, pour une requête donnée, de ne choisir que les sources contenant des données liées à cette requête.
- Infomaster [Michael Genesereth et Duschka 1997] est un système d'intégration de données qui fournit un chemin d'accès aux sources

de données hétérogènes et distribuées sur Internet, en donnant l'impression d'utiliser un système d'information homogène et centralisé. L'idée de ce système est de déterminer une approche simple pour répondre à des requêtes. Dans son fonctionnement, Infomaster s'intéresse à la structure des sources de données et leur contenu. Il permet d'intégrer différents sources de données (principalement : les bases de données) à l'aide des adaptateurs qui assurent l'intégration : ce qui mène à un système qui fonctionne sous l'architecture LAV.

- Dans PICSEL [C. Reynaud et Gagliardi 2005], la description du contenu de chaque source de données se fait à partir d'un vocabulaire constitué de noms de relations locales. Il faut alors chercher le meilleur formalisme de représentation compte tenu de la mise en oeuvre des fonctionnalités du médiateur.

2.2.2 Autres informations liées au traitement de requêtes

Il existe d'autres informations pouvant être utiles pour le traitement de requêtes dans un système de médiation. Dans cette section, nous essayons de les résumer.

Description des informations de coût

Dans le processus du traitement de requêtes, les informations de coût sont importantes pour optimiser le plan d'exécution. Ces informations sont issues des sources de données et des médiateurs intégrant les sources. Les travaux sur la description des informations liées au coût d'exécution seront présentés dans la section 2.4.

Description de la localisation des sources

Une des informations internes que peut contenir une source de données est sa localisation. La localisation de la source de données dans un système de médiation ou dans une environnement pair-à-pair ou autres, est l'information décrivant le lieu de la source. Cette information peut être par exemple l'*URL* de la source (*Uniform Resource Locator* : un format universel pour adresser les ressources du World Wide Web) ou encore l'adresse IP (Internet Protocol : c'est le numéro qui identifie chaque ressource connectée à un réseau informatique utilisant l'internet protocol).

Il existe des systèmes de médiation qui se sont basés sur la localisation de la source pour la manipuler et l'utiliser dans le processus d'intégration. Des

systèmes comme Infomaster [Michael Genesereth et Duschka 1997] et DISCO [Tomasic *et al.* 1997] utilisent des catalogues pour classifier les ressources avec leurs localisations. Connaître la localisation ou l'adresse d'une source de données, c'est avoir un accès rapide aux informations stockées dans cette source.

Description des sources spécifiques

Un système de médiation peut intégrer les sources de données autre que les sources traditionnelles (relationnelle, objet-relationnelle, semi-structurée, service Web etc.). *Open Geosaptial Consortium* (OGC) a approuvé un standard basé sur XML pour décrire les capteurs dans un réseau : SensorML [OGC 2007].

SensorML fournit des modèles standard et un codage XML pour la description des capteurs et des processus de mesure. De sa conception, SensorML peut être utilisé pour décrire une vaste gamme de capteurs, y compris différentes plates-formes dynamiques et stationnaires, avec les capteurs intégrés à distance.

Les fonctions prises en charge dans SensorML comprennent :
- Découverte du capteur ;
- Géolocalisation du capteur ;
- Traitement des observations du capteur ;
- Un mécanisme de programmation du capteur
- Abonnement à des alertes de capteur

Il existe d'autres informations intéressantes à décrire pour le traitement de requêtes :
- Les droits d'accès aux sources de données [L. Bouganim 2004].
- Le niveau de batterie [Christian Y. A. Brenninkmijer et Paton 2009].

Mais ces travaux ont pour objectif de décrire une source de données spécifique et n'ont pas de possibilité d'être extensibles sur tous les types de sources de données.

2.2.3 Modèle et langage de description

Nous avons résumé ce que les systèmes de médiation existants peuvent décrire sur les sources de données. Dans cette partie, nous résumons les modèles/langages utilisés pour décrire les sources de données.

Description et échange des informations de sources : RDF

Resource Description Framework (RDF) [W3C 2004] est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de telles descriptions. Développé par le W3C, RDF est le langage de base du Web sémantique. L'une des syntaxes (ou sérialisations) de ce langage est RDF/XML.

Un document structuré en RDF est un ensemble de triplets. Un triplet RDF est une association :

(sujet, prédicat, objet)

- Le sujet représente la ressource à décrire ;
- Le prédicat représente un type de propriété applicable à cette ressource ;
- L'objet représente une donnée ou une autre ressource : c'est la valeur de la propriété.

Le sujet, et l'objet dans le cas où c'est une ressource, peuvent être identifiés par une URI ou être des noeuds anonymes. Le prédicat est nécessairement identifié par une URI.

Décrire une source de type Service Web : WSLD

Le WSDL [W3C 2001] décrit une interface publique d'accès à un Service Web, notamment dans le cadre d'architectures de type SOA (*Service Oriented Architecture*). C'est une description fondée sur le XML qui indique comment communiquer pour utiliser le service. Les opérations possibles et messages sont décrits de façon abstraite mais cette description renvoie à des protocoles réseaux et formats de messages concrets.

Le WSDL s'utilise pour décrire :

- le protocole de communication (SOAP RPC ou SOAP orienté message) ;
- le format de messages requis pour communiquer avec ce service ;
- les méthodes que le client peut invoquer ;
- la localisation du service.

DataGuide

Les schémas de données semi-structurées peuvent être représentés avec la structure d'arbre sous l'aide de DataGuide [Goldman et Widom 1997]. Le DataGuide est essentiellement une représentation concise de la structure de la collection qui est sous forme arborescente appelée arbre de la base de données, dans laquelle chaque chemin d'étiquette de la collection apparaît exactement une seule fois.

La figure 2.3 (b) présente le DataGuide de l'arbre de la base de données dans la figure 2.3 (a) (l'arbre de la base de données est construit en regroupant les arbres de tous les documents de la collection sous la même racine).

Le DataGuide est souvent plus petit que l'arbre de la base de données, ce qui permet de le maintenir en mémoire centrale lors de la recherche. Sans identification des noeuds de la collection, le DataGuide ne peut que renseigner sur l'existence d'un chemin d'étiquettes mais ne peut en aucun cas identifier de manière unique ce chemin dans la collection. Pour cela, un identifiant est attribué à chaque noeud du DataGuide. Cet identifiant est ensuite sauvegardé dans une table dite table des annotations (figure 2.3 (c)) sur le disque et pointe vers les identifiants des noeuds des documents de la collection qu'il représente. Le noeud #5 du DataGuide de la figure 2.3 (b) pointe vers les noeuds &5 et &8 de la base de données dans la figure 2.3 (a).

Le DataGuide et la table des annotations permettent ensemble d'indexer toute la structure de la base de données, mise à part la relation père/fils entre les noeuds de niveaux différents (par exemple, nous savons que les noeuds portant l'étiquette para sont des fils des noeuds portant l'étiquette section mais on ne peut pas déterminer avec exactitude que le noeud &5 est le fils du noeud &4. Le DataGuide ne représente que la hiérarchie entre les étiquettes, mais pas entre les noeuds eux-mêmes.).

2.2.4 Exigence d'un modèle générique de description de sources

Bien qu'il existe différents modèles ou langages pour décrire différents types de sources de données, ces modèles sont spécifiques pour le type de source qu'ils décrivent. Dans un système de médiation, nous avons besoin d'intégrer de nouveaux types de source sans changer le modèle de description, pourtant, avec les solutions existantes, nous ne pouvons atteindre cet objectif. Nous

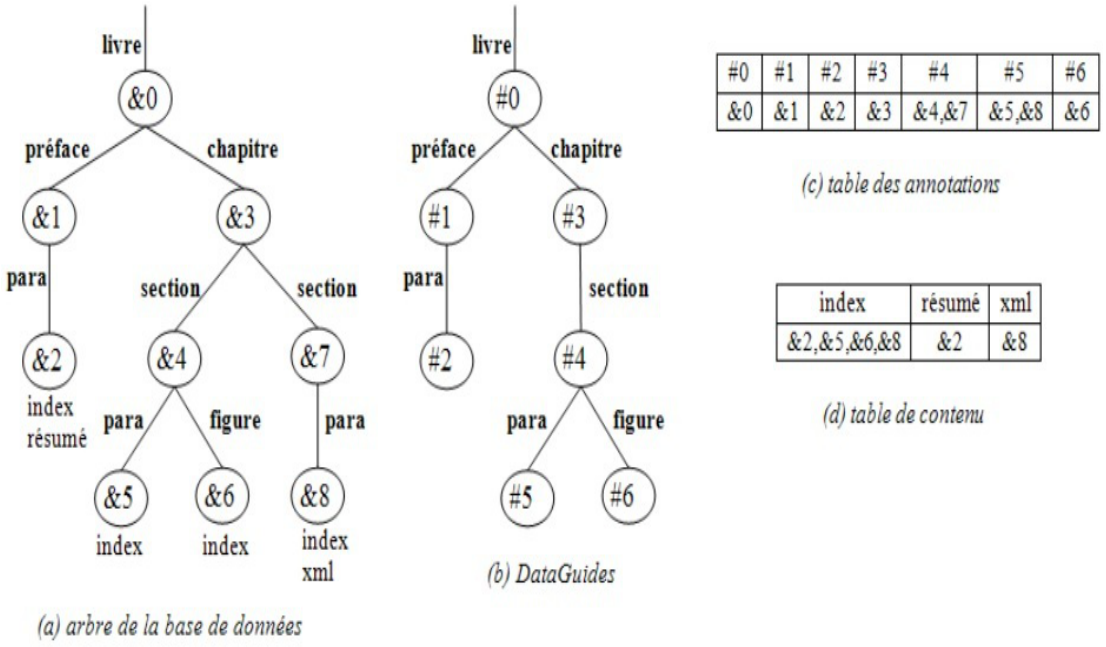

FIGURE 2.3 – Exemple de l'index avec DataGuide

avons ainsi besoin d'un modèle générique pouvant décrire n'importe quel type de source.

De plus, il est nécessaire de pouvoir ajouter des informations supplémentaires (coût, localisation, etc.) à la description des schémas. Ces informations sont nécessaires pour le traitement de requêtes dans un environnement hétérogène réparti. Nous avons ainsi besoin d'un modèle de description permettant d'intégrer tous les types d'informations possibles dans la description de sources.

2.3 Modèles de coût

La performance d'un optimiseur dépend de deux aspects essentiels, la précision du modèle de coût appliqué et la stratégie d'optimisation choisie. Le modèle de coût peut être considéré comme une fonction générique qui accepte en paramètre un plan d'exécution physique et retourne, après le calcul, le coût d'exécution total du plan. Nous évoquons dans cette section, les mo-

dèles de coût proposés dans les systèmes SGBD classiques et ceux avec des méthodes particulières utilisées par les systèmes de médiation.

2.3.1 Statistiques

Pour calculer le coût du plan d'exécution, le modèle de coût doit élaborer les formules de coût pour chaque opérateur élémentaire présent dans le plan, afin de cumuler le coût total. Dans ces formules de coût, nous pouvons avoir besoin des informations concernant les statistiques sur les données interrogées, ou sur les traitements de base. Beaucoup de travaux ont été réalisés sur les statistiques utilisées dans le modèle de coût pour l'optimisation de requêtes.

Statistiques sur les données

Dans [Bernstein *et al.* 1981] [Daniels *et al.* 1982] [Apers *et al.* 1983], les informations suivantes sont considérées importantes en tant que statistiques :

- pour chaque collection, le nombre de tuples et leur taille moyenne ;
- le nombre de pages occupées par la collection.
- le nombre de valeurs distinctes pour chacun des attributs, afin d'évaluer la sélectivité des opérations de restriction contenant des prédicats d'égalité ; les valeurs minimales et maximales des attributs de sélection pour des prédicats de comparaison ;

Les formules de coût pour estimer la cardinalité du résultat sorti de l'exécution des opérateurs (sélection, jointure, semi-jointures, union et différence) ont été résumées dans [Özsu et Valduriez 1999].

Dans [Piatetsky-Shapiro et Connell 1984], une approche pour estimer la sélectivité des prédicats a été proposée. Cette sélectivité est un paramètre fondamental pour calculer le coût des algorithmes [Ioannidis et Christodoulakis 1991]. Ces travaux sont basés sur l'hypothèse d'une distribution uniforme des valeurs des attributs et d'une indépendance entre les attributs de chaque collection. Néanmoins, il existe des travaux qui ne supposent aucune distribution particulière [Perriezo 1958], en utilisant une approche basée sur les histogrammes [Ioannidis 1993]. [Gardy et Puech 1989] a proposé une approche plus théorique en tentant de résoudre le problème de l'estimation de sélectivité des prédicats de jointures.

Statistiques sur le système

Les statistiques sur l'exécution des opérations de base du système incluent :

1. Le temps pour initialiser la lecture d'une source ;
2. Le temps unitaire pour retrouver un tuple satisfaisant un critère de sélection ;
3. Le temps unitaire pour projeter un tuple sélectionné ;

Pour les systèmes SGBD, ces informations peuvent être obtenues à l'aide de méthodes de coût spécifiques (cf. Section 2.3.3).

2.3.2 Modèles de coût dans le cadre d'une seule source

La plupart des modèles de coût pour les systèmes SGBD ne concernent que le temps d'exécution des opérateurs sur la source. Pour les SGBD relationnels, les formules pour estimer le coût des opérateurs sont proposées dans [Daniels *et al.* 1982]. Pour les SGBD orientés objet, les formules sont définies de la même façon que pour les SGBD relationnels [Blakeley *et al.* 1993] [Cluet et Delobel 1992] [Dogac *et al.* 1994]. Dans [Gardarin *et al.* 1996a], le modèle de coût d'une opération appelée *pointer chasing* fréquemment utilisée pour évaluer les requêtes orientées-objet a été proposé. Cette opération consiste à naviger à travers des identifiants d'objects. La fragmentation des données a une telle influence sur le temps de réponse des requêtes que la plupart des SGBD orientés-objet donnent à l'administrateur la possibilité de définir une stratégie de fragmentation. Les répercussions de cette stratégie au niveau du modèle de coût sont importantes et ont été prises en compte par les travaux réalisés dans [G. Gardarin et Tang. 1995].

En semi-structuré, il s'agit de considérer le facteur de sélectivité en tenant compte des chemins. Dans [Aboulnaga *et al.* 2001], deux stratégies sont présentées pour estimer la sélectivité d'un XPath. Dans [McHugh et Widom February 1999], des formules de coût pour les données semi-structurées natives ont été proposées. L'idée est de stocker les statistiques sur tous les sous-chemins possibles jusqu'à une longueur k où k est le paramètre de contrôle. Et puis, à partir de ces statistiques, le coût de tous les opérateurs de l'algèbre définie dans LORE peut être estimé en utilisant les formules. Le coût d'un opérateur LORE contient le coût d'entrées-sorties, le coût CPU et leur nombre d'évaluation. Cependant, cette approche peut surcharger la source car le volume d'informations conservées peut être énorme.

2.3.3 Méthodes de coût spécifiques pour sources hétérogènes

Dans un système de médiation intégrant les sources de données hétérogènes distribuées, l'objectif de l'optimisation de requête est de réduire le temps d'exécution et la consommation des ressources du systèmes [Hevner et Yao 1979]. Ces deux aspects ne sont pas indépendants parce que généralement si nous arrivons à réduire la consommation de ressources par l'exécution de requête, le temps de réponse sera également diminué. Il existe d'autres objectifs pour améliorer l'exécution de requête, par exemple, réduire la congestion du réseau utilisé pour relier les médiateurs et les adaptateurs, économiser l'énergie limitée d'une source de données autonome, etc. Mais la réduction du temps d'exécution demeure l'objectif principal dans la plupart de systèmes d'information. Nous allons nous concentrer sur le coût en temps d'exécution pour la suite de cette thèse.

Si la source de données intégrée dans un système de médiation divulgue les informations précises de coût : les coût entrées/sorties, le temps unitaire de traitement CPU etc, le médiateur sera capable d'estimer le coût avec les formules de coût proposées dans [Daniels *et al.* 1982]. Dans le cas contraire, il existe plusieurs solutions qui essayent de construire un modèle de coût s'appuyant sur les caractéristiques logiques des sources. Nous résumons ci-dessous ces méthodes spécifiques suivant l'ordre de connaissance des sources.

Méthode par calibration

La méthode par calibration [Du *et al.* 1992] est utilisée par le système Pegasus [Du et Shan 1995] qui intègre des bases de données relationnelles. L'approche est basée sur les hypothèses suivantes :

- le médiateur connaît les méthodes d'accès pour exécuter les requêtes sur un adaptateur ;
- le médiateur connaît la cardinalité de la table accédée et la sélectivité du prédicat P de la requête ;
- la taille du tuple est fixe ;
- il y a exactement une condition de jointure/sélection dans une relation ;
- le tuple entier est projeté en résultat ;
- tous les attributs sont des entiers.

Les coefficients du modèle représentent le coût moyen pour initialiser le traitement de la requête et les coûts CPU et d'entrées-sorties pour produire

le résultat de la requête. Une procédure de calibration est effectuée : exécuter une série de requêtes sur une base de données gérée par le même SGBD que la source. Ces requêtes spéciales et une base de données de calibration sont construites pour faciliter le déroulement de procédure de calibration. Le nombre de requêtes exécutées dépend du nombre de coefficients inconnus. D'après les résultats de l'exécution, les valeurs des coefficients peuvent être déduites en résolvant le système de n équations à n inconnues.

Vu les hypothèses prises ci-dessus, cette approche ne tient pas compte des variations dynamiques de l'environnement. Dans [Gardarin *et al.* 1996b], une extension de l'approche de [Du *et al.* 1992] est proposée pour les SGBD orientés objet.

Modèles de coût par échantillonnage de requêtes

La méthode par échantillonnage [Zhu 1995] [Zhu et Larson 1998] a été proposée pour construire un modèle de coût pour médiation de bases de données relationnelles. L'hypothèse de l'approche est que les propriétés physiques et logiques des collections accédées par la requête sont connues.

La méthode par échantillonnage contient 3 étapes :

1. Regrouper les requêtes en classes homogènes. Le coût des requêtes de chaque classe peut être estimé de façon satisfaisante avec la même formule. Les informations utiles pour classifier les requêtes sont :
   - les caractéristiques de la requête (opérateur, prédicat, etc.)
   - les caractéristiques des données (cardinalité, nombre d'attributs, type d'index, etc.)
2. Extraire un échantillon de requêtes pour chaque classe. Exécuter les requêtes issues de l'échantillonnage sur la source pour mesurer le coût.
3. Utiliser les mesures des échantillons pour déduire les formules de coût de chaque classe par une méthode de régression multiple. D'abord, un modèle de coût est construit pour chaque classe, avec ses paramètres. Puis, une équation linéaire est construite en fonction d'une partie de ces paramètres : seuls les paramètres significatifs sont choisis. Ce modèle est testé pour vérifier sa qualité. Si le résultat n'est pas satisfaisant, c'est-à-dire que nous ne pouvons pas estimer le coût de toutes les requêtes de cette classe en utilisant le même modèle, nous revenons à la première étape.

Dans [Zhu 1995], une méthode étend le modèle de coût par échantillonnage pour prendre en considération les variations de l'environnement. Deux stratégies sont proposées pour résoudre le problème de la perte de précision des formules de coût dans le temps :

- invoquer périodiquement (ou au moment de changement majeur de l'environnement) l'exécution des échantillons de requêtes et le processus pour déduire les paramètres des formules de coût.
- ajouter des variables supplémentaires dans les équations linéaires du modèle de coût pour refléter les changements de l'environnement d'exécution.

Méthode par historiques

Lorsque la source au niveau de l'adaptateur se comporte comme une boîte noire pour le médiateur, c'est-à-dire que le médiateur ne connaît aucune information de la source, il est impossible d'estimer le coût avec les méthodes précédentes. Une solution est proposée dans [Adali *et al.* 1996] pour estimer le coût en s'appuyant uniquement sur les statistiques des exécutions antérieures. C'est la solution retenue par HERMES [V.S. Subrahmanian et Ward 1995] avec son modèle de coût fondé sur les historiques de coût.

Chaque sous-requête traitée par un adaptateur est associée avec un prédicat d'appel à une source. Les performances des appels aux sources sont mesurées par le système. Les statistiques suivantes de ces appels sont enregistrées dans une base des historiques au niveau du médiateur :

- le temps pour obtenir le premier tuple du résultat ;
- le temps pour obtenir tout le résultat le temps de réponse total ;
- la cardinalité du résultat ;
- le prédicat correspondant à l'appel.

Dès qu'une nouvelle requête arrive, pour estimer son coût, nous comparons le prédicat d'appel associé à cette requête avec les prédicats stockés dans la base des historiques. Les prédicats les plus ressemblants sont retenus pour l'évaluation du coût de la nouvelle requête.

Cette approche est efficace dans le cas où le médiateur est interrogé avec des requêtes comportant des sous-requêtes similaires. Néanmoins, la base des historiques posera des problèmes de stockage, en raison des ressources limitées disposées par le médiateur, et il est difficile d'estimer le coût si les requêtes sont hétérogènes.

2.3.4 Modèle de coût générique

Nous avons résumé ci-dessus les principales solutions existantes qui proposent un modèle de coût hétérogènes. La méthode par calibration [Du *et al.* 1992] propose un modèle de coût indépendant des sources hétérogène qui tend vers l'aspect générique du coût. Le modèle de coût proposé dans le système Garlic [Haas *et al.* 1997a] distingue le coût global du plan d'exécution et le coût des sous-plans évalués par les adaptateurs. Ce modèle de coût préfère un modèle de coût spécifique pour chaque adaptateur plutôt qu'un modèle universel pour tous les adaptateurs.

Pour soulager les limitations des approches ci-dessus, le travail de [Naacke *et al.* 1998] a proposé un modèle de coût générique pour un système de médiation appelé DISCO [Tomasic *et al.* 1998] intégrant des bases de données hétérogènes. Ce modèle de coût distingue également les opérateurs exécutés sur les sources de données de ceux exécutés sur les médiateurs. DISCO intègre dans son modèle de coût générique les informations de coût extraites depuis les adaptateurs avec une hiérarchie de coût.

Langage de communication du coût dans DISCO

Dans le modèle de coût de DISCO, un langage de coût orienté objet a été proposé. Étendu avec une définition pour les statistiques et pour les formules de coût, ce langage est utilisé pour communiquer le modèle de coût depuis les adaptateurs vers le médiateur. Le modèle de coût de DISCO contient trois types d'informations :

- *statistiques*, qui décrivent le coût des paramètres de la sources ;
- *formules de coût*, qui dépendent des statistiques et décrivent le coût des opérations traitées par la source et l'adaptateur ;
- *règles de coût*, qui associent un ensemble d'opérations avec les formules décrivant le coût de cet ensemble d'opérations et qui servent de support pour exporter les formules de coût.

Pour exprimer ces trois types d'information, le langage définit les mots clefs identifiant les éléments principaux dans les informations de coût. Par exemple, pour les statistiques de données, les variables *count, totalSize, tupleSize* sont prédéfinies pour exprimer la cardinalité, la taille de la collection et la taille d'un tuple dans la collection.

Limitation Cette solution permet aux développeurs d'adaptateurs de décrire et d'exporter précisément les coût des opérations traitées par les adaptateurs d'une manière standard et formalisée. Le langage prédéfinit un certain

nombre de variables en supposant que les informations de coût ne se limitent qu'à ces éléments principaux. Néanmoins, dans un système de médiation intégrant des sources de données très hétérogènes et réparties, le modèle de coût peut concerner beaucoup d'autres types informations par exemple, le coût de consommations d'énergie dans une batterie, le coût de congestion de réseau. Ces variables prédéfinies ne permettront plus de décrire ces informations et nous avons besoin d'un langage descriptif plus générique qui permet d'exprimer n'importe quelle information de coût en gardant la cohérence entre ces informations au sein du système.

Évaluation du coût dans DISCO

DISCO propose un algorithme pour intégrer les modèles de coût dans le médiateur en classant les fonctions de coût dans une *hiérarchie*. Cette hiérarchie est au centre de l'évaluation du coût qui se déroule en deux phases. La première phase consiste à sélectionner, parmi la hiérarchie obtenue, les fonctions les plus spécifiques ; La deuxième phase est l'évaluation des formules sélectionnées.

Le modèle de coût *global* regroupe le modèle de coût générique du médiateur avec les modèles de coût des adaptateurs. Le niveau racine contient le modèle de coût générique. Les trois autres niveaux sont de plus en plus spécialisées ; ils contiennent les règles exportées, qui sont classées par niveau en fonction de leur domaine d'application. Les niveaux sont définis comme suit :

- Le *niveau générique* contient les règles applicables à tous les adaptateurs ;
- Le *niveau adaptateur* contient les règles applicables à toutes les collections d'un adaptateur donnée ;
- Le *niveau collection* contient les règles applicables à tous les prédicats d'un opérateur, pour une collection et un adaptateur donnés ;
- Le *niveau prédicat* contient les règles applicables à toutes les opérandes d'un prédicat donné pour un opérateur, une collection et un adaptateur donnés.

Limitation La classification des coûts est utilisée pour déterminer pendant l'évaluation quelle est la règle la plus spécifique qui s'applique à un opérateur du plan d'exécution. Néanmoins, la hiérarchie de coût utilisée dans DISCO distingue les opérateurs exécutés sur les adaptateurs et ceux sur le médiateur, ceci réduit la généralité du modèle parce que s'il existe une information de

coût concernant à la fois des parties médiateur et adaptateur du plan d'exécution, elle ne peut pas être classée dans la hiérarchie. De plus, la capacité fonctionnelle de traitement des sources est prise en compte dans le système DISCO mais elle n'est pas intégrée dans ce modèle de coût. La description de telle capacité n'est pas compatible avec la description des autres informations de coût.

Pour résumer, les solutions existantes sur le modèle de coût générique sont spécifiques à certains systèmes de médiation avec des limitations en terme de généralité du modèle. Nous avons besoin d'un modèle de coût générique pour tous les objectifs d'optimisation possibles et pour tous les types de système de médiation.

2.4 Optimisation de requêtes dans un système de médiation

Dans la section précédente, nous avons résumé les essentiels travaux sur le modèle de coût pour des bases de données hétérogènes. Dans un système de médiation, pour effectuer l'optimisation de requêtes basée sur le coût, en plus d'un modèle de coût précis pour évaluer le coût des plans d'exécution, nous avons également besoin d'un cadre d'optimisation qui définit la procédure pour trouver, à partir d'un plan d'exécution initial, le plan optimal pour évaluer la requête. Ce cadre d'optimisation est composé de composants qui réalisent la procédure d'optimisation : une stratégie de recherche qui pilote le déroulement d'exploration de l'espace de recherche contenant les plans d'exécution candidats (cf. section 2.1.2). Concernant la construction de l'espace de recherche, dans les environnements hétérogènes distribués, il existe des règles de transformation liées aux aspects d'hétérogénéité et de répartition des sources de données.

Dans cette section, nous résumons tout d'abord les principaux algorithmes de stratégies de recherche appliquées par les systèmes d'information dans la section 2.4.1 ; ensuite nous résumons dans la section 2.4.2 les travaux liés à l'espace de recherche dans les environnements distribués ; dans la section 2.4.3, nous introduisons les technologies concernant l'optimisation dynamique ; enfin, dans la section 2.4.4, nous analysons le besoin d'intégrer les technologies existantes dans un cadre générique d'optimisation.

2.4.1 Stratégies de recherche : algorithmes métaheuristiques

Le rôle d'une stratégie de recherche est d'exploiter l'ensemble des plans d'exécution candidats afin d'en trouver celui de coût optimum en utilisant les modèles de coût. Intuitivement, il est préférable de vérifier tous les plans d'exécution dans l'espace de recherche afin d'assurer que le plan choisi pour l'évaluation de requête est le meilleur en terme de coût d'exécution. Il s'agit donc d'un *algorithme exhaustif* qui essaie toutes les combinaisons de règles de transformation pour générer tous les plans candidats possibles pour la requête donnée. Pour chaque plan candidat, son coût d'exécution est calculé grâce au modèle de coût et le plan ayant le coût minimal est choisi pour évaluer la requête.

Cependant, pour les requêtes complexes composées d'un grand nombre d'opérations, le nombre de plan candidats peut être immense et l'application de l'algorithme exhaustif s'avère très coûteux. Le temps d'optimisation d'une requête est alors inacceptable. Pour éviter d'explorer tout l'espace de recherche, l'optimiseur utilise une stratégie de recherche issue de la métaheuristique, afin de n'explorer qu'une partie de l'espace et de trouver un plan candidat *optimal* selon le critère de la stratégie utilisée.

Les *métaheuristiques* sont des méthodes génériques qui peuvent optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme employé. L'objectif des métaheuristiques est de résoudre un problème d'optimisation donné : elle cherche un objet mathématique (une permutation, un vecteur, etc.) minimisant (ou maximisant) une fonction objectif, qui décrit la qualité d'une solution au problème. L'ensemble des solutions possibles forme l'espace de recherche. Les métaheuristiques manipulent une ou plusieurs solutions, à la recherche d'un optimum, une meilleure solution au problème. Les itérations successives doivent permettre de passer d'une solution de mauvaise qualité à une solution optimale. L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement en l'atteinte du temps d'exécution imparti ou en une précision demandée. Il faut noter que la solution trouvée par l'algorithme est souvent une solution *optimale locale* qui est plus ou mois proche de la solution optimale selon la stratégie.

Dans le domaine d'optimisation de requêtes, certains algorithmes issus de métaheuristiques sont couramment utilisés. Ce sont des algorithmes de

programmation dynamique, incrémental et recuit simulé. Nous résumons ces algorithmes comme suit :

Algorithme incrémental L'algorithme incrémental [Nahar *et al.* 1986] est souvent utilisé par l'optimisation de requêtes dans une source de données locale. L'algorithme commence par un plan d'exécution aléatoirement choisi et tente, en appliquant en chaque étape de transformation, la règle de transformation diminuant le plus le coût d'exécution de plan. L'optimisation s'arrête quand il n'existe plus de règles de transformation applicables qui peut réduire le coût d'exécution du plan.

L'avantage de cet algorithme est que le temps de calcul du plan optimal local est court mais le plan obtenu est souvent loin d'être optimal.

Algorithme recuit simulé Le recuit simulé [Kirkpatrick *et al.* 1983] s'appuie sur l'algorithme de Metropolis-Hastings, qui permet de décrire l'évolution d'un système thermodynamique. Partant d'une solution donnée, en la modifiant, on en obtient une seconde. Soit celle-ci améliore le critère que l'on cherche à optimiser, on dit alors qu'on a fait baisser l'énergie du système, soit celle-ci le dégrade. Si on accepte une solution améliorant le critère, on tend ainsi à chercher l'optimum dans le voisinage de la solution de départ. L'acceptation d'une « mauvaise » solution permet alors d'explorer une plus grande partie de l'espace de solution et tend à éviter de s'enfermer trop vite dans la recherche d'un optimum local.

En général, cette solution permet de trouver une solution meilleure que l'algorithme incrémental parce qu'elle explore une plus grande partie de l'espace de recherche. Mais la solution trouvée n'est pas forcément la solution optimale dans l'espace de recherche global.

Programmation dynamique La programmation dynamique [Selinger *et al.* 1979a] est une technique algorithmique qui permet de résoudre une catégorie particulière des problèmes d'optimisation sous contrainte. Elle a été désignée par ce terme pour la première fois dans les années 1940 par le professeur Richard Bellman. Elle s'applique à des problèmes d'optimisation dont la fonction objectif se décrit comme la somme de fonctions monotones non-décroissantes des ressources. La programmation dynamique s'appuie sur une relation entre la solution optimale du problème et celles d'un nombre fini de sous-problèmes. Concrètement, cela signifie que nous allons pouvoir déduire la solution optimale d'un problème à partir d'une solution optimale

d'un sous problème. Généralement, cette relation est utilisée pour évaluer les solutions des problèmes *de bas en haut*, c'est-à-dire que nous calculons les solutions des sous-problèmes les plus petits pour ensuite déduire petit à petit les solutions de tous les sous-problèmes. Cette solution s'avère intéressante notamment quand les sous-problèmes sont sous-problèmes d'un grand nombre de problèmes plus grands.

Par rapport aux algorithmes incrémental et recuit simulé, la programmation dynamique finit par trouver une solution plus proche de l'optimal dans l'exploration de l'espace de recherche, mais le temps de calcul peut être très long s'il y a beaucoup d'opérateurs binaires dans le plan.

Algorithme Génétique L'algorithme génétique est un algorithme d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle : croisements, mutations, sélection. Un algorithme génétique recherche le ou les extrema d'une fonction définie sur un espace de données. Le déroulement d'un algorithme génétique peut être découpé en cinq parties :

1. La création de la population initiale
2. L'évaluation des individus
3. La création de nouveaux individus
4. L'insertion des nouveaux individus dans la population
5. Itération du processus

Algorithme Colonies de fourmis L'idée originale de l'algorithme provient de l'observation de l'exploitation des ressources alimentaires chez les fourmis. En effet, celles-ci, bien qu'ayant individuellement des capacités cognitives limitées, sont capables collectivement de trouver le chemin le plus court entre une source de nourriture et leur nid. Un modèle expliquant ce comportement est le suivant :

1. une fourmi (appelée " éclaireuse ") parcourt plus ou moins au hasard l'environnement autour de la colonie ;
2. si celle-ci découvre une source de nourriture, elle rentre plus ou moins directement au nid, en laissant sur son chemin une piste de phéromones ;
3. ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste ;
4. en revenant au nid, ces mêmes fourmis vont renforcer la piste ;

5. si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste ;
6. la piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive ;
7. la longue piste, elle, finira par disparaître, les phéromones étant volatiles ;
8. à terme, l'ensemble des fourmis a donc déterminé et " choisi " la piste la plus courte.

Chaque stratégie de recherche est appropriée à certains types de requêtes ou certains objectifs d'optimisation. Pour un cadre d'optimisation, il est préférable que ce cadre puisse intégrer de différentes stratégies de recherche sans modifier ses principaux composants comme le calcul de coût, la gestion de règles de transformation etc.

2.4.2 Travaux sur l'espace de recherche pour sources de données réparties

Parallélisation des opérateurs

Dans un SGBD hétérogène, plusieurs sources peuvent effectuer des traitements en parallèle au moyen d'une jointure inter-site. Deux algorithmes de jointure inter-site : la jointure par hachage et la jointure par fusion, exploitent le parallélisme car ils accèdent une seule fois, en parallèle, aux collections à joindre. A l'inverse, la jointure par boucles imbriquées est plus lente car pour chaque tuple provenant de la première source, une connexion doit être établie avec la deuxième source. Le coût de ces nombreuses connexions devient le coût prédominant de la jointure [Du *et al.* 1994]. En effet, dans l'architecture médiateur/adaptateurs, le médiateur utilise un langage déclaratif (ex. SQL) pour envoyer les sous-requêtes aux adaptateurs. Or, ce langage ne permet pas de décrire précisément la manière dont les opérations sont traitées. De ce fait, le médiateur ne peut pas bénéficier des méthodes d'accès et de structures de données internes des adaptateurs. Le seul cas mineur où la jointure inter-site par boucles imbriquées est plus rapide que les autres algorithmes survient lorsqu'une des deux collections a une petite cardinalité.

Ce changement de l'espace de recherche a un impact indirect sur la stratégie de recherche employée par le médiateur, notamment pour la construction des arbres de jointure inter-site. Cela s'explique comme suit. Les jointures par

hachage et par fusion sont très rapides lorsque l'arbre est réparti entre la gauche et la droite (arbre appelé *bush*), car elles exploitent le parallélisme entre les sources. En revanche, ces jointures sont très lentes pour les arbres gauches (saturation du médiateur qui reçoit en parallèle les données de toutes les sources) ou les arbres droits (aucun parallélisme). Or, la stratégie de programmation dynamique fréquemment utilisée pour construire les arbres de jointure, ne produit que des arbres gauches parce qu'il ajoute l'un après l'autre les opérateurs binaires sur le plan d'exécution d'une manière linéaire gauche. En conséquence, de nouvelles stratégies de recherche s'imposent.

Capacités réduites des sources

Les systèmes hétérogènes qui intègrent des sources très diverses doivent tenir compte des capacités restreintes de certaines d'entre elles pour traiter des requêtes. Ainsi l'espace de recherche exploré par le médiateur est réduit, il contient seulement les plans composés de sous-requêtes acceptables par les sources.

Les capacités limitées d'une source sont prises en compte dans les différents systèmes de médiation existants. La solution de Garlic [Papakonstantinou *et al.* 1993] repose sur un langage de description de requêtes acceptées par la source [Vassalos et Papakonstantinou 1997]. Ce langage permet de définir des modèles de requêtes contenant des variables et de spécifier le domaine de ces variables. Ce langage est suffisamment puissant pour décrire un ensemble infini de requêtes de taille quelconque sur un schéma quelconque. Une méthode de normalisation du modèle des requêtes est également définie pour faciliter l'optimisation des requêtes.

Dans Information Manifold [Levy *et al.* 1996a], les capacités de traitement sont décrites pour chaque vue qu'un adaptateur exporte. Ainsi, il est possible de spécifier le type d'opérations de sélections acceptées sur la vue, ainsi que les attributs devant obligatoirement être instanciés avant d'utiliser la vue.

Le premier objectif imputé à la stratégie de recherche est de générer des plans valides pour les sources de faible capacité. Dans Garlic [Haas *et al.* 1997b] [Roth et Schwarz 1997], les adaptateurs sont impliqués dans la construction du plan d'exécution, durant la phase d'optimisation qui précède l'exécution. Un plan est construit de manière ascendante en appliquant des règles de production. Ces règles transforment une description de la requête en un plan d'exécution. Des paramètres sont associés à chaque opérateur du plan :

nom des sources et des collections accédées, prédicats appliqués, deux paramètres booléens pour spécifier la matérialisation et le tri du résultat. Pendant l'optimisation, le médiateur de Garlic propose aux adaptateurs la liste des traitements à effectuer. L'adaptateur renvoie un ou plusieurs sous-plans d'exécution qu'il est capable de traiter mais qui ne couvrent éventuellement qu'une portion de la liste des traitements proposée par le médiateur. Les traitements non couverts par les adaptateurs sont pris en charge par le médiateur [Roth et Schwarz 1997].

Dans Garlic et Information Manifold, l'optimiseur utilise une méthode heuristique visant à déléguer au maximum le traitement auprès des sources pour soulager le médiateur. L'optimisation dans Garlic se déroule en trois étapes successives [Papakonstantinou 1997] :

1. L'identification des sous-requêtes maximales acceptées par les adaptateurs et susceptibles de traiter la requête. Les sous-requêtes contiennent le plus grand nombre possible de sélections et de jointures. L'algorithme est similaire à celui proposé dans [Levy *et al.* 1996b].
2. La construction du plan en combinant les sous-requêtes. L'algorithme proposé vérifie trois conditions suffisantes pour construire un plan :
   (a) tous les prédicats de la requête sont *consommés*,
   (b) les jointures entre les sous-requêtes sont correctes,
   (c) toutes les sous-requêtes sont instanciées correctement.

   Cet algorithme diffère de ceux proposés dans [Levy *et al.* 1995] [Ramakrishanan 1998] car les conditions nécessaires ne sont pas vérifiées.
3. Le raffinement du plan en poussant les projections vers les adaptateurs, autant que possible, et en éliminant les plans non optimaux au sens algébrique du terme.

Cependant, cette méthode heuristique ne permet pas toujours de trouver le plan le plus rapide. En effet, il existe des jointures, entre deux collections d'une même source, qui sont traitées plus rapidement par le médiateur que par la source [Roth *et al.* 1999].

2.4.3 Optimisation dynamique de requêtes

A la différence de l'optimisation statique qui précède l'exécution de la requête, l'optimisation dynamique a lieu pendant l'exécution de la requête. Les deux phases d'optimisation et d'exécution ne sont pas séparées séquentiellement : elles sont entrelacées dans un processus composé alternativement d'exécution et d'optimisation.

Dans les systèmes hétérogènes, il est difficile de prévoir avec précision le temps de réponse des requêtes traitées par le sources. Plus particulièrement, lorsque la congestion du réseau est forte, le transfert des résultats entre une source et le médiateur peut s'interrompre temporairement. L'objectif est de réduire l'impact des sources les plus lentes sur le temps de réponse total de la requête, dans l'optique d'améliorer globalement l'exécution de la requête.

Les technique de *query scrambling* [Amsaleg *et al.* 1996] [Urhan *et al.* 1998] proposent de contourner les situations de blocage en modifiant dynamiquement le plan d'exécution pour permettre au médiateur d'effectuer des traitements annexes en attendant la fin du blocage. Les traitements annexes représentent :

1. la matérialisation des résultats intermédiaires, et
2. la synthèse de nouveaux opérateurs pour manipuler les résultats intermédiaires disponibles.

Lorsque tous les traitements annexes sont réalisés, le médiateur attend que les sources bloquées reprennent la production des résultats.

Prise en compte des sources indisponibles

Du fait de l'autonomie des sources, il arrive fréquemment qu'une partie des sources soient indisponibles au moment où l'utilisateur pose une requête. Le problème est de traiter la requête en tenant compte des sources indisponibles sans attendre que toutes les sources soient disponibles pour débuter le traitement de la requête.

Pour résoudre ce problème, une approche est proposé, suivant deux directions [Bonnet et Tomasic 1998b] [Bonnet et Tomasic 1998c] :

1. le traitement incrémental des requêtes vise à obtenir efficacement le résultat d'une requête en commençant le traitement avant que toutes les sources soient disponibles. Dans ce cas, la requête est évaluée partiellement et les résultats partiels sont matérialisés dans le médiateur, en tenant compte d'un espace de stockage limité. La requête initiale est reformulée de façon à utiliser les résultats partiels ; elle est appelée requête incrémentale. Une fois que les sources indisponibles deviennent accessibles, la requête incrémentale est exécutée ; elle produit un résultat équivalent à celui de la requête initiale, cependant plus rapide. Un des avantages de cette méthode est de pouvoir traiter une requête bien qu'à aucun instant l'ensemble des sources ne soient disponibles simultanément.

2. l'extraction d'informations partielles a pour but de dévoiler des informations pertinentes à celui qui pose la requête, avant même qu'il obtienne la réponse complète. Les informations partielles sont obtenues à partir des résultats partiels qui sont matérialisés pendant le traitement incrémental de la requête. Les résultats partiels sont combinés de manière à obtenir un résultat approchant au maximum de la réponse complète. Dans l'hypothèse où l'utilisateur pose simultanément plusieurs requêtes, les sous-requêtes communes sont isolées pour être matérialisées en priorité. L'objectif n'est pas de réduire le temps de réponse total des requêtes, mais de maximiser la réutilisation des résultats matérialisés [Bonnet 1999] [Bonnet et Tomasic 1998a].

Ordonnancement dynamique des jointures

L'optimisation dynamique offre de nouvelles possibilités d'optimisation pour déterminer l'enchaînement des opérations à traiter afin de répondre à une requête. Une piste intéressante concerne la construction d'un arbre de jointures entre plusieurs sources.

Par définition, l'espace de recherche est immense car il contient tous les arbres qui réalisent une jointure entre toutes les sources. L'idée est de réduire fortement l'espace de recherche en imposant un ordre partiel d'accès aux sources. Or, cet ordre partiel est connu dynamiquement (*i.e.* pendant l'exécution) en supposant que les sources terminent leurs traitement à des dates différentes.

La stratégie d'optimisation du système MIND [Ozcan *et al.* 1996] [Ozcan *et al.* 1997] concerne l'ordonnancement des jointures inter-site. L'idée est ici de traiter une jointure inter-site le plus tôt possible, dès que deux sous-requêtes ont produit leur résultat. Cette approche a deux avantages : elle remplace la stratégie d'optimisation statique qui détermine un plan d'exécution à partir d'un modèle de coût, et elle prend en considération le temps de réponse réel des sources.

La méthode se déroule en quatre étapes [Evrendilek *et al.* 1997] :

1. En préliminaire à l'exécution, le coût des jointures entre deux sites est estimé au moyen des formules traditionnelles basées sur la sélectivité des jointures.
2. Toutes les sous-requêtes sont envoyées en parallèle aux adaptateurs, pour être exécutées.

3. Dès qu'une sous-requête a produit son résultat, une valeur de seuil est calculée en fonction du coût des jointures qui sont reliées à cette sous-requête.
4. Parmi toutes les jointures impliquant les résultats des sous-requêtes, celles qui ont un coût supérieur au seuil sont éliminées ; la jointure de moindre coût est exécutée. On reprend alors l'étape 3 jusqu'à ce que toutes les jointures soient exécutées.

La validation de cette approche [Evrendilek *et al.* 1997] montre une amélioration du temps de réponse total de la requête, par rapport à la stratégie d'équilibrage des jointures [Du *et al.* 1995].

Optimisation adaptative

L'approche proposée par [Bodorik et Riordon 1988] [Bodorik et Riordon 1991] [Bodorik *et al.* 1992] suppose que l'exécution d'une requête se déroule selon trois phases :

1. la phase de contrôle (*monitoring*) pendant laquelle l'état d'avancement de l'exécution est mesuré ;
2. la phase de prise de décision au cours de laquelle l'optimiseur décide éventuellement de corriger le plan courant s'il n'est plus optimal à cause de la perte de précision des estimations utilisées pour son choix ;
3. une phase de correction pendant laquelle le plan courant est abandonné au profit d'un nouveau plan. Des méthodes sont suggérées pour chacune de ces phases.

Plus précisément, trois méthodes sont présentées pour décider s'il faut corriger un plan, pendant la phase de prise de décision :

1. la *méthode de reformulation* consiste à reformuler la portion du plan qui n'est pas encore traitée en utilisant des informations de coût mises à jour. Si le coût du plan obtenu est plus faible que celui du plan courant, ce dernier est remplacé par le plan reformulé.
2. la *méthode de seuil* dans laquelle la reformulation intervient seulement lorsque le coût des résultats intermédiaires dépasse un certain seuil ou sort d'un intervalle spécifié.
3. la *méthode de la moindre durée globale* dans laquelle des plans alternatifs sont préparés en tache de fond à l'aide de règles heuristiques. Pour estimer avec précision le coût de ces plans alternatifs, la taille du résultat des jointures est déduite par échantillonnage. Le plan courant

est remplacé seulement si le coût du nouveau plan est plus faible que le temps restant au plan courant pour s'accomplir (*i. e.* le nouveau plan étant censé se terminer avant le plan en cours).

2.4.4 Exigence d'intégration des solutions dans un cadre générique d'optimisation

Dans cette section, nous avons résumé les travaux liés à l'optimisation de requêtes dans les systèmes de médiation intégrant des sources de données hétérogènes et réparties. Un résumé des travaux liés à l'optimisation de requêtes dans les systèmes de médiation peut être trouvé dans [Ouzzani et Bouguettaya 2004] et [Haas *et al.* 2002]. Ces technologies sont proposées pour résoudre les problématiques d'optimisation de requêtes dans les cas particuliers : types de requêtes spécifiques et autonomies de sources de données.

Optimisation de différents langages de requêtes du médiateur Pour un système de médiation, le langage de requête utilisé par le médiateur est unique. Ceci facilite l'utilisateur qui a une vue uniforme et centralisée du système. Pourtant, le langage choisi peut être varié pour différents systèmes de médiation. Pour implémenter un optimiseur qui ne dépend pas du langage de requête du médiateur, le cadre d'optimisation doit être générique et ne pas être spécifique à un certain format ou langage. Les système de médiation existants sont tous spécifiques pour un certain langage de requête pivot. Par exemple, Garlic [Papakonstantinou *et al.* 1993] utilise un langage relationnel et DISCO [Tomasic *et al.* 1998] utilise un langage objet-relationnel.

Modèle de coût générique L'optimisation de requêtes basée sur le coût nécessite un modèle de coût précis afin d'avoir une performance d'optimisation satisfaite. Les sources de données sont hétérogènes et autonomes, les informations de coût sont partielles et de différents formats. Nous avons besoin d'un modèle de générique au niveau du médiateur qui intègre les informations de coût hétérogènes, y compris celles sur les opérations exécutées sur le médiateur. Comme nous avons montré dans la section 2.3.4, les modèles de coût existants dans la littérature ne sont pas suffisamment génériques pour traiter la problématique de l'optimisation de requêtes dans tous les systèmes de médiation.

Intégration des stratégies de recherche Les stratégies de recherche pilotent la procédure d'optimisation de requêtes pour indiquer le début et la

fin du déroulement d'exploration dans l'espace de recherche. Un cadre générique d'optimisation ne doit pas être dépendant d'une stratégie de recherche spécifique et doit disposer des composants unitaires qui réalisent les différentes étapes d'optimisation et qui sont facilement contrôlés et utilisés par l'implémentation d'une stratégie de recherche.

Gestion des règles de transformation Dans les environnements hétérogènes et réparties, les règles de transformation pour un plan d'exécution doit prendre en compte les aspects liés à l'exécution de requêtes sur différentes sources de données. Chaque règle de transformation modifie une partie du plan d'exécution et peut avoir une grande influence sur le coût d'exécution du plan. Pour que la stratégie de recherche se déroule d'une manière efficace, le cadre d'optimisation doit classifier les règles de transformation disponibles pour un plan d'exécution en sachant l'effet que peut avoir l'application de la règle sur le plan.

Pour résumer, il est intéressant d'avoir un cadre générique d'optimisation pour les systèmes de médiation de données hétérogènes et réparties, qui ne dépende pas du langage pivot du médiateur et qui puisse intégrer les technologies d'optimisation de requêtes dans un environnement hétérogène et réparti.

À notre connaissance, il n'existe pas à ce jour de travaux portant sur un cadre générique d'optimisation pour les systèmes de médiation de données hétérogènes et réparties.

2.5 Conclusion

Dans ce chapitre, nous avons résumé les travaux existants liés au contexte de l'optimisation de requêtes basée sur le coût dans un système de médiation. La figure 2.4 illustre les différentes technologies qui doivent prises en compte dans un cadre d'optimisation de requêtes. Nous pouvons distinguer trois aspects essentiels pour l'optimisation de requêtes dans les systèmes de médiation :

- différentes stratégies de recherche doit être applicables sur le même cadre d'optimisation ;
- il faut avoir un modèle de description de sources sur lequel les informations liées à au traitement de requêtes sont décrites ;
- les modèles de coût utilisés doivent refléter les caractéristiques de l'environnement de l'exécution des requêtes afin d'avoir une estimation de coût la plus précise possible.

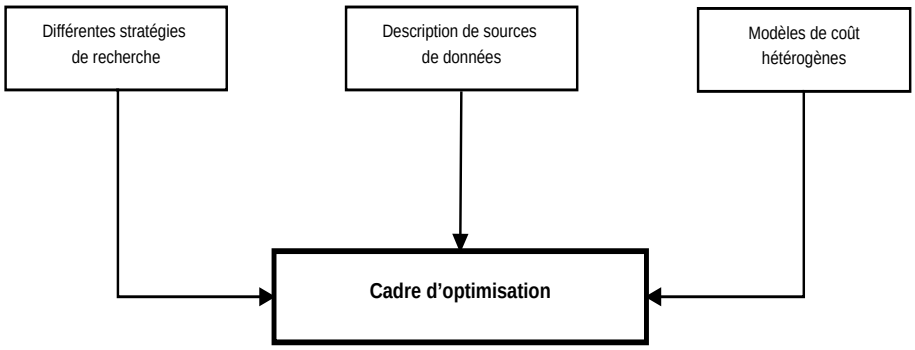

FIGURE 2.4 – Technologies intégrées dans un cadre d'optimisation

Nous synthétisons les différents modèles de coût dans la figure 2.5. Les modèles de coût sont classés par type de systèmes : sources de données relationnelles, objet (objet-relationnelle) et semi-structurées. Ils sont également classés par la précision des informations de coût. Les modèles de coût dans le cadre « Modèle de coût pour sources connues » contiennent les formules de coût précises pour calculer le coût des opérateurs présents dans le plan d'exécution. En général, les statistiques sont nécessaires pour de telles formules. Quand les sources de données sont autonomes, les formules de coût précises ainsi que les statistiques ne seront plus disponibles. Les méthodes spécifiques comme la calibration sont proposées pour avoir une estimation de coût au niveau de médiateur. De plus, cette figure indique les relations entre différents travaux de recherche en modèles de coût pour l'optimisation de requêtes.

La description du modèle de coût générique proposé dans le système DISCO permet d'améliorer l'optimisation de requêtes dans un système de médiation. Cependant, cette solution a ses limitations :

- le langage utilisé pour communiquer les informations de coût n'est pas générique parce qu'il prédéfinit les variables en supposant l'objectif de l'optimisation (pour réduire le temps de réponse) et le coût des sources de données (relationnel ou objet-relationnel).
- le processus d'évaluation distingue le modèle de coût du médiateur et ceux des adaptateurs englobant des sources de données. Il manque un modèle plus centralisé et géré par le médiateur.
- pour un système de médiation, l'optimisation de requêtes peut être envisagée pour des requêtes exprimées dans différents langages. Il faut définir un modèle de coût générique basé sur un modèle d'exécution plus générique.

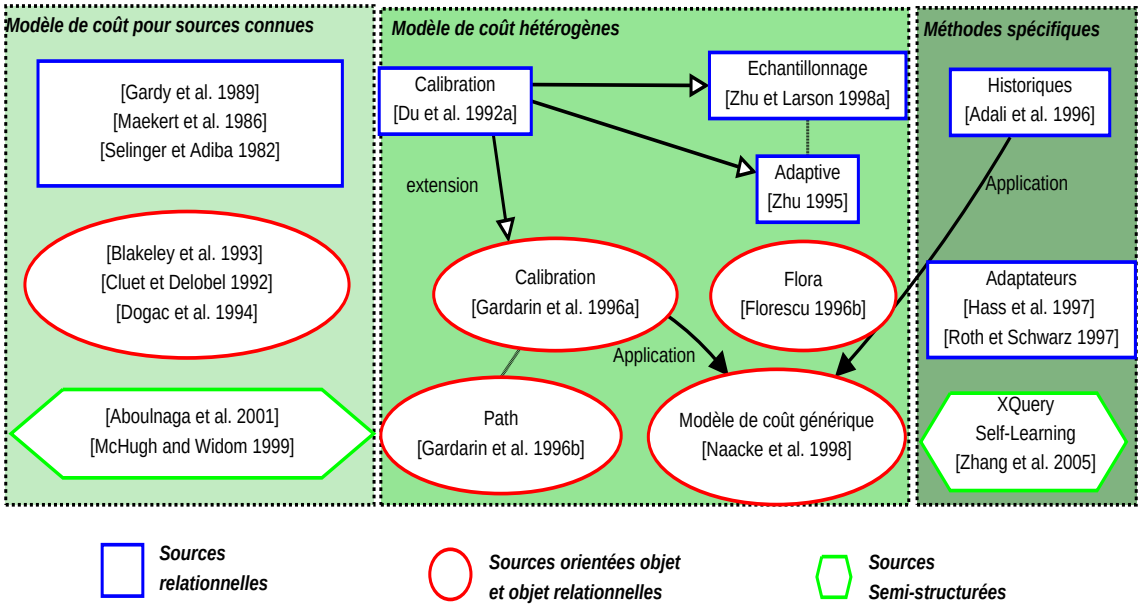

FIGURE 2.5 – Évolution des modèles de coût

Les solutions existantes présentées dans ce chapitre nous inspirent de proposer un cadre générique d'optimisation de requêtes, pour éviter la réimplémentation complète de l'optimiseur pour différents systèmes de médiation, et pour avoir une bonne performance d'optimisation. Ce cadre générique d'optimisation doit couvrir les aspects suivants :

- Il doit se baser sur un modèle de coût générique.
- Dans la mesure où la stratégie de recherche appliquée varie selon le système, et où les règles de transformation dépendent fortement de l'environnement d'exécution des requêtes, le cadre doit être capable d'intégrer les différentes stratégies de recherche.
- Le cadre d'optimisation doit intégrer les technologies liées à l'optimisation de requêtes dans les environnements hétérogènes et répartis. Ces technologies incluent la normalisation des règles de transformation, la prise en compte des capacités fonctionnelles réduites des sources, etc.

Nous détaillons la solution que nous proposons et qui répond à ces exigences dans les chapitres suivants de cette thèse.

Chapitre 3

Description Générique des Sources

3.1 Introduction

Dans un système d'information intégrant des sources de données hétérogènes réparties, afin d'assurer le bon déroulement du traitement de requêtes envoyées au système, nous avons besoin de diverses informations sur les données hébergées dans les sources, ainsi que sur le système lui-même traitant les requêtes. Nous supposons que les sources de données divulguent partiellement ou complètement les informations susceptibles de permettre le traitement efficace des requêtes, comme par exemple, les schémas, la localisation et les opérateurs disponibles, etc.

Pour illustrer les différents aspects concernant la description des sources, nous revoyons brièvement le processus de traitement de requêtes que nous avons référencé dans le chapitre 2, en nous focalisant sur les informations demandées par chaque étape de ce processus. Nous prenons un exemple de traitement de requête dans un système de médiation intégrant plusieurs sources hétérogènes et réparties.

Supposons que nous avons les données concernant **la gestion des employés dans les entreprises**. Ces données sont stockées dans des sources de données hétérogènes :

- Les données concernant les entreprises sont stockées dans une source de données objet-relationnelle.
- Les données concernant les départements des entreprises sont stockées dans une base de données semi-structurée .

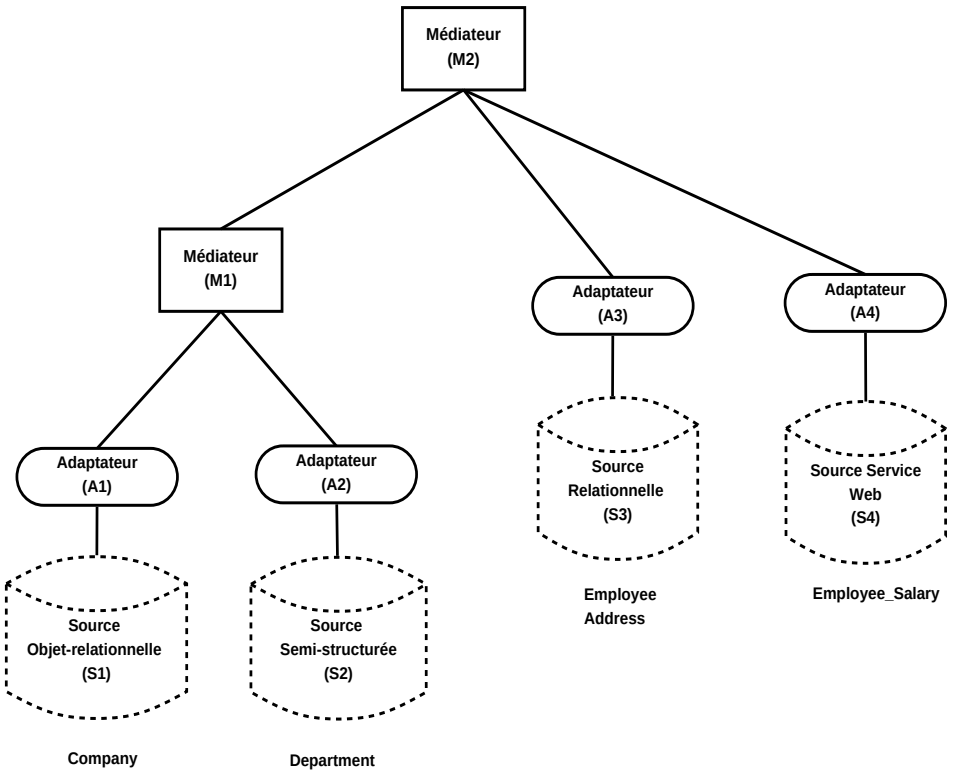

FIGURE 3.1 – Exemple d'un système de médiation intégrant des sources hétérogènes réparties

- Les données concernant les employés et leurs adresses sont stockées dans une base de données relationnelle.
- Les données concernant le salaire des employées sont disponibles via une source de type *Service Web*.

Nous avons ensuite un système de médiation intégrant ces différentes sources. Comme ce qui est illustré dans la figure 3.1, le médiateur M1 intègre la source objet-relationnelle S1 et la source semi-structurée S2 via deux adaptateurs A1 et A2. Le médiateur M2 intègre le médiateur M1, ainsi que la source relationnelle S3 et la source Service Web S4 via les adaptateurs A3 et A4. Ce médiateur M2 a donc une vue globale sur toutes les sources intégrées dans le système.

Formuler des requêtes basées sur les schémas Pour formuler une requête *valide* (pouvant être traitée par le système d'information), à part le langage défini, nous devons connaître les schémas des sources de données.

Ces sources de données étant hétérogènes ont différents formats de représentation de leurs schémas. Pourtant, pour traiter une requête interrogeant ces sources, il est important d'avoir un modèle générique utilisé pour la description de tous les types de schémas.

Pour l'exemple dans la figure 3.1, l'adaptateur de chaque source de données doit décrire les schémas de la source qu'il encapsule d'une façon uniforme, pour que chaque médiateur puisse comprendre et fusionner tous les schémas issus de toutes les sources qu'il intègre. Nous supposons que le médiateur M2 possède tous les schémas intégrés (*Company, Department, Employee, Address et Salary*), les requêtes interrogeant les données décrites par ces schémas peuvent donc formulées et envoyées aux médiateur M2.

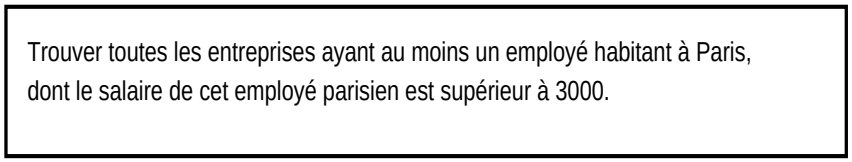

FIGURE 3.2 – Exemple d'une requête répartie

Par exemple, nous pouvons avoir une requête illustrée dans la figure 3.2. Cette requête nécessite d'accéder à toutes les données sur toutes les sources.

Construction des plans d'exécution en fonction des capacités des sources et des médiateurs Quand une requête est validée par l'analyse syntaxique, le système construit un plan d'exécution pour évaluer cette requête. Si nous sommes dans un environnement homogène centralisé, il suffit de traduire le plan d'exécution entier en une requête écrite en langage accepté par la source qui exécute cette requête. Mais s'il s'agit d'un système d'information réparti et hétérogène, nous devons connaître la localisation des données référencées dans le plan d'exécution et la capacité fonctionnelle de traitement de ces sources. Ensuite, le système formule des sous-plans qui seront envoyés sur les sources correspondantes, et exécute sur les médiateurs les opérateurs *inter-sites* ou ceux qui ne peuvent pas être exécutés par une source.

Pour l'exécution de la requête dans la figure 3.2, nous avons besoin de relier les données stockées dans différentes sources. Ainsi, les jointures inter-sites (*Company-Deparmtent, Department-Employee*, etc.) sont nécessaires.

Nous pouvons probablement demander l'exécution de la jointure *Employee-Address* sur la source possédant les données correspondantes si elle en est capable.

Optimisation des plans d'exécution basée sur les informations de coût Pour évaluer la requête d'une manière efficace, il est intéressant de procéder à une optimisation qui modifie le plan d'exécution initial et en obtient des nouveaux ayant des coûts d'exécution plus faibles, afin de choisir ensuite un plan ayant un coût optimal. Pour effectuer cette procédure d'optimisation, nous avons besoin de décrire les informations liées au coût d'exécution des opérateurs. Les formules de coût, ainsi que les statistiques concernant les données et le système, doivent être décrites d'une façon générique pour tous les types de sources de données.

Par exemple, pour la requête dans la figure 3.2, nous avons l'intérêt de comparer le coût de la jointure Employee-Address entre son exécution sur la source S3 ou sur le médiateur. Si les deux solutions sont possibles et si les informations de coût permettent de calculer le coût précis de chaque solution, nous pouvons ensuite choir la solution avec le coût optimal.

D'autres informations nécessaires Il est possible que le système possède d'autres informations liées au traitement de requêtes. Par exemple, le droit d'accès pour certaines sources de données, l'autonomie de la batterie, etc. Ces informations doivent être décrites de la même façon générique.

Dans ce chapitre, nous proposons un modèle générique de description de sources qui permet de décrire n'importe quel type d'information sur n'importe quel type de source de données (ou médiateur). Ce modèle hiérarchise les informations à décrire : Les éléments de base comme les schémas et les opérateurs sont d'abord décrits, et les informations supplémentaires sont ensuite associées à ces éléments de bases. Chaque élément du modèle est défini en considérant ses caractéristiques et ses relations avec les autres éléments. Ce modèle permet l'extraction des informations pertinentes demandées par le traitement de requêtes.

Organisation du chapitre Le chapitre est organisé comme suit : Dans la section 3.2, nous discutons des informations nécessaires à décrire pour une source de données dans le cadre du traitement de requêtes et nous introduisons la structure des informations décrites. Dans la section 3.3, nous formalisons notre modèle de description de source appelé GSD (*Generic Source*

Description). Dans la section 3.4, nous montrons plus en détail comment le GSD est appliqué pour décrire les informations de coût et comment le coût d'un plan d'exécution peut être calculé. Et nous concluons le chapitre dans la section 3.5.

3.2 Description des sources de données

Dans cette section, pour illustrer la structure de notre modèle générique de description de source, nous analysons et détaillons d'abord les informations à décrire pour le traitement de requêtes. Nous présentons ensuite l'organisation et la structuration des descriptions appliquées. Cette section montre les principes du fonctionnement de notre modèle dont la formalisation sera présentée dans la section 3.3.

3.2.1 Description des informations fondamentales

Nous commençons par décrire les informations fondamentales de source de données, qui incluent :

- les schémas décrivant les données hébergées et gérées par la source ;
- les opérateurs, manipulant et traitant les données, et pouvant être exécutés par la source. Ces derniers décrivent la capacité fonctionnelle de traitement de la source.

L'objectif est de pouvoir décrire, pour une source de données ou un médiateur intégrant des sources de données, les informations de bases liées directement aux données et aux manipulations possibles sur ces données.

Description des schémas de données

Nous choisissons la forme *Graphe* pour représenter les schémas des sources hétérogènes. Nous montrons pour chaque type de source, comment leurs schémas peuvent être présentés sous forme de graphes.

Source relationnelle Les sources relationnelles fournissent les méta-données incluant les noms des tables et de leurs attributs, ainsi que les types de ces attributs. Si nous représentons les schémas relationnels sous forme de graphe, chaque noeud dans le graphe correspond à un nom de table ou d'attribut et les arcs orientés indiquent la relation d'appartenance table/attributs.

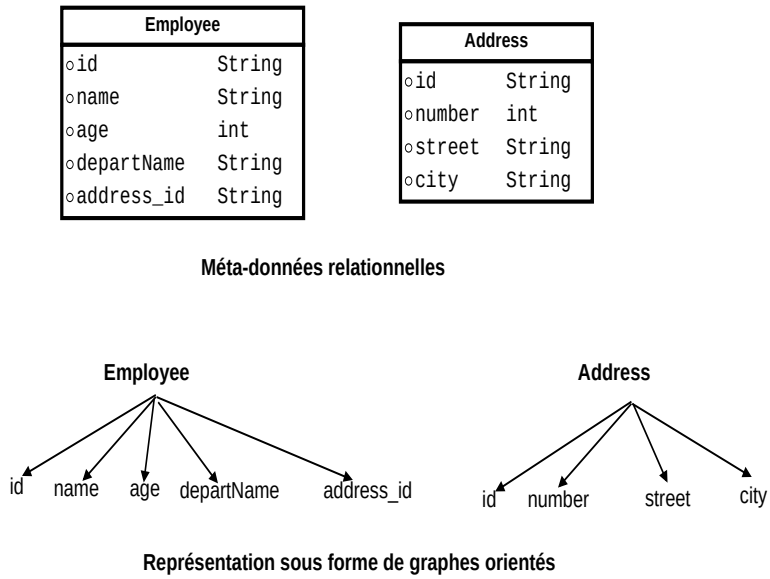

FIGURE 3.3 – Exemple de représentation de schémas relationnels en graphes

La figure 3.3 illustre un exemple de représentation des schémas concernant la source relationnelle S3 dans l'exemple dans la figure 3.1. Nous pouvons voir que les schémas des deux tables *Employee* et *Address* sont représentés par deux graphes orientés. Tous les attributs sont reliés avec des arcs aux noms des tables correspondants. Les types des attributs ne sont pas présentés dans les graphes. Ces informations seront présentées dans une autre partie de notre modèle (annotation), dont nous allons discuter plus tard dans la section.

Source objet-relationnelle Les base de données objet-relationnelles utilisent un modèle de données relationnel tout en permettant le stockage des objets. Dans ces bases de données, les associations d'héritage des objets ainsi la navigation entre les schémas s'ajoutent entre les tables(entités) du modèle relationnel.

Par exemple, pour la source objet-relationnelle S1 dans la figure 3.1, nous avons deux sous-types *Bookstore* et *MovieStudio* qui héritent le type *Company*. La figure 3.4 illustre la représentation des schémas de cette source sous forme de graphe. Nous pouvons voir les deux arcs entre *Company/Bookstore* et Company/MovieStudio sont utilisés pour décrire la relation d'héritage entre les types.

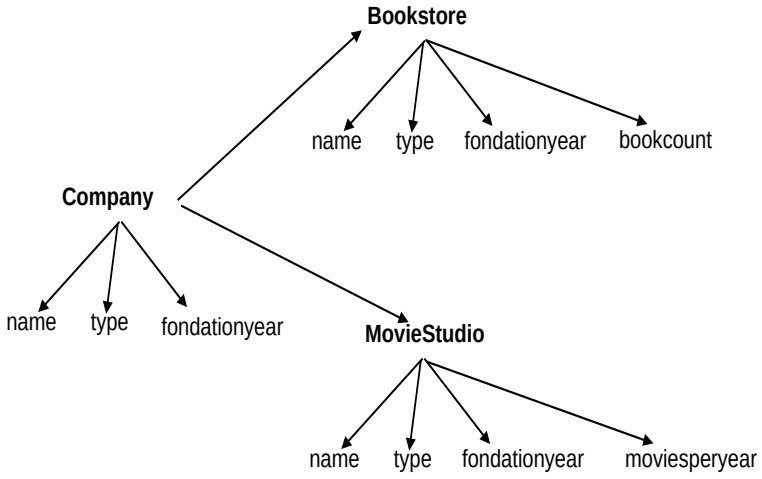

FIGURE 3.4 – Exemple de représentation de schémas objet-relationnels en graphes

Source semi-structurée En général, les méta-données d'une source semi-structurée sont représentées sous forme d'arborescence, qui peuvent facilement être représentées avec notre modèle graphes. La figure 3.5 illustre la représentation des schémas concernant *Department* dans la source S2 dans la figure 3.1.

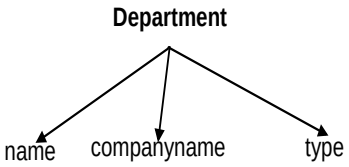

FIGURE 3.5 – Exemple de représentation de schémas semi-structurés en graphes

Source Service Web Les sources service Web ne divulguent généralement pas leurs schémas entiers des données hébergées sur les serveurs. Nous ne disposons souvent que des fonctions définies par ces services qui permettent de récupérer les résultats de requêtes utilisant ces fonctions. La description de schéma pour ces sources consiste à formuler des schémas partiels issus des informations divulguées par les fonctions.

Par exemple, la source Service Web S4 dans la figure 3.1 ne fournit qu'une méthode *getSalaryById(String id)* qui retrouve le salaire d'un employé par

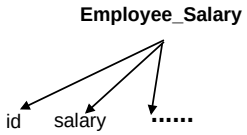

FIGURE 3.6 – Exemple de représentation de schéma pour une source service Web

son Id. Tout ce que nous connaissons, c'est que *Employee* contient un*id* et un *salary*. Donc nous pouvons décrire le schéma par un graphe qui est illustré dans la figure 3.6. Bien que partiel, cela représente toutes les informations concernant le schéma de la source à décrire.

Pour résumer, la forme graphe peut exprimer tous les types de schémas de source de données.

Description de capacité fonctionnelle de traitement : opérateurs

La capacité fonctionnelle de traitement consiste à définir la possibilité de l'exécution des opérateurs manipulant des données. Dans un système intégrant des sources hétérogènes réparties, il est nécessaire de connaître la capacité fonctionnelle de traitement de chaque source, afin de définir les sous-plans d'exécution envoyés à chaque source. Il est également crucial de définir la capacité de traitement du système gérant ces sources (les médiateurs).

Les opérateurs manipulent les données stockées sur les sources, la description des opérateurs est basée sur la description des schémas. Un opérateur peut avoir un ou plusieurs noeuds de schémas comme opérandes (paramètres). Nous ne pouvons définir un opérateur que sur les schémas existants de la source.

Pour un opérateur pouvant être exécuté par une source de données ou un médiateur, il est intéressant de préciser :

1. Le nom de l'opérateur indiquant sa fonctionnalité.
2. Les noeuds des schémas que l'opérateur peut manipuler.
3. L'endroit où l'opérateur s'exécute, qui doit être l'identifiant d'une source de données ou d'un médiateur à décrire.

Plus précisément, notre modèle de description doit permettre de décrire un opérateur pour plusieurs cas possibles :

- **L'opérateur ne peut manipuler que certains noeuds précis dans le schéma.** Dans ce cas-là, notre modèle doit permettre de préciser la localisation des noeuds concernés. Par exemple, pour la source S3 dans la figure 3.1 dont les schémas sont décrits dans la figure 3.3, nous pouvons avoir un opérateur de jointure qui peut s'exécuter uniquement entre le noeud *address_id* du graphe *Employee* de la source S3 et le noeud *id* du graphe *Address* de la même source (cf. la figure 3.3). Ici, tous les noeuds concernés sont issus de la source S3, mais quand nous devons décrire un opérateur inter-site sur un médiateur, il est nécessaire de préciser le nom de la sources de chaque noeud concerné.
- **L'opérateur peut s'effectuer sur des ensembles de noeuds se trouvant sur une source**. Par exemple, nous pouvons décrire un opérateur de jointure inter-site pouvant s'exécuter sur tous les noeuds de la source S2 et tous les noeuds de la source S3.
- **La description de l'opérateur peut être encore plus générale :** par exemple, nous pouvons effectuer une jointure entre n'importe quels noeuds dans l'intégralité des schémas du système.

3.2.2 Informations supplémentaires

Nous avons vu comment les schémas et les opérateurs sont décrits pour une source de données ou un médiateur. Ces informations ne suffisent pas pour le traitement efficace de requêtes. Nous devons ajouter des informations supplémentaires en nous appuyant sur ces éléments fondamentaux. Dans cette partie, nous introduisons notre système d'annotation qui permet d'associer des informations supplémentaires aux éléments fondamentaux d'une source de données ou d'un médiateur.

Description de la localisation de sources de données

La localisation de source est une information annotée sur les schémas de données. Tous les noeuds des graphes dans la description de schéma peuvent être annotés avec des informations de localisation. La localisation peut être présentée sous forme d'une chaîne de caractères contenant par exemple l'adresse physique du schéma. Les formats standard comme URI peuvent être utilisés. Selon le système gérant des sources de données, nous pouvons aussi annoter la localisation avec d'autres types de formats, par exemple, les coordonnées GPS, l'adresse IP, etc.

Dans l'exemple illustré dans la figure 3.1, chaque source de données est reliée à un médiateur avec un certain type de connexion. Les connexions peuvent être locales ou dans les réseaux à petite/grande échelle. Dans tous les cas, les médiateurs doivent connaître l'adresse unique de chaque source afin d'y accéder.

Description des informations de coût

Pour optimiser le traitement d'une requête en termes de temps d'exécution, nous devons comparer le coût des plans d'exécution candidats pour trouver le plan avec le coût minimal. Le calcul de coût s'appuie sur les informations liées aux opérateurs présents dans le plan d'exécution. Ces informations sont souvent sous forme de formules de coût permettant d'évaluer le coût de l'exécution de l'opérateur en fonction des paramètres (statistiques) sur les données et sur le système. Par exemple, la cardinalité des tables sélectionnées et des résultats intermédiaires, la sélectivité des prédicats de restriction ou de jointure, le nombre de valeurs distinctes des attributs, etc.

Pour construire un modèle générique permettant de décrire n'importe quelle information de coût, nous devons traiter les aspects suivants :

- Pour le même opérateur, nous pouvons annoter plusieurs informations de coût. Par exemple, nous pouvons annoter des algorithmes d'implémentation de l'opérateur et nous pouvons annoter différentes formules de coût avec différentes précisions, etc.
- Pour les informations statistiques, nous devons pouvoir annoter n'importe quelle granularité des schémas : chaque noeud, chaque relation (père-fils, héritage, ...), voire l'ensemble des graphes.
- Les informations de coût dans différentes annotations doivent rester cohérentes. C'est-à-dire que les noms des paramètres utilisés dans les formules doivent correspondre aux noms utilisés dans les statistiques. Par exemple, si nous avons une formule de coût utilisant une variable *card_employee* représentant la cardinalité de la table *Employee* dans la source S3 (Figure 3.3), la même variable doit se retrouver dans l'annotation de cardinalité.

Dans la section 3.4, nous allons présenter comment le calcul de coût peut s'effectuer en fonction des informations de coût annotées dans notre modèle de description.

Description d'autres types d'information

Notre modèle de description doit permettre de décrire n'importe quel type d'information. L'ensemble d'annotations de chaque type d'informations peuvent être groupées. Mais ces annotation doivent respecter plusieurs principes :

1. Toutes les informations doivent être représentées sous forme de chaîne de caractères.
2. L'annotation doit toujours s'appuyer sur les éléments fondamentaux de la source (schémas et opérateurs).
3. La fusion des annotations d'information ne doit pas perdre la précision d'information. Par exemple, une formule de coût pour un opérateur exécuté sur une source doit garder la localisation d'exécution (la référence de la source). Nous ne pouvons pas fusionner cette information avec des formules de coût annotées pour ce même opérateur mais l'exécution se trouve sur le médiateur.

3.2.3 Structure du modèle de description de sources

Le principe de notre modèle de description de source, c'est de décrire d'abord les éléments fondamentaux des sources liées directement aux données (schémas) et aux manipulation des données (opérateurs), et ensuite il est possible d'ajouter sur ces éléments autant d'informations supplémentaires utiles que nécessaire.

Les figures 3.7 3.8 illustrent un exemple de la structure de notre modèle de description. Nous voulons décrire le médiateur M2 dans la figure 3.1, qui doit inclure toutes les informations des médiateurs et des sources sous-jacents. Nous pouvons voir que les informations décrites sont organisées par **couches** :

- La couche des éléments fondamentaux (C0) contient tous les schémas représentés par les graphes et tous les opérateurs décrits en se basant sur les schémas. Pour chaque opérateur, l'ensemble de noeuds qu'il peut manipuler est décrit un utilisant les identifiants des sources et des noeuds. Ici, par exemple, *S3 :** signifie tous les noeuds de la source S3. *on S3* signifie que l'opérateur s'exécute sur la source S3, etc. Nous pouvons ainsi décrire n'importe quel type d'opérateur manipulant n'importe quels noeuds dans les schémas.
- La couche C1 au-dessus de la couche fondamentale consiste à décrire la sélectivité moyenne de chaque valeur distincte pour chaque attri-

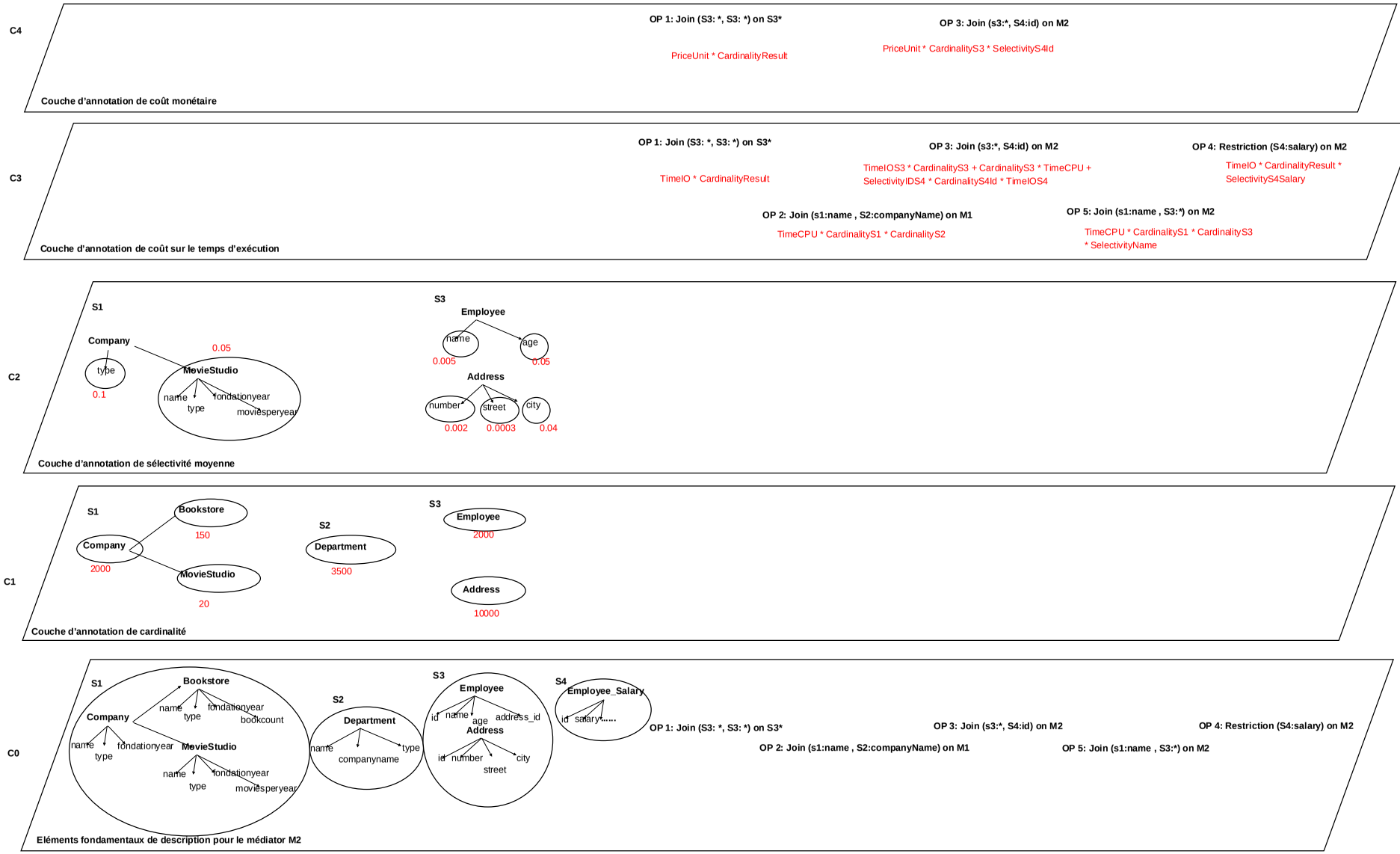

FIGURE 3.7 – Description d'un système de sources de données

FIGURE 3.8 – Description d'un système de sources de données (suite)

but. Dans cette couche, les sous-ensembles de noeuds des schémas sont annotés avec des sélectivités.
- La couche C2 est utilisée pour décrire la cardinalité de chaque entité/type. Nous ne prenons également que des sous-ensembles de noeuds (les noeuds désignant des entités/types) à annoter avec des entiers indiquant le nombre de tuples/objets/fils.
- La couche C3 contient des formules de coût sur le temps d'exécution annotées sur les opérateurs. Les formules sont des fonctions utilisant des paramètres correspondants aux statistiques comme les cardinalités et les sélectivités annotées dans d'autres couches.
- La couche C4 contient des formules de coût monétaire, ce qui est différent du modèle de coût sur le temps d'exécution annoté la couche C3. Les informations sont partielles car les formules de coût pour OP2, OP4 et OP5 sont inconnues.
- La couche C5 contient les informations statistiques sur le temps de traitement du CPU, le temps unitaire d'entrées-sorties de chaque source.
- La couche C6 décrit la localisation des sources exprimée par les URI.
- La couche C7 décrit certains droits d'accès aux données et aux opérateurs. Nous pouvons voir qu'en tant qu'un utilisateur ayant des droits restreints, il ne peut pas accéder à la source S4 et les opérateurs OP2, OP4 et OP5 sont indisponibles pour cet utilisateur.
- La couche C8 contient les annotations sur l'autonomie de batterie pour chaque source de données.
- Nous pouvons ajouter autant de couches d'annotation que nécessaire.

Nous voyons que après la description des schémas et des opérateurs, n'importe quel type d'information peut être ajouté (annoté) sur ces éléments. Il n'y pas de limitation sur la granularité des éléments annotés, qui peut être un noeud/opérateur, un ensemble de noeuds/opérateurs, tous les noeuds d'une source voire tous les noeuds existants, etc. Chaque élément fondamental a son identifiant unique, il est très facile de retrouver les informations pertinentes annotées dans différentes couches d'annotation.

Dans la section suivante, nous allons formaliser notre modèle générique de description de source (GSD). Nous allons spécifier chaque composant du GSD et montrer la manipulation de ces éléments pour décrire toutes les sources de données et tous les médiateurs.

3.3 GSD (Generic Source Description)

Dans cette section, nous formalisons notre modèle générique de description de sources, appelé GSD (*Generic Source Description*). Nous décrivons d'abord chaque composant du GSD. Ensuite, nous expliquons la création et la fusion des GSDs dans un système de médiation pour décrire non seulement des sources de données mais aussi des médiateurs intégrant des sources.

3.3.1 Formalisation du GSD

Nous utilisons certaines normalisations pour faciliter la présentation du GSD :

- A* signifie une collection de A ;
- A (B,C) signifie que A est composé de B et C.
- A | B signifie A ou B.

Node

Un noeud dans un graphe représentant un schéma de données est défini :

Node(nodeName, nodeLoc)

Il est composé de deux attributs :

1. *nodeName* représente le nom du noeud qui peut être un nom d'une table, d'un attribut d'une table, etc.
2. *nodeLoc* référence au nom du *site* dans laquelle se trouve le noeud. Nous allons voir la définition d'un *site* plus tard dans cette section.

Cette définition permet d'identifier un noeud qui est unique dans les schémas d'une source de données. Différentes sources de données peuvent avoir des noeuds ayant le même nom. Mais comme le nom de la source (*site*) est associé à chaque noeud, nous pouvons facilement les distinguer.

Par exemple, pour les schémas dans la figure 3.3, nous pouvons décrire le noeud *Employee* par *Node(Employee, S3)*, où *S3* indique que ce noeud est sur la source S3.

Edge

Un arc dans un schéma représente une relation entre deux noeuds, il est défini :

Edge(firstNode, secondNode, type)

La définition indique qu'un arc est composé d'un couple de noeuds :

1. *firstNode* référence le noeud de départ pour l'arc.
2. *secondNode* référence le noeud de destination.
3. *type* indique le type de relation entre les deux noeuds, qui peut être attribut, héritage, etc.

Comme nous prenons des graphes orientés pour représenter des schémas, ce couple de noeuds indique la direction de l'arc. Par exemple, dans la figure 3.3, l'arc entre *Employee* et *Name* peut être décrit : *Edge(Node(Employee, S3), Node(name, S3), attribute).*

Graph

Un graphe dans le schéma représente une entité, un type, etc. Il est défini :

Graph(Node*, Edge*)

Operator

Nous pouvons manipuler les données par les opérateurs algébriques, par exemple, restriction, jointure, projection, etc. Les opérateurs peuvent être effectués sur les données au sein d'une source de données, ce sont des opérations *locales*. Dans un environnement réparti, des opérateurs peuvent être effectués également entre deux sources(ou plus). Ce sont des opérateurs *inter-sites*. Afin de décrire toutes les opérateurs possibles, nous donnons la définition suivante :

Operator(opName, (Node*, ..., Node*), opLoc)

Dans cette définition, *opName* représente le nom de l'opérateur. *(Node*, ..., Node*)* est un ensemble de collections de noeuds. La collection à l'intérieur *Node** signifie un paramètre de l'opérateur pouvant être un ensemble de noeuds. Et un opérateur peut avoir un ensemble de paramètres. Le dernier composant *opLoc* indique sur quel site l'opérateur s'exécute. Par exemple, pour une jointure définie pour un médiateur, elle peut joindre les données issues des sources intégrées par le médiateur, mais l'exécution de la jointure peut s'effectuer sur le médiateur.

Dans la figure 3.7, les cinq opérateurs illustrés sont décrits en utilisant notre format défini ci-dessus.

1. Join(S3 :*, S3 :*) on S3
2. Join(S1 :name, S2 :companyName) on M1
3. Join(S3 :*, S4 :id) on M2
4. Join(S1 :name, S3 :*) on M2
5. Restriction(S4 :salary) on M2

Site

Dans un système d'information, il est non seulement intéressant de décrire une source seule, mais aussi intéressant de décrire un ensemble de sources qui sont manipulées par exemple par un médiateur. Nous pouvons aussi décrire des opérateurs pouvant être effectués au sein d'une source ou entre les sources. Nous pouvons dire qu'une source ou un médiateur de sources est composé d'un ensemble de schémas et d'opérations.

Nous définissons une source de données ou un médiateur de sources de données par la notion *Site* :

Site(siteName, Graph*, Operator*)

Le *siteName* représente l'identifiant unique d'un site (Ses utilisations peuvent être trouvées dans la définition de *Node* et *Operator*. La collection *Graph** et la collection *Operator** sont des éléments de base associés à ce site.

Annotation

Dans le processus de traitement de requêtes, il est nécessaire d'avoir des informations autres que les schémas et des opérations. Comme nous l'avons expliqué dans la section précédente, ces informations doivent être associées

aux éléments de base de la source. Nous appelons ces informations les *Annotations*.

Annotation((Node | Operator)*, information)

Une annotation est composée d'un ensemble d'éléments fondamentaux (noeuds ou opérateurs) sur lesquels une information (de type String) est annotée. Pour le même élément de base, nous pouvons avoir plusieurs annotations. Ces différentes annotations donnent différents types d'informations qui peuvent être localisation, coût, etc. Les exemples d'annotation peuvent être retrouvés dans la figure 3.7.

Layer

Pour regrouper toutes les annotations du même type, nous utilisons une notion de *couche*. Chaque couche a un nom indiquant le type de toutes les annotations appartenant à cette couche.

Layer(layerName, Annotation*)

layerName indique le type de la couche d'annotation et la collection *Annotation** contient toutes les annotations dans la couche. Dans la figure 3.7, nous avons plusieurs couches d'annotation ajoutées sur la couche fondamentale.

GSD

Enfin, notre modèle GSD concerne tous les éléments définis ci-dessus :

GSD (Site*, Layer*)

Le GSD est composé de

- tous les sites contenant des schémas de données et des opérateurs possibles sur les données ;
- toutes les couches d'annotations qui ajoutent différents types d'information sur les éléments de base des sites.

Synthèse du GSD

Le tableau 3.3.1 résume les composants du GSD que nous avons formalisés.

Composant du GSD	Définition	Signification
Node	Node(nodeName, nodeLoc)	Représente le nom d'une table, entité, attribut, étiquette, etc.
Edge	Edge(firstNode, secondNode, type)	Représente une relation.
Graph	Graph(Node*, Edge*)	Représente un schéma.
Operator	Operator(opName, (Node*, ..., Node*), opLoc)	Représente un opérateur pouvant être exécuté par la source
Annotation	Annotation((Node | Operator)*, information)	Représente l'ajout d'une information sur un FloorElement
Layer	Layer(layerName, Annotation*)	Représente un ensemble d'annotation du même type
GSD	GSD (Site*, Layer*)	Représente l'intégralité des informations décrites pour une source de données

3.3.2 Construction du GSD

Nous avons défini différents composants de notre modèle GSD. Ce modèle permet de décrire tous les types d'informations pour une source de données. Dans un système de médiation, nous avons besoin de décrire chaque source de données par leur adaptateur, ensuite, nous devons fusionner les GSD sur chaque médiateur intégrant des sources. Dans cette section, nous allons d'abord montrer comment un GSD est construit pour une source de données, ensuite nous décrirons la construction du GSD pour un médiateur en définissant la fusion des GSD.

Construction du GSD d'une source de données

Le GSD d'une source de données est composé de schémas propres à la source, d'opérateurs pouvant être effectués sur les données, des couches contenant différents types d'informations supplémentaires annotées sur les schémas et les opérateurs. Pour construire ce GSD, nous devons d'abord construire les schémas et les opérateurs, et ensuite ajouter des couches d'annotation s'appuyant sur ces éléments de base.

Nous définissions plusieurs fonctions de base pour construction d'un GSD. Ces fonctions permettent d'initialiser un GSD et d'ajouter des éléments progressivement.

Création d'un GSD Cette fonction permet de créer un GSD vide ayant juste son nom. Le nom référence au nom de la source pour laquelle le GSD est défini.

$$create() \rightarrow GSD$$

Ajout d'un schéma Cette fonction permet d'ajouter un graphe représentant d'un schéma des données. Comme ce qui est décrit dans la section précédente, un schéma a une référence sur le nom de la source à laquelle le schéma appartient.

$$add(GSD, graph) \rightarrow GSD$$

Ajout d'un opérateur Cette fonction permet d'ajouter un opérateur dans un GSD. Comme l'opérateur ne peut s'effectuer que sur les données de la source qu'il décrit, les paramètres de l'opérateur référencent au nom de la source.

$$add(GSD, operator) \rightarrow GSD$$

Ajout d'une couche d'annotation Les annotations du même type sont organisées sur la même couche (*layer*). Cette fonction permet d'ajouter une couche d'annotation dans un GSD. Les annotations dans la couche doivent s'appuyer sur les schémas et les opérateurs. Chaque couche définit une projection de ces éléments de base avec des informations supplémentaires ajoutées.

$$add(GSD, layer) \rightarrow GSD$$

Avec ces quatre fonctions, nous pouvons créer un GSD vide et l'enrichir avec des éléments de base et des annotations.

Construction du GSD d'un médiateur

Pour un médiateur intégrant plusieurs sources de données sous-jacentes, nous devons effectuer la fusion de tous les GSD de ces sources dans le GSD du médiateur. Nous définissions ainsi des fonctions permettant de fusionner des GSD. La fusion des 2 GSD consiste à fusionner les schémas, les opérateurs et les couches d'annotation.

Fusion des graphes Cette fonction définit la fusion des schémas issus de deux sources de données au niveau du médiateur. Le résultat de la fusion et l'union de tous les schémas des deux sources. Chaque schéma garde le nom de la source originale. Ceci permet d'assurer la cohérence de description des opérateurs contenant sa localisation de l'exécution.

Soit G1, G2 deux ensembles de graphes,
$merge(G1, G2) = G1$ UNION $G2$

Fusion des opérateurs La fusion des opérateurs issus des deux sources de données donne l'union de tous les opérateurs décrits pour les deux sources.

Soit O1, O2 deux ensembles d'opérateurs,
$merge(O1, O2) = O1$ UNION $O2$

Fusion des couches d'annotation Puisque les schémas et les opérateurs sont fusionnés, ceci forme au médiateur un ensemble unique des éléments de base. Les couches d'annotation du même type dans les deux sources doivent être fusionnées et forme une couche contenant l'union des annotations de ce type. Par exemple, deux couches d'annotation sur le modèle de coût monétaire peuvent être fusionnées en une couche.

Soit L1, L2 deux ensemble de couches d'annotation, $merge(L1, L2)$ est défini :

1. Créer un ensemble de couches vide : L3.
2. Pour chaque couche d'annotation $\mathcal{L}$ dans L1, créer une couche vide L' dans L3 et ajouter toutes les annotations de $\mathcal{L}$ dans L'. Vérifier s'il existe une couche dans L2 ayant le même type (*layerName*), si oui, ajouter toutes les annotations de cette couche dans L'.

3. Pour toutes les couches de L2 non traitées dans l'étape 2, les ajouter directement dans L3.

Construction de GSD d'un médiateur

Le médiateur peut contenir lui-même des schémas et des opérateurs définis, voire les couches d'annotation. Après la fusion des schémas, des opérateurs et des couches d'annotation des sources, il suffit d'appeler les fonctions d'ajout décrites ci-dessus pour ajouter ces éléments au GSD du médiateur. La construction s'effectue en plusieurs étapes :

1. Créer un GSD vide G pour le médiateur :
   $G = create()$
2. Ajouter dans G le résultat de la fusion des graphes (G1, G2, ...) issus des GSDs des sources intégrées :
   $G = add(G, merge(merge(G1, G2), G3)...)$
3. Ajouter dans G le résultat de la fusion des opérateurs (O1, O2, ...) issus des GSDs des sources intégrées :
   $G = add(G, merge(merge(O1, O2), O3)...)$
4. Ajouter dans G le résultat de la fusion des couches d'annotation (L1, L2, ...) issues des GSDs des sources intégrées :
   $G = add(G, merge(merge(L1, L2), L3)...)$
5. Ajouter les schémas propres (G0) à ce médiateur s'il en existe :
   $G = add(G, G0)$
6. Ajouter les opérateurs propres (O0) à ce médiateur s'il en existe :
   $G = add(G, O0)$
7. Ajouter les couches d'annotation propres (L0) à ce médiateurs s'il en existe :
   $G = add(G, L0)$

Dans la section suivante, nous allons intéresser à l'application du modèle GSD pour décrire les informations de coût.

3.4 Application du GSD : Estimation du coût des plans d'exécution

Nous avons proposé le modèle GSD permettant de décrire toutes les informations concernant le traitement de requêtes dans les systèmes d'information. L'estimation du coût des plans d'exécution est une étape cruciale

dans l'optimisation du traitement de requêtes. Dans cette section, nous allons montrer comment utiliser le GSD pour estimer le coût des plans d'exécution.

La section 3.4.1 introduit un langage descriptif utilisé pour exprimer d'une façon uniforme les formules de coût annotées dans le GSD. La section 3.4.2 spécifie certaines étapes nécessaires dans le traitement de requêtes afin d'obtenir un plan d'exécution valide et d'estimer son coût sous l'aide du GSD.

3.4.1 Langage de description de coût

Dans un système de médiation, d'un côté, chaque adaptateur peut utiliser le GSD pour décrire les formules de coût des opérateurs sur la source de données qu'il intègre, d'un autre côté, les formules de coût pour les opérateurs exécutés sur les médiateurs sont décrites par ces médiateurs dans leurs GSD. Toutes ces formules de coût ne sont pas indépendantes. Par exemple, l'estimation du coût d'une jointure exécutée sur un médiateur dépend de la taille des résultats intermédiaires et du temps de l'exécution de ses opérateurs sous-jacents.

Dans le GSD, les formules de coût sont annotées sur les opérateurs sous forme de chaîne de caractères dans les couches d'annotation. Pourtant, les chaînes de caractères ne permettent pas une manipulation facile pour extraire les informations de coût pertinentes. Nous avons besoin de définir un format uniforme permettant d'assurer la description des coûts telle que :

- Chaque formule décrite peut être analysée d'une manière uniforme.
- Les variables utilisées dans les formules doivent être cohérentes, c'est-à-dire que avec toutes informations de coût, nous devons pouvoir connaître ou calculer la valeur de chaque variable.

Pour atteindre ces objectifs, nous utilisons un langage semi-structuré basé sur MathML pour exprimer les formules de coût annotées.

Langage semi-structuré basé sur MathML

Nous choisissons le modèle semi-structuré, dont le principal format est XML, pour notre langage descriptif de coût. Le modèle semi-structuré a plusieurs avantages comme modèle de description :

- il facilite l'échange automatisé de contenus entre systèmes d'informations hétérogènes.
- il est définissable et validable par un schéma (par exemple DTD schéma).

– un document XML est entièrement transformable dans un autre document XML.

MathML [W3C 2003] est un langage basé sur XML permettant l'expression de symboles mathématiques, notamment sur Internet. Ce langage est capable d'exprimer les formules de coût de façon concise. MathML ne s'occupe pas uniquement de la présentation mais aussi du sens des différents composants des formules mathématiques. Dans sa feuille de style, nous pouvons retrouver les formats standards pour ces différents composants. Par exemple,

– Les identifiants (par exemple les noms de variables, qui sont des textes et symboles devant être affichés tels quels), sont encadrés par les balises <mi>...</mi>.
– Les opérateurs sont encadrés par les balises <mo>...</mo>.
– Les nombres sont encadrés par les balises <mn>...</mn>.

Ces formalisations permettent de décrire les formules de coût de façon uniforme pour les adaptateurs et les médiateurs. De plus, il existe des implémentations du *parser* de MathML qui pourront être réutilisées dans l'analyse des formules de coût annotées.

Nous allons ensuite montrer avec un exemple, comment utiliser ce langage pour exprimer des formules de coût.

Description des formules de coût

Les formules doivent utiliser de manière cohérente les variables définies dans les couches d'annotation. Elles sont exprimées sous forme de système d'équations dont chaque équation correspond à une formule de coût.

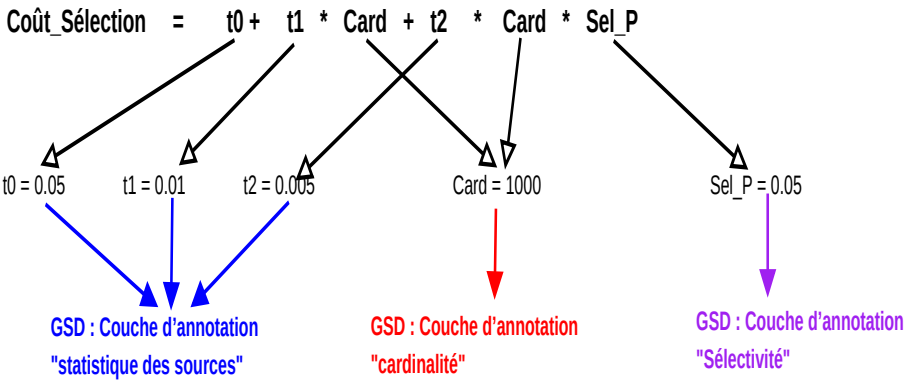

FIGURE 3.9 – Un exemple de modèle de coût

La figure 3.9 illustre une formule de coût issue de la méthode par calibration [Du *et al.* 1992] pour calculer le coût d'exécution d'une sélection avec un prédicat sur une relation. Les valeurs des variables dans la formule peuvent être retrouvées sous l'aide des couches d'annotation du GSD et les résultats sont décrits en équations. Le variable $t0$ désigne le temps pour initialiser la lecture de la relation, $t1$ le temps unitaire pour retrouver un tuple satisfaisant le critère de sélection, $t2$ le temps unitaire pour projeter un tuple sélectionné. Les valeurs de ces trois variables sont issues de la couche d'annotation sur les statistiques du système gérant la source. $Card$ la cardinalité de la relation, dont la valeur est issue de la couche d'annotation des cardinalités. Sel_P est la sélectivité du prédicat de la sélection, dont la valeur est issue de la couche d'annotation des sélectivités.

La figure 3.10 illustre une partie du ficher XML décrivant cette formule de coût avec notre langage descriptif de coût au format XML. Nous pouvons retrouver les variables et les opérateurs encadrés par les balises définies dans la feuille de style de MathML.

```
<costs source="relational">
      <apply><eq/>
            <apply><mi>costselection</mi></apply>
            <apply><plus/>
                  <mi>t0</mi>
                  <apply><times>
                        <mi>t1</mi>
                        <mi>Card</mi>
                  </apply>
                  <apply><times>
                        <mi>t2</mi>
                        <mi>Card</mi>
                        <mi>SelP</mi>
                  </apply>
             </apply>
      <apply>
            ......
</costs>
```

FIGURE 3.10 – Un exemple de formule de coût exprimée par le langage descriptif

Avec ce système d'équations, nous pouvons calculer le coût d'une sélection sur cette relation ayant des statistiques décrites dans les équations :

$$Cout\_Selection = 0.05 + 0.01 * 1000 + 0.005 * 1000 * 0.05 = 10.30$$

3.4.2 Estimation de coût des plans

La figure 3.11 illustre les étapes nécessaires dans le traitement d'une requête.

1. Quand une requête est envoyée au système, elle est tout d'abord analysée pour construire *arbre algébrique logique* contenant les opérateurs traduits de différentes parties de la requête et des données accédées.
2. Ensuite, ce arbre logique est annoté avec les informations liées à l'exécution et l'optimisation et devient un *arbre algébrique physique*, qui est prêt à être exécuté pour produire le résultat de la requête.
3. Le coût de l'exécution de l'arbre physique peut être estimé.

Le GSD fournit les informations nécessaires pour la réalisation de ces trois étapes. Nous détaillons ci-dessous chaque étape.

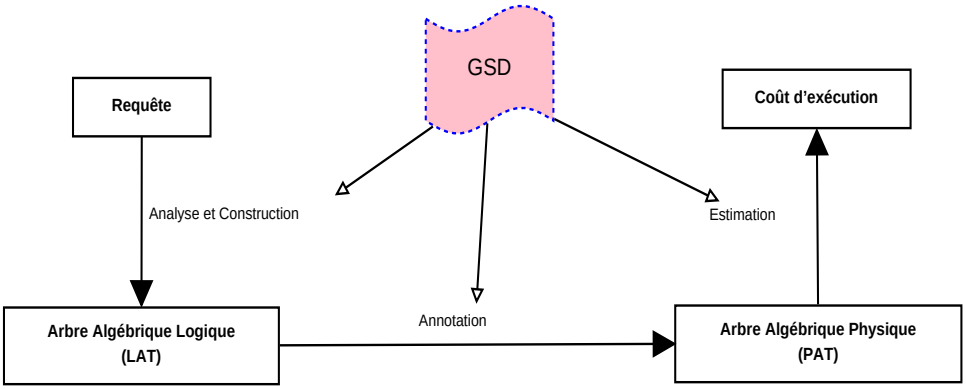

FIGURE 3.11 – Estimation du coût

Arbre Algébrique Logique (LAT : *Logical Algebra Tree*)

L'arbre algébrique logique (cité comme **LAT** dans le reste de la thèse) contient des opérateurs algébriques issus de l'analyse de la requête originale. L'algèbre utilisée doit contenir tous les opérateurs existants dans toutes les sources et médiateurs intégrés. Dans le travail sur le *Tree Graph View (TGV)* [Travers *et al.* 2006b], une algèbre abstraite a été utilisée pour représenter les opérateurs du modèle TGV, qui permet de représenter les requêtes envoyées

aux systèmes de médiation intégrant des sources de données très hétérogènes : relationnelle, objet-relationnelle, semi-structurée, etc. Nous reprenons cette algèbre comment notre modèle pivot pour représenter les plans d'exécution.

Le LAT définit l'organisation structurée des opérateurs, telle que :

- Pour chaque noeud (sauf les feuilles) de l'arbre, il a une entrée représentant les données que l'opérateur manipule et une sortie représentant le résultat de l'exécution de l'opérateur.
- La racine représente l'opérateur fournissant le résultat final de la requête.
- Les feuilles représente les données de source accédées par la requête.
- L'ordre de l'exécution des opérateurs est défini par l'arbre : depuis les feuilles vers la racine.

La figure 3.12 illustre le LAT correspondant à la requête citée dans les sections précédentes. Nous pouvons voir :

1. Les quatre opérateurs *Sélection* depuis *Company, Department, Employee* et *Address*.
2. Un opérateur spécial *Appel de fonction* définie par la source Service Web : *getSalaryById(id)* depuis *Salary*.
3. Les opérateurs jointure et restriction sont reliées à la sorties des opérateurs décrits dans 1 et 2 avec leurs prédicats.

Le LAT décrit de façon sémantique la solution pour résoudre la requête mais ne spécifie pas la manière précise de l'exécution. Il manque des informations supplémentaires permettant d'aboutir l'exécution de la requête : localisation des sources accédées, algorithmes d'exécution pour les opérateurs sur les médiateurs, etc. Ces informations se trouvent dans les couches d'annotation du GSD et nous devons les associer à LAT pour obtenir un plan d'exécution prêt à être exécuté.

Arbre Algébrique Physique (PAT : *Physical Algebra Tree*)

Pour avoir un arbre algébrique physique (cité comme **PAT** dans le reste de la thèse) d'une requête exécutée dans un système de médiation, nous devons spécifier plusieurs aspects :

- Les opérateurs pouvant être exécutées sur la même source de données doivent être regroupés dans un sous-plan d'exécution (s'ils se trouvent tous dans un sous-arbre de LAT).

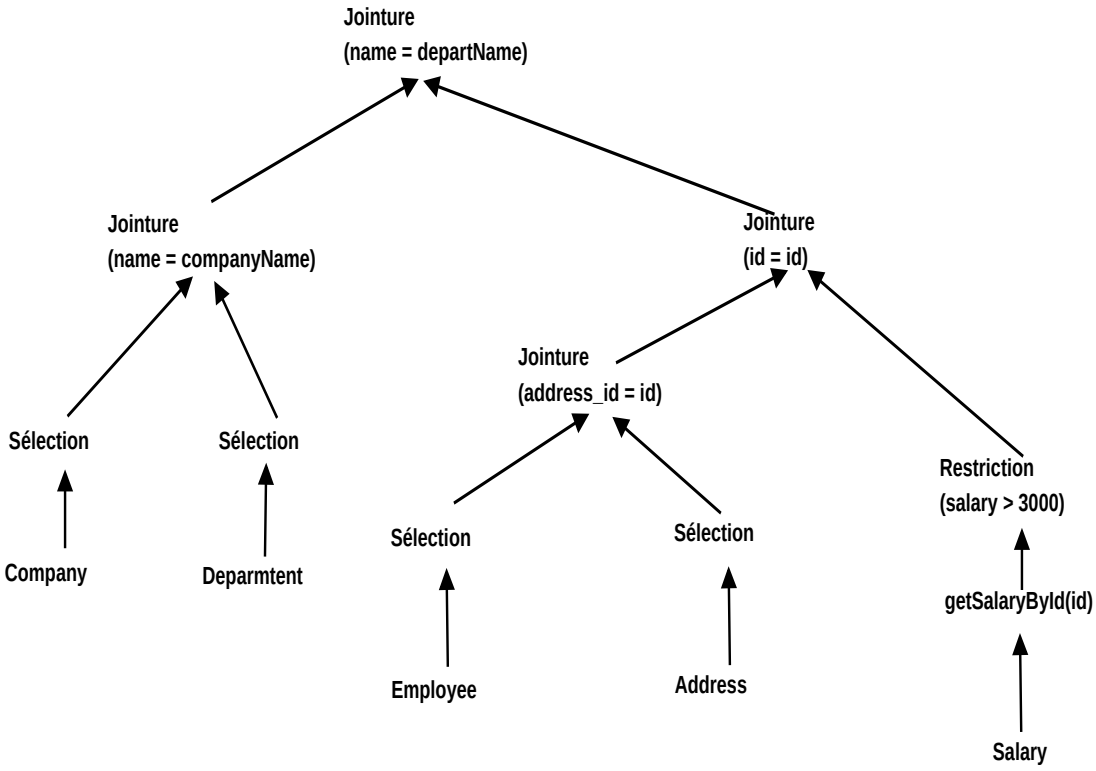

FIGURE 3.12 – Arbre Algébrique Logique de la requête

- Les opérateurs ne pouvant pas être exécutées par une source doivent être marqués par une localisation de l'exécution sur un certain médiateur.
- Pour les opérateurs exécutés sur les médiateurs, l'algorithme d'exécution utilisé doit être spécifié s'il existe plusieurs algorithmes possibles.
- Pour estimer le coût d'exécution du plan, les informations liées aux coûts des opérateurs doivent être annotées permettant les éventuelles optimisations du plan.

La figure 3.13 illustre le PAT correspondant au LAT dans la figure 3.12. Nous pouvons voir :

- Quatre sous-plans sont définis avec quatre sous-arbres compris dans quatre cercles avec des couleurs différentes. La délimitation des sous-plans est basée sur la capacité fonctionnelle de traitement de chaque source (les opérateurs disponibles) décrit dans le GSD.
- Tous les opérateurs sur les médiateurs sont annotés avec la localisation d'exécution et l'algorithme s'il y a des choix. Par exemple, *I-M (In-Memory)* signifie l'algorithme de jointure en-mémoire, *N-*

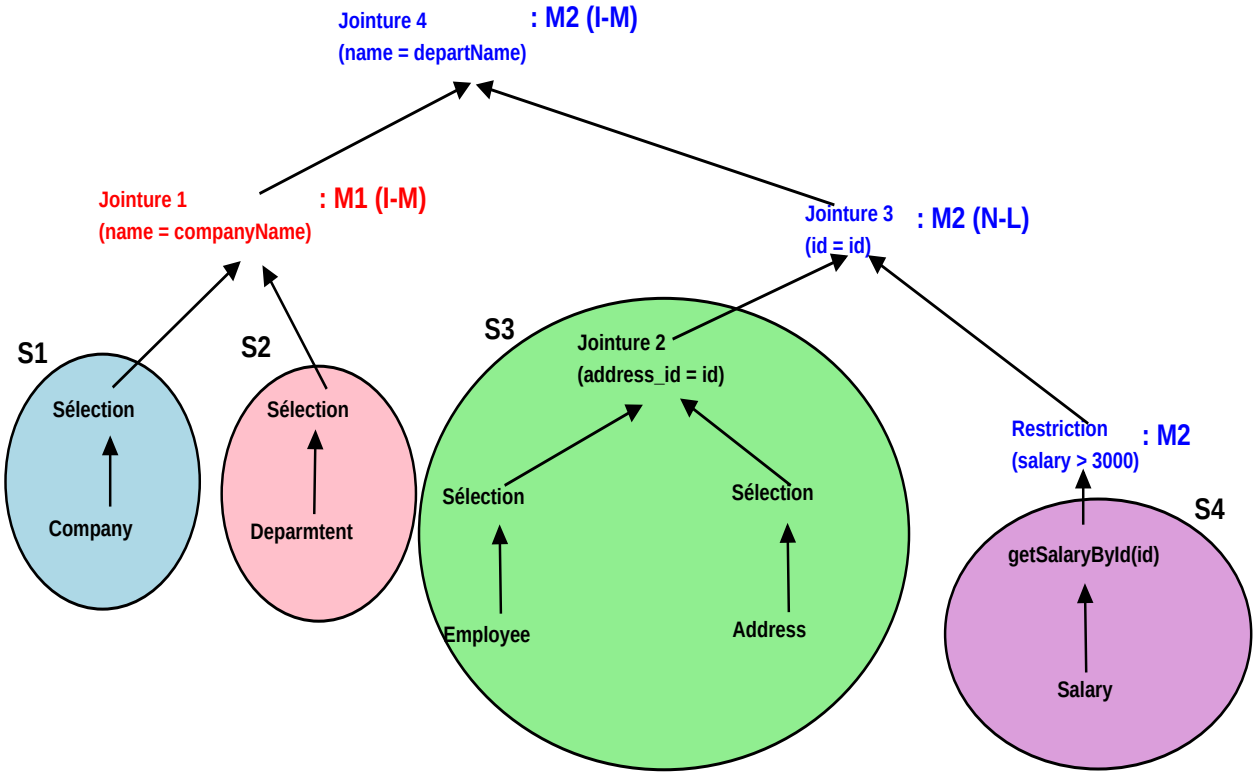

FIGURE 3.13 – Arbre Algébrique Physique de la requête

L(Nested-Loop) signifie l'algorithme de jointure par boucle imbriquée, etc. Dans le GSD, nous pouvons facilement extraire ces informations dans les couches d'annotation correspondantes(description des opérateurs et algorithmes, etc.).

Estimation de coût de PAT

Avec les informations de coût associées aux opérateurs d'un PAT, nous pouvons estimer son coût d'exécution. La figure 3.14 illustre un algorithme récursif de parcours préfixe de l'arbre. Cet algorithme permet l'estimation de coût de PAT représenté récursivement par le noeud racine de l'arbre. Nous expliquons le fonctionnement de l'algorithme comme suit :

- La **ligne 2** consiste à trouver la formule de coût associée à l'opérateur courant à traiter. Par la localisation de l'exécution de l'opérateur, nous pouvons trouver la formule de coût dans la couche d'annotation du GSD de la source (ou médiateur) exécutant l'opérateur.
- La **ligne 3** consiste à trouver dans le GSD, les valeurs des variables utilisées dans la formule de coût, qui peuvent être les cardinalités,

```
1  calculer(PAT A) {
2    Trouver la formule de coût pour A;
3    Trouver les valeurs des variables dans la formule;
4    Préparer la liste des variables indéterminées;
5
6    Pour chaque noeud-fils A' de A
7        Si A' n'est pas vide
8            calculer(A');
9
10   Associer les valeurs aux variables non derterminées;
11   Calculer le résulat de la formule;
12 }
```

FIGURE 3.14 – Algorithme de l'estimation de PAT

sélectivités ou les variables décrivant les statistiques sur le système (cf. 3.4.1.

- Pourtant, les valeurs des variables dépendant des résultats intermédiaires et de l'exécution des opérateurs fils ne peuvent pas être déterminées. La **ligne 4** prépare la liste de ces variables afin de pouvoir trouver leurs valeurs plus tard.
- Les **lignes 6-8** consiste à calculer le coût pour chaque noeud-fils du noeud courant, en appelant récursivement la même fonction (**ligne 1**).
- Après le calcul des noeuds-fils, les valeurs des variables non déterminées doivent être retrouvées (**ligne 10**).
- Enfin, la **ligne 11** calcule le résultat de la formule pour obtenir le coût de l'opérateur. Et si c'est le noeud racine, cela sera le coût du PAT.

3.5 Conclusion

Dans ce chapitre, nous avons proposé un modèle générique pour la description des sources de données. Ce modèle permet de décrire tous les types d'informations pour tous les types de systèmes gérant des sources.

Nous avons défini les différents composants du GSD permettant de décrire chaque type d'information d'une manière hiérarchisée. Le modèle GSD a plusieurs avantage :

- Le GSD utilise le format de graphe pour décrire les schémas, ceci résout l'hétérogénéité des sources parce que tout peut être représenté par des graphes ;
- Les opérateurs décrits avec des opérandes (paramètres) référençant aux schémas. Ceci permet de décrire des opérateurs avec une précision. Les opérateurs d'une source ou inter-sites sont bien distingués.
- La description des informations supplémentaires est classifiée par les couche d'annotation, ceci facilite l'extraction des informations pertinentes.
- Les schémas et les opérateurs peuvent être annotés en n'importe granularité, ceci permet de décrire les informations de n'importe précision.

Nous avons montré en appliquant le modèle GSD, nous pouvons décrire des informations de coût. Avec la description de coût, nous pouvons calculer le coût des plans d'exécution. Ceci est utile dans le cadre de l'optimisation de requêtes. Dans le chapitre suivant, nous allons introduire notre cadre générique d'optimisation de requêtes.

Chapitre 4

Cadre Générique d'Optimisation

4.1 Introduction

Dans le chapitre précédent, nous avons proposé un modèle générique GSD, qui permet de décrire les informations utiles sur les sources de données pour le traitement de requêtes. Dans ce chapitre, nous allons nous focaliser sur un processus important dans le traitement de requête : optimisation des plans d'exécution.

Dans la section 3.4, nous avons montré qu'un LAT (*Logical Algebra Tree*) correspondant à une requête est transformé en un PAT (*Physical Algebra Tree*) pour l'exécution. L'utilisation de ce PAT initial pourrait engendrer une exécution non-performante car ce PAT n'est souvent pas la meilleure solution pour résoudre la requête. Un traitement spécifique est donc sollicité pour modifier le PAT initial, afin de trouver un autre PAT équivalent dont l'exécution produit exactement les mêmes résultats, mais avec un coût optimal. Pour atteindre cet objectif, un *processus d'optimisation* est appliqué par le système.

Pendant ce processus d'optimisation, le PAT initial est réécrit un certain nombre de fois en utilisant des *règles de transformation*, jusqu'à obtenir le plan optimal d'après certains critères (par exemple, le temps d'exécution minimal, le coût monétaire minimal, l'utilisation de ressources du système minimale etc.). Pour mieux explorer les solutions possibles, l'application des règles de transformation est contrôlée par une *Stratégie de Recherche* issue des algorithmes métaheuristiques, basée en général sur le coût d'exécution des plans candidats. L'ensemble de ces éléments intégrés dans le système pour trouver la meilleure solution pour les requêtes utilisateurs compose un

module essentiel du système : l'*optimiseur*.

Dans un système traitant des requêtes hétérogènes réparties, afin d'améliorer le traitement de ces requêtes, la conception et l'implémentation d'un optimiseur performant deviennent cruciales. Comme expliqué ci-dessus, la performance d'un optimiseur dépend de la stratégie de recherche utilisée, des règles de transformation et de l'efficacité du calcul de coût des plans. Nous avons besoin de résoudre trois problématiques principales :

1. Nous avons besoin d'un cadre d'optimisation intégrant ces différents éléments pour avoir un optimiseur fonctionnel. Une architecture (organisation) de ces éléments doit être définie pour faciliter le déroulement de l'optimisation utilisant ces éléments.
2. Ce cadre doit être générique pour pouvoir intégrer différents types d'éléments (différentes stratégies de recherches, règles, ou modèles de coût). L'ajout ou le changement des éléments ne doit pas demander une modification majeure de l'implémentation de l'optimiseur.
3. Ce cadre doit fournir des modules unitaires génériques et réutilisables pour faciliter l'implémentation des optimiseurs ayant différents objectifs d'optimisation de requêtes.

Dans ce chapitre, en prenant en considération les critères ci-dessus, nous proposons un cadre générique d'optimisation pour faciliter la construction des optimiseurs dans les systèmes de sources de données hétérogènes réparties. Ce cadre utilise le modèle générique de description des sources (GSD) proposé dans le chapitre 3 pour réaliser certaines fonctionnalités nécessaires pour optimiser les requêtes. Le cadre permet surtout d'intégrer très facilement, sans demander beaucoup de nouvelles implémentations, les différentes stratégies de recherche qui pilotent le processus d'optimisation.

Ce chapitre est organisé comme suit : Dans la section 4.2, nous allons tout d'abord discuter de l'espace de recherche en analysant les caractéristiques des règles de transformation équivalentes. Ensuite, nous allons présenter dans la section 4.3 notre cadre générique d'optimisation avec ses fonctions unitaires. Pour montrer la généricité du cadre, la section 4.4 illustre l'application de certains algorithmes de stratégie de recherche dans notre cadre générique d'optimisation, allant des algorithmes naïfs comme *Exhaustif* aux algorithmes plus complexes comme *Génétique*. Dans la section 4.5, nous expliquerons comment utiliser notre cadre générique pour construire des optimiseurs ayant différents critères d'optimisation. Nous conclurons le chapitre dans la section 4.6 pour résumer notre contribution sur la proposition de ce cadre.

4.2 Espace de recherche

L'espace de recherche est une collection de solutions possibles à une problématique. Il peut être défini par des variables discrètes ou continues. Dans un processus d'optimisation, l'espace de recherche est dédié à trouver le plan d'exécution optimal pour une requête envoyée au système, chaque variable correspond donc à un plan d'exécution candidat (PAT). Ces variables sont ainsi discrètes et dans le contexte d'optimisation de requêtes basée sur le coût, la solution optimale dans cet espace est celle avec un coût minimal.

La taille de l'espace de recherche dépend des contraintes du problème. Dans le contexte d'optimisation de requêtes, pour le PAT correspondant à la requête en question, il existe un ensemble de règles de transformation pouvant être appliquées sur le PAT pour générer d'autres PAT candidats. De plus, si l'environnement est réparti, certaines règles peuvent avoir des contraintes de validation en raison des capacités fonctionnelles de traitement restreintes des sources ou du médiateur. Un opérateur du PAT peut être exécuté soit sur le médiateur soit sur la source, cela enrichit l'ensemble de règles possibles pour transformer le PAT, et ainsi agrandit l'espace de recherche.

Dans cette section, nous nous focalisons sur des caractéristiques des règles de transformation utilisées pour construire l'espace de recherche.

4.2.1 Règles de transformation

Une *règle de transformation* consiste à modifier un PAT afin de générer un autre PAT *équivalent* en termes du résultat de l'exécution de PAT. Dans [Ioannidis 1996], [Cherniack et Zdonik 1998], [Ali et Moerkotte 2004], nous pouvons trouver les règles de transformation équivalentes pour modifier un arbre algébrique. Dans cette thèse, nous ne tentons pas d'inventer de nouvelles règles mais de résumer les caractéristiques des règles liées à l'optimisation :

1. Certaines règles de transformation sont atomiques, cela signifie que chaque règle est dédiée à modifier un seul opérateur du PAT. Ce sont les règles qui modifient les attributs d'un opérateur. Par exemple, une règle qui change l'algorithme d'exécution d'une jointure, de *boucle imbriquée* à *tri par fusion* ;
2. Certaines règles permettent de modifier l'ordre des opérateurs dans un PAT (déplacement, ajout, suppression, fusion ou scission des opé-

rateurs). Par exemple, une règle qui fait descendre une restriction en dessous d'une jointure.

A noter que l'application de deux règles de transformation pour un PAT peut générer deux PAT identiques. Ceci est souvent le cas pour les règles qui modifient l'emplacement d'un opérateur. Par exemple, faire descendre une restriction en dessous d'une projection aura le même effet que faire monter une projection au dessus d'une restriction.

Le système contient un ensemble de règles de transformation définies pour son algèbre utilisée pour formuler les plans d'exécution. Dans un processus d'optimisation, pour un PAT donné, il existe toujours un ensemble de règles de transformation qui est un sous-ensemble de toutes les règles existantes pouvant être appliquées sur ce PAT. Nous pouvons dire que ces règles sont *valides* pour ce PAT qui contient des éléments (opérateurs) satisfaisant aux conditions d'application définies dans ces règles.

4.2.2 Poids d'une règle

Dans le processus d'optimisation, chaque étape consiste à choisir, parmi les règles valides pour le PAT courant, une ou plusieurs règles pour générer un nouveau PAT. Le choix de règle peut être aléatoire, c'est-à-dire, chaque règle a une probabilité égale d'être choisie ; ce choix peut aussi s'appuyer sur l'effet du changement en terme de coût d'exécution du PAT en appliquant la règle. Dans ce cas, il est intéressant de mesurer cet effet d'une manière numérique et statistique. Nous introduisons ainsi une notion *poids* pour chaque règle.

Définition du poids de règle

Le poids d'une règle de transformation équivalente décrit l'effet sur le changement de coût d'exécution d'un arbre algébrique avant et après l'application de la règle sur l'arbre algébrique. La définition du poids d'une règle est donnée comme suit :

Pour une règle de transformation équivalente, si les coûts d'exécution avant et après l'application de la règle sur un PAT sont respectivement $cost_{before}$ et $cost_{after}$, le *poids* de cette règle est *la moyenne* de $cost_{after}$ par rapport à $cost_{before}$ des n PAT sur lesquels la règle peut s'appliquer :

$$\frac{\sum_{i=1}^{n} \frac{cost_{after_i} - cost_{before_i}}{cost_{before_i}}}{n}$$

Cette valeur moyenne est calculée en se basant sur les historiques de calcul de coût sur les PAT pendant les processus d'optimisation. Quand la valeur est positive, cela signifie que l'application de la règle améliore le PAT en termes de coût d'exécution. L'application de la règle augmente le coût d'exécution du PAT quand la valeur est négative.

Comme le poids est une valeur en fonction de la moyenne des résultats de l'application de la règle dans tous les processus d'optimisation du système, sa valeur devient de plus en plus précise au fil du temps. Ceci pourrait éviter des cas *au hasard* où le poids des règles est mal estimé.

Pour les règles qui n'ont pas été appliquées, le poids est inconnu. Dans ce cas, le système a le choix entre les ignorer ou générer un poids aléatoire pour ces règles pour qu'elles aient une probabilité d'être choisies. Dans la suite du chapitre, nous allons voir l'utilisation des poids des règles.

4.3 Cadre générique d'optimisation

Nous proposons dans cette section, un cadre générique d'optimisation fournissant des outils (fonctions) nécessaires pour construire facilement un optimiser pour les systèmes de sources de données hétérogènes et réparties. Ces outils permettent aux développeurs du système d'implémenter n'importe quelle stratégie de recherche sans modifier la structure de l'optimiseur.

4.3.1 Description du cadre générique d'optimisation

Un optimiseur peut appliquer différentes stratégies de recherche pour piloter les processus d'optimisation. Bien qu'il existe un grand nombre d'algorithmes métaheuristiques différents, allant de la simple recherche locale à des algorithmes complexes de recherche globale, seulement certains d'entre eux sont pratiquement appliqués en tant que stratégies de recherche par la plupart des optimiseurs dans les systèmes de bases de données ou de médiation de données.

Les différentes stratégies de recherche appliquent des algorithmes de déroulement différents. Cependant, elles ont des points communs :

1. ces stratégies sont toutes basées sur l'estimation du coût, qui joue un rôle essentiel sur l'évolution du plan d'exécution pendant le déroulement de l'optimisation.
2. chaque stratégie consiste à faire évoluer le plan d'exécution initial en utilisant des règles de transformation ;
3. le déroulement de l'optimisation défini par chaque stratégie est toujours une série d'évolutions composée d'*étapes unitaires* et permet d'avancer dans l'étape suivante décidée par des critères de sélection ou de revenir dans une étape précédente.

Ayant identifié les points ci-dessus, nous pouvons définir un cadre générique qui fournit des fonctions utilisées par le déroulement des processus d'optimisation de requêtes. Ces fonctions sont unitaires et fournissent des traitements conçus pour l'objectif de prendre en considération les caractéristiques et les besoins de chaque stratégie. Des notions comme le poids de règles (cf. section 4.2.2) peuvent être appliquées.

Pour le déroulement des processus d'optimisation, chaque stratégie doit définir son propre algorithme de déroulement en composant les fonctions définies par le cadre d'optimisation. Le cadre d'optimisation consiste également en des normalisations de règles de transformation. Ces normalisations sont utilisées pour définir le critère de sélection de règle de chaque étape du déroulement de l'algorithme de recherche.

La figure 4.1 décrit le processus générique d'optimisation proposée. Il prend un arbre algébrique logique (LAT), l'optimise par une succession de transformations en PAT équivalents et de comparaisons de coût, suivant une stratégie de recherche choisie, et finalement produit un PAT optimal pour l'évaluation de la requête. Les entrées du cadre incluent nécessairement les informations définies par le GSD (schéma de données, localisation des sources, modèle de coût etc.).

Le processus générique pour toutes les stratégies de recherche peut ainsi être résumé brièvement :

1. Annoter un LAT avec les informations liées à l'exécution et au coût ;
2. Estimer le coût d'exécution du PAT à partir des annotations ;
3. Trouver une règle de transformation pour générer un nouveau PAT ;

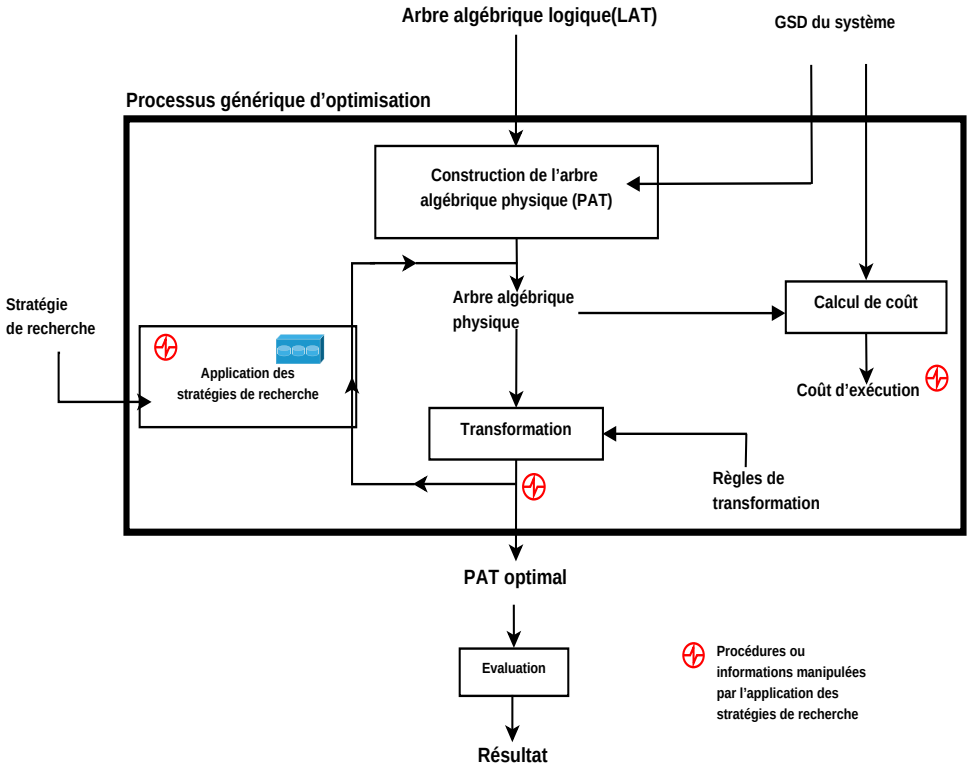

FIGURE 4.1 – Processus générique d'optimisation basé sur le coût

4. Concevoir un algorithme de déroulement défini par la stratégie de recherche ;
5. Définir les conditions d'arrêt pour que le processus d'optimisation se termine en considérant que le PAT optimal est retrouvé.

Pour satisfaire aux besoins ci-dessus, les fonctions unitaires doivent être définies pour traiter chaque étape du processus. Ces fonctions prennent les sorties disponibles issues des fonctions unitaires et fournissent en tant que résultats les nouvelles sorties qui serviront aux autres fonctions. Nous classifions ces fonctions en deux catégories :

- Les fonctions dont l'implémentation ne dépend pas des stratégies de recherche (tableau 4.1).
- Les fonctions dont l'implémentation dépend des stratégies de recherche appliquée par l'optimiseur (tableau 4.2).

Nous allons donner ensuite l'explication détaillée de chaque fonction et nous résumerons ces fonctions dans la section 4.3.8.

Nom fonction	Description	Paramètres	Sortie
annotate	Annoter un PAT avec des informations supplémentaire	LAT	PAT
calculateCost	Calculer le coût d'exécution d'un PAT	PAT	Coût
getRuleWeight	Obtenir le poids d'une règle de transformation	Règle de transformation	Poids
extractRules	Extraire toutes les règles de transformation valides pour le PAT	PAT	un ensemble de règles
applyRule	Transformer le PAT en appliquant une règle	PAT, Règle de transformation	un autre PAT

TABLE 4.1 – Fonctions unitaires indépendantes de la stratégie de recherches appliquée

Nom fonction	Description	Paramètres	Sortie
getOptimalPlan	Obtenir le plan optimal	LAT (initial)	PAT (optimal)
chooseRules	Choisir un ensemble de règles à appliquer sur le PAT	PAT	un ensemble de règles

TABLE 4.2 – Fonctions unitaires dont l'implémentation dépend de la stratégie de recherches appliquée

4.3.2 Fonctions *annotate* et *calculateCost*

L'annotation a pour objectif de marquer les informations complémentaires sur un LAT, pour aider à l'exécuter ou à l'optimiser. Dans le chapitre 3, nous avons montré qu'il y a différents types d'annotation pour des usages variés (droits d'accès, localisation, coût en temps d'exécution, cardinalité, coût en consommation électrique, etc.). Dans le cadre d'optimisation de PAT, les annotations de coût et de localisation des sources de données sont les plus importantes.

Dans le chapitre 3, nous avons montré comment un LAT peut être annoté pour obtenir un PAT, et comment le coût d'exécution du PAT peut être calculé. A noter que ces deux fonctions sont unitaires et **indépendantes** des stratégies de recherche. Pour un LAT donné, nous pouvons toujours, d'une manière générique, sous l'aide du GSD, l'annoter et calculer le coût d'exécution.

4.3.3 Fonction *getRuleWeight*

Dans la section 4.2.2, nous avons défini le poids de règle de transformation qui mesure l'effet de changement de coût d'exécution en appliquant sur la règle. Cette fonction a pour l'objectif d'obtenir le poids calculé par l'optimiseur. Pour l'implémentation de la fonction, l'optimiseur doit disposer d'une base de connaissance des résultats de l'application des règles afin de pouvoir calculer les poids en s'appuyant sur les statistiques.

L'implémentation est indépendante des stratégies de recherche et pourra être réutilisée par toutes les implémentations des différentes stratégie de recherche souhaitant prendre en considération le poids des règles.

4.3.4 Fonction *extractRules*

Cette fonction est appelée pour trouver toutes les règles de transformation **valides** (ou **applicables**) pour le PAT donné. Pour chaque règle de transformation existante, l'implémentation de la fonction cherche les éléments présents dans le PAT pour vérifier s'ils satisfont toutes les conditions d'application définies par la règle.

Dès que les règles de transformation sont prédéfinies par l'optimiseur, l'implémentation de cette fonction reste générique et peut être réutilisée par toutes les stratégies de cherche.

4.3.5 Fonction *applyRule*

Cette fonction est utilisée pour générer, à partir d'un PAT donné, un autre PAT en appliquant une règle de transformation équivalente. Chaque règle définit quelle partie du PAT doit être modifiée et comment elle est modifiée.

Cette fonction ne dépend pas de la stratégie de recherche. L'application de chaque stratégie de recherche consiste à appeler la fonction quand la transformation de PAT est demandée.

4.3.6 Fonction *getOptimalPlan*

Cette fonction est le point d'entrée du processus d'optimisation. Elle prend un LAT en entrée et fournit un PAT optimal prêt à être exécuté pour produire le résultat de la requête.

L'implémentation de cette fonction dépend fortement des stratégies de recherche. Chaque stratégie définit un algorithme dans cette fonction en composant des autres fonctions unitaires pour obtenir le plan optimal. Les conditions d'arrêt du déroulement du processus d'optimisation doivent également être définies dans cette fonction.

4.3.7 Fonction *chooseRules*

L'objectif de cette fonction est de choisir un ensemble de règles à appliquer sur un PAT donné. Le choix de règle de transformation est défini par la stratégie de recherche. Pourtant, différentes méthodes peuvent faire intervenir des facteurs significatifs sur les règles de transformation dans le cadre d'optimisation. Par exemple, comme mentionné ci-dessus, le poids des règles peut être pris en compte.

L'implémentation de cette fonction dépend ainsi des stratégies de recherche.

4.3.8 Synthèse des fonctions

La première catégorie contient les fonctions unitaires dont l'implémentation est indépendante du choix de stratégies de recherche. C'est-à-dire qu'une fois ces fonctions sont implémentées, nous ne refaisons pas le même travail quand l'optimiseur change de stratégie de recherche. Les cinq fonctions suivantes sont unitaires et indépendantes :

- *annotate*

- *calculateCost*
- *getRuleWeight*
- *extractRules*
- *applyRule*

Au contraire, la deuxième catégorie contient les deux fonctions dont l'implémentation dépend du choix de stratégies de recherche. *Pour chaque stratégie différente, nous devons uniquement effectuer l'implémentation de deux fonctions pour que cela corresponde à la définition de la stratégie* :

- *getOptimalPlan*
- *chooseRules*

Par conséquent, pour concevoir, dans notre cadre générique d'optimisation, un optimiseur intégrant différentes stratégies de recherche, il suffit de prédéfinir l'implémentation des fonctions indépendantes et de changer l'implémentation des fonctions dépendantes chaque fois que l'optimiseur choisit une stratégie différente. Notre cadre permet de soulager le travail de conception des optimiseurs qui répondent aux différents besoin d'optimisation dans différents systèmes traitant des requêtes variées.

Dans la suite du chapitre, nous allons montrer l'application de certains algorithmes de stratégie de rechercher dans notre cadre générique d'optimisation.

4.4 Application sur différentes stratégies de recherche

Dans cette section, nous montrons l'application de notre cadre générique d'optimisation sur six stratégies de recherche :

1. Algorithme exhaustif
2. Algorithme incrémental
3. Algorithme recuit simulé
4. Algorithme programmation dynamique
5. Algorithme génétique
6. Algorithme colonies de fourmis

Pour chaque stratégie de recherche, nous résumons brièvement son principe de fonctionnement (l'état de l'art de chaque stratégie se trouve dans le chapitre 2). Nous nous focalisons surtout sur l'implémentation de la stratégie en utilisant les fonctions unitaires définies dans notre cadre générique.

Pour illustrer l'implémentation des stratégies, nous utilisons un format pseudo-code du style des langages C/Java. En plus des opérations de contrôle de flux (IF, ELSE, FOR, WHILE etc.), nous utilisons certains types primitifs tels que *string, integer, float, etc.* Un format spécifique *COLLECT<A>* est également utilisé pour représenter une collection d'éléments de type A.

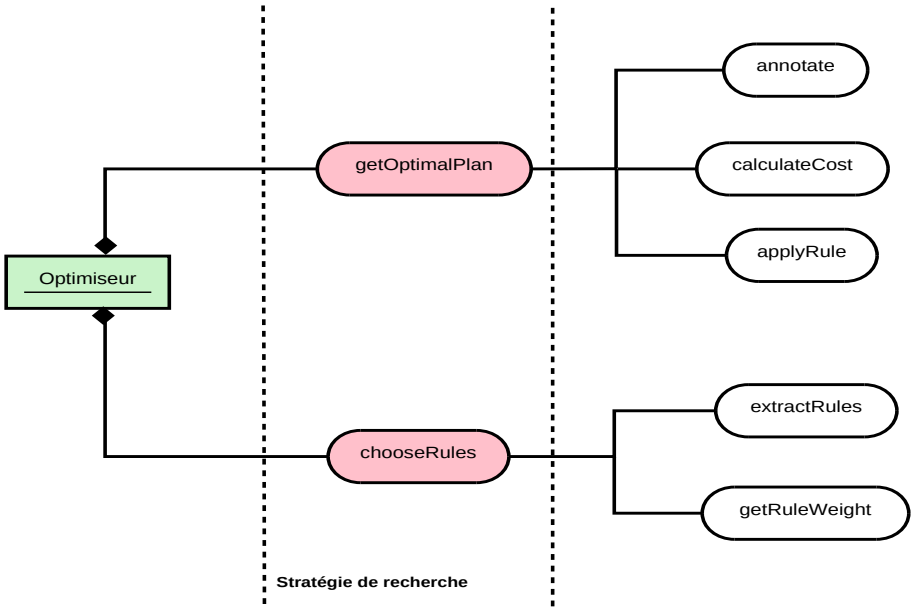

FIGURE 4.2 – Composition d'un optimiseur appliquant une stratégie de recherche avec des fonctions unitaires

Pour chaque stratégie de recherche, comme ce qui est résumé dans la section 4.3.8, nous donnons l'implémentation des deux fonctions dépendant de la stratégie : *getOptimalPlan* et *chooseRules*. La figure 4.2 illustre la composition d'un optimiseur appliquant une stratégie de recherche avec des fonctions unitaires. Nous pouvons voir que l'implémentation de la fonction *getOptimalPlan* est une composition algorithmique des trois fonctions unitaires : *annotate*, *calculateCost* et *applyRule*, tandis que l'implémentation de la fonctions *chooseRules* utilise les deux fonctions unitaires *extractRules* et *getRuleWeight*.

4.4.1 Algorithme exhaustif

L'algorithme exhaustif consiste à explorer tout l'espace de recherche et vérifier pour chaque étape de l'optimisation, toutes les règles valides pour le PAT à optimiser.

Pseudo-code

```
1   //Pseudo-code d'une application de l'algorithme exhaustif.
2
3   FUNCTION getOptimalPlan(LAT initialPlan) {
4           PAT p1 = annotate(initialPlan);
5           Cost costP1 = calculateCost(p1);
6
7           Cost minCost = costP1;
8           PAT optimalPAT = p1;
9
10          COLLECT<Rules> rulesToApply = chooseRules(p1);
11          FOR EACH rule IN rulesToApply {
12                  PAT newPlan = applyRule(p1, rule);
13                  PAT localOptimalPlan = getOptimalPlan(newPlan);
14                  Cost newPlanCost = calculateCost(localOptimalPlan);
15                  IF (newPlanCost < minCost) {
16                          minCost = newPlanCost;
17                          optimalPAT = localOptimalPlan;
18                  }
19          }
20          RETURN optimalPAT;
21  }
22
23  FUNCTION chooseRules(PAT p) {
24          COLLECT<Rule> validRules = extractRules(p);
25          RETURN validRules;
26  }
```

Dans cette implémentation, nous pouvons voir d'abord que la fonction *chooseRules* retourne toutes les règles applicables pour un PAT donné (lignes 23-26). Dans la fonction *getOptimalPlan*, nous utilisons une variable *minCost* (ligne 7) pour stocker le coût minimal qui sera mise à jour chaque fois qu'un meilleur PAT est trouvé (ligne 15-18). L'appel récursif de la fonction getOptimalPlan (ligne 13) permet d'explorer tous les PAT possibles en appliquant

toutes les règles valides. Nous pouvons voir que la fonction *getRuleWeight* n'est pas utilisée parce que toutes les règles valides sont testées indépendamment de leur poids.

4.4.2 Algorithme incrémental

L'algorithme incrémental consiste à essayer de réduire le coût du PAT à chaque étape de l'optimisation, en appliquant la meilleure règle de transformation parmi les règles applicables. Si l'application de la règle choisie fait augmenter le coût du PAT, le processus d'optimisation s'arrête et le *PAT optimal* pour la stratégie est annoncé trouvé.

Pseudo-code

```
1   //Pseudo-code d'une application de l'algorithme recuit simulé.
2
3   FUNCTION PAT getOptimalPlan(LAT initialPlan) {
4           PAT p1 = annotate(initialPlan);
5           Cost costP1 = calculateCost(p1);
6
7           Rule chosenRule = choosesRule(p1);
8           PAT newPlan = applyRule(p1, chosenRule);
9           Cost newPlanCost = calculateCost(newPlan);
10          IF (newPlanCost < costP1) {
11                  RETURN getOptimalPlan(p1);
12          } ELSE {
13                  RETURN p1;
14          }
15  }
16
17  FUNCTION Rule chooseRules(PAT p) {
18          COLLECT<Rule> validRules = extractRules(p);
19          Weight bestWeight = 0;
20          Rule bestRule;
21          FOR EACH rule IN validRules {
22                  Weight weight = getRuleWeight(rule);
23                  IF (weight > bestWeight) {
24                          bestRule = rule;
25                  }
26          }
27          RETURN result;
```

```
28  }
```

Le choix de la meilleure règle est basé sur le poids des règles applicables (lignes 17-28). La règle avec le poids le plus important est choisie. Dans le déroulement du processus d'optimisation, nous comparons uniquement le coût du nouveau PAT généré en appliquant la règle choisie avec le coût du PAT avant l'application (ligne 5 et 10). L'optimisation continue si le coût continue à diminuer (ligne 9) et s'arrête si nous ne pouvons plus diminuer le coût du PAT actuel (ligne 13).

4.4.3 Algorithme recuit simulé

Pour l'algorithme recuit simulé, partant d'une solution donnée, en la modifiant, nous en obtenons une seconde. Soit celle-ci améliore le critère que nous cherchons à optimiser (baisser le temps d'exécution), soit celle-ci le dégrade. Le point clé de l'algorithme est l'acceptation d'une *mauvaise* solution avec certaines conditions. Ceci permet alors d'explorer une plus grande partie de l'espace de recherche et tend à éviter de s'enfermer trop vite dans la recherche d'un optimum local.

Pseudo-code

```
1   //Pseudo-code d'une application de l'algorithme recuit simulé.
2
3   FUNCTION PAT getOptimalPlan(LAT initialPlan) {
4           int acceptableSteps = 3;
5           float acceptableWeightLost = 0.1;
6
7           PAT p1 = annotate(initialPlan);
8           Cost costP1 = calculateCost(p1);
9
10          PAT localOptimalPlan = p1;
11          PAT currentPlan = p1;
12
13          int usedSteps = 0;
14          WHILE (usedSteps <= acceptableSteps) {
15                  usedSteps = usedSteps + 1;
16                  Rule chosenRule = chooseRules(currentPlan);
17                  PAT newPlan = applyRule(p1, chosenRule);
18                  Cost newPlanCost = calculateCost(newPlan);
19                  IF (newPlanCost < costP1) {
```

```
20                          RETURN getOptimalPlan(p1);
21                  } ELSE IF(((costP1-newPlanCost)/costP1)
22                                          <= acceptableWeightLost) {
23                          currentPlan = newPlan;
24                  }
25          }
26          RETURN localOptimalPlan;
27  }
28
29  FUNCTION Rule chooseRules(PAT p) {
30          COLLECT<Rule> validRules = extractRules(p);
31          float bestWeight = 0;
32          Rule bestRule;
33          FOR EACH rule IN validRules {
34                  Weight weight = getRuleWeight(rule);
35                  IF (weight > bestWeight) {
36                          bestRule = rule;
37                  }
38          }
39          RETURN bestRule;
40  }
```

L'implémentation de la fonction *chooseRule* (lignes 29-40) est identique à celle de l'application de l'algorithme incrémental, en prenant en compte les poids des règles. En revanche, dans la fonction principale *getOptimalPlan*, nous définissions deux conditions d'arrêt (lignes 4-5) :

1. Nous ne pouvons essayer qu'au maximum trois étapes (ce qui est paramétrable) si nous obtenons des mauvaises solutions (PAT avec des coûts plus élevés).
2. L'augmentation de coût ne peut dépasser 10% du coût du PAT actuel.

Avec ces deux conditions d'arrêt, le processus se déroule dans une boucle WHILE ayant les deux conditions (lignes 14-25).

4.4.4 Algorithme programmation dynamique

L'algorithme programmation dynamique consiste à tout d'abord chercher les meilleures solutions pour les plans constitués uniquement des opérateurs unaires et puis ajouter étape par étape des opérateurs binaires. Quand chaque opérateur binaire est ajouté, la meilleure solution pour cet opérateur est cherchée pour avoir une meilleure combinaison.

Pseudo-code

```
1  //Pseudo-code d'une application de l'algorithme programmation dynamique
2
3  FUNCTION getOptimalPlan(LAT initialPlan) {
4          PAT p1 = annotate(initialPlan);
5          COLLECT<PAT> subUnaryPlans = findSubUnaryPlans(p1);
6          COLLECT<Operator> binaryOperators = findBinaryOperators(p1);
7
8          FOR EACH subPlan IN subUnaryPlans {
9                  //Utiliser une autre stratégie(exhaustif,
10          incrémental, etc.)
11                 subPlan = findOptimalSubPlan(subPlan);
12         }
13
14         PAT optimalPAT = p1;
15         FOR EACH operator IN binaryOperators {
16                 PAT newPlan = addOperator(p1, operator);
17                 Rule bestRuleOnBinaryOperators = chooseRules(newPla);
18                 optimalPAT = applyRule(bestRule);
19         }
20
21         RETURN optimalPAT;
22 }
23
24 FUNCTION chooseRules(PAT p) {
25         COLLECT<Rule> validRules = extractRules(p);
26         Weight bestWeight = 0;
27         Rule bestRule;
28         FOR EACH rule IN validRules {
29                 IF (isOnBinaryOperator(rule)) {
30                         Weight weight = getRuleWeight(rule);
31                         IF (weight > bestWeight) {
32                                 bestRule = rule;
33                         }
34                 }
35         }
36         RETURN bestRule;
37 }
```

La fonction *chooseRule* (lignes 23-36) est basée sur les poids. Mais il faut noter que les règles choisies ne concernent que des opérateurs binaires. C'est parce que dans la fonction *getOptimalPlan*, nous pouvons très facilement trouver une meilleure solution pour chaque sous-plan ne contenant que des opérateurs unaires, en utilisant une stratégie comme exhaustive (ligne 10). L'algorithme est divisé donc en trois parties :

1. Identifier les sous-plans *unaires* et les opérateurs binaires. (lignes 5-6)
2. Optimiser les sous-plans unaires. (lignes 8-11)
3. Ajouter les opérateurs binaires au sous-plans unaires optimal progressivement. (lignes 14-18).

Le processus s'arrête quand tous les opérateurs binaires sont ajoutés sur les sous-plans unaires.

4.4.5 Algorithme génétique

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte (ou que la solution est inconnue) pour le résoudre en un temps acceptable.

Pseudo-code

```
1   //Pseudo-code d'une application de l'algorithme génétique.
2
3   FUNCTION getOptimalPlan(LAT initialPlan) {
4           boolean stopCondition= false;
5           PAT p1 = annotate(initialPlan);
6           Cost minCost = calculateCost(p1);
7           PAT optimalPlan = p1;
8           COLLECT<Rule> chosenRules = chooseRules(optimalPlan);
9           COLLECT<PAT> newGeneration;
10          newGeneration.add(optimalPlan);
11          FOR EACH rule in chosenRules {
12                  PAT newPlan = applyRule(optimalPlan, rule);
13                  newGeneration.add(newPlan);
14          }
15
16          WHILE(stopCondition == false) {
17                  FOR EACH plan IN newGeneration {
18                          Cost planCost = calculateCost(plan);
```

```
19                          IF(planCost < minCost) {
20                                  minCost = planCost;
21                                  optimalPlan = getOptimalPlan(plan);
22                          } ELSE {
23                                  newGeneration.delete(plan);
24                          }
25                  }
26                  stopCondition = verifyStopCondition(optimalPlan);
27          }
28
29          RETURN optimalPlan;
30  }
31
32  FUNCTION chooseRules(PAT p) {
33          COLLECT<Rule> validRules = extractRules(p);
34          RETURN randomSubCollection(validRules);
35  }
```

Le déroulement de l'algorithme commence par choisir aléatoirement un sous-ensemble des règles applicables sur le PAT initial (ligne 8, 32-25). L'implémentation de la fonction *randomSubCollection* doit prendre en compte le poids des règles. Par exemple, nous pouvons choisir N règles dans la collection où N est inférieur à la cardinalité de la collection *validRules*).

Dans la fonction *getOptimalPlan*, en appliquant chaque règle dans ce sous-ensemble, des nouveaux PAT sont générés et forment une *génération* (lignes 9-14). Une condition d'arrêt doit être définie pour que le processus s'arrête à un certain moment. Dans la boucle WHILE (lignes 16-27), la génération évolue en améliorant des individus ayant de "bons gènes" (coûts diminués) ; et les *mauvais* individus sont éliminés de la génération (ligne 22-24). Le PAT optimal obtenu une fois que la stopCondtion devient vraie. L'implémentation de la fonction *verifyStopCondition* dépend des paramètres de la stratégie génétique, qui peut être, par exemple, le nombre maximal de générations (la recherche s'arrête lorsque ce nombre de générations sont vérifiées.)

4.4.6 Algorithme colonies de fourmis

D'une façon très générale, les algorithmes de colonies de fourmis sont considérés comme des métaheuristiques à population, où chaque solution est représentée par une fourmi se déplaçant sur l'espace de recherche. Les fourmis

marquent les meilleures solutions, et tiennent compte des marquages précédents pour optimiser leur recherche.

Pseudo-code

```
1   //Pseudo-code d'une application de l'algorithme colonies de fourmis.
2
3   FUNCTION getOptimalPlan(LAT initialPlan) {
4           boolean stopCondition = false;
5           PAT p1 = annotate(initialPlan);
6           Cost minCost = calculateCost(p1);
7           PAT localBestPlan = p1;
8           COLLECT<Rule> bestRules;
9
10          WHILE (stopCondition == false) {
11                  COLLECT<Rule> randomRules = chooseRules(localBestPlan);
12                  Rule localBestRule;
13                  FOR EACH rule IN randomRules {
14                          Plan newPlan = applyRule(p1, rule);
15                          Cost newCost = calculateCost(newPlan);
16                          IF(newCost < minCost) {
17                                  minCost = newCost;
18                                  localBestRule = rule;
19                                  localBestPlan = newPlan;
20                          }
21                  }
22                  bestRules.addWithAnOrder(localBestRule);
23                  stopCondition = verifyStopCondition(optimalPlan);
24          }
25
26          PAT optimalPlan = p1;
27          minCost = calculateCost(optimalPlan);
28          FOR EACH rule IN bestRules {
29                  PAT newPlan = applyRule(optimalPlan, rule);
30                  Cost newCost = calculateCost(newPlan);
31                  if (newCost < minCost) {
32                          optimalPlan = newPlan
33                  }
34          }
35
36          RETURN optimalPAT;
```

```
37  }
38
39  FUNCTION chooseRules(PAT p) {
40          COLLECT<Rule> validRules = extractRules(p);
41          RETURN randomSubCollection(validRules);
42  }
```

Dans cette implémentation de l'algorithme, similaire au déroulement de l'algorithme génétique, nous choisissons aléatoirement un sous-ensemble de règles applicables sur le plan initial (lignes 34-37). Ce choix peut être basé sur le poids de règles.

Après, dans la fonction *getOptimalPlan*, tant que la condition d'arrêt n'est pas satisfaite (la fonction *verifyStopCondition* dépend des paramètres de la stratégie, cf. section 4.4.5), de nouvelles meilleures règles dans ce sous-ensemble sont trouvées et ajoutées dans une liste (meilleur chemin) (lignes 39-42). Quand la recherche des meilleures règles est terminée, celles-ci sont appliquées par ordre décroissant sur le plan initial pour enfin obtenir le plan optimal.

4.4.7 Résumé de l'application des algorithmes

Dans cette section, nous avons montré comment utiliser notre cadre générique d'optimisation pour implémenter des optimiseurs utilisant six différentes stratégies de recherche. Nous avons vu que pour chaque stratégie, il suffit d'implémenter différemment **deux fonctions** (getOptimalPlan et chooseRules). Les autres fonctions unitaires sont implémentées une seule fois et réutilisées de la même manière dans les différentes stratégies.

Le tableau 4.3 illustre le nombre de lignes de pseudo codes à écrire pour chaque stratégie de recherche. Nous pouvons voir que l'application d'une nouvelle stratégie de recherche ne demande pas beaucoup de changement dans l'implémentation de l'optimiseur.

Stratégie de recherche	Nombre de lignes de pseudo-code
Exhaustif	26
Incrémental	28
Recuit simulé	40
Programmation dynamique	36
Génétique	35
Colonies de fourmis	42

TABLE 4.3 – Résumé de l'application des stratégies de recherche

4.5 Application sur différents critères d'optimisation

Pour un optimiseur de requêtes, il est possible d'avoir d'autres objectifs d'optimisation que la diminution du temps d'exécution. Un meilleur plan peut avoir une exécution avec un coût monétaire réduit, moins de consommation de ressources du système (mémoire, CPU, réseaux, etc.), ou même un critère défini par l'utilisateur du système. Le changement de critère d'optimisation demande ainsi un changement dans l'implémentation de l'optimiseur. Dans cette section, nous analysons l'application de notre cadre générique d'optimisation dans cette circonstance.

Quand le critère d'optimisation change, le modèle de coût utilisé pour mesurer le coût d'exécution des plans doit être remplacé par un nouveau modèle de coût décrivant ce nouveau critère. Ceci change la valeur des poids de règles de transformation car la valeur des coûts avant et après l'application de la règle changent (cf. section 4.2.2). Dans notre cadre d'optimisation, la fonction unitaire *getRuleWeight* fournit le poids d'une règle de transformation, son implémentation dépend des poids des règles, ainsi que du modèle de coût utilisé. En conséquence, l'implémentation de cette fonction est dépendante du critère d'optimisation choisi.

En revanche, si nous ne changeons pas la stratégie de recherche, l'implémentation des fonctions *getOptimalPlan* et *chooseRules* ne changerons pas car elles ne dépendent que de l'algorithme de stratégie. Quel que soit le critère d'optimisation choisi, quel que soit le modèle de coût utilisé, nous considérons toujours qu'un plan avec un coût diminué est un meilleur plan. Dans l'implémentation de ces deux fonctions (cf. section 4.4), la comparaison des

coûts des plan est utilisée, ce qui ne dépend pas du type de modèle de coût.

Pour résumer, si le développeur de l'optimiseur veut changer le critère d'optimisation, il suffit de modifier la fonction *getRuleWeight*, et l'implémentation de toutes les autres fonctions restent valides pour le nouveau critère d'optimisation.

4.6 Conclusion

Dans ce chapitre, nous avons proposé un cadre générique d'optimisation basé sur le modèle GSD proposé dans le chapitre 3, qui permet de construire facilement des optimiseurs de requêtes dans les systèmes intégrant des sources de données hétérogènes et réparties. Pour l'implémentation de l'optimiseur de requêtes dans de tels systèmes, notre cadre générique d'optimisation a les avantages suivants :

Réduction du temps de développement pour différentes stratégies de recherche La stratégie de recherche utilisée pour explorer l'espace de recherche est un paramètre pour le cadre d'optimisation. Cela signifie que l'implémentation du cadre ne dépend pas d'une stratégie spécifique. Le cadre fournit un certain nombre de fonctions unitaires pour réaliser les procédures utilisées par chaque stratégie. Comme ce qui est montré dans la section 4.4, l'implémentation d'un optimiseur utilisant une stratégie de recherche ne nécessite que l'implémentation spécifique des deux fonctions (*getOptimalPlan* et *chooseRules*.

Modèle de coût générique Le calcul de coût de chaque plan d'exécution candidat joue un rôle très important dans l'optimisation de requête. La performance de l'optimiseur dépend largement de la précision du modèle de coût utilisé. Notre cadre générique d'optimisation utilise un modèle de coût générique supporté dans le GSD permettant de décrire n'importe quel type d'information de coût pour toutes les granularités possibles du plan d'exécution, allant d'un simple opérateur à l'ensemble du plan.

Différents critères d'optimisation L'application du cadre générique d'optimisation sur différents critères d'optimisation utilisant différents types de modèle de coût ne demande pas un grand changement de l'implémentation de l'optimiseur (uniquement une fonction : *getRuleWeight*). Ceci justifie la flexibilité et la généricité de notre cadre.

Indépendance du langage pivot du médiateur Le processus d'optimisation est basé sur une algèbre générique issue du travail du TGV ([Travers *et al.* 2006b]). Les requêtes écrites en différents langages peuvent être optimisées avec notre cadre générique d'optimisation grâce au modèle semi-structuré générique. Dans le chapitre suivant, nous allons montrer les résultats expérimentaux de notre cadre générique d'optimisation en utilisant différents langages de requête (JPQL et SQL) pour formuler les requêtes envoyées au médiateur, afin de montrer la généricité du cadre d'une façon expérimentale.

Chapitre 5

Validation

5.1 Introduction

Dans le chapitre 3, nous avons montré comment calculer le coût d'exécution d'un arbre algébrique physique (PAT). Ce calcul est basé sur les informations de coût des opérateurs du PAT. Toutes les informations se trouvent dans la description des sources du système en utilisant notre modèle générique GSD.

Pour valider notre modèle GSD, nous effectuons l'exécution d'un ensemble de requêtes dans un système de médiation intégrant des sources hétérogènes réparties. Pour le PAT utilisé pour l'exécution de chaque requête, nous mesurons le coût d'exécution réel afin de le comparer avec le coût calculé avec le modèle GSD. Ceci montre la précision du modèle de coût décrit par le modèle GSD.

Dans le chapitre 4, un cadre générique d'optimisation a été proposé, afin de concevoir un optimiseur qui transforme le PAT initial en d'autres PAT éventuellement moins coûteux en termes du temps d'exécution. Ce cadre générique peut appliquer différentes stratégies de recherche qui ont toutes l'objectif de trouver un plan d'exécution optimal pour la requête donnée, en un temps acceptable pour explorer l'espace de recherche. Il est intéressant de vérifier, lors de l'application de chaque stratégie de recherche, comment ce cadre générique réagit en termes de temps d'optimisation.

La validation du cadre générique d'optimisation avec le modèle de coût intégré se fait au sein d'un système de médiation. Ce dernier devrait accepter les requêtes qui interrogent différentes sources de données distribuées et éventuellement hétérogènes. Il est aussi impératif que ce système possède un

optimiseur pouvant récupérer les informations de coût et des plans au cours d'optimisation, afin que nous puissions avoir des résultats explicites.

Organisation du chapitre

Dans ce chapitre, nous allons d'abord introduire le système de médiation utilisé pour notre validation dans la section 5.2 ; ensuite, nous allons montrer la validation de l'application du GSD pour calculer le coût d'exécution des plans dans la section 5.3. Dans la section 5.4, les résultats de l'application des différentes stratégies de recherche dans notre cadre générique d'optimisation seront présentés. Nous présentions également dans cette section l'efficacité de notre optimiseur en faisant varier les caractéristiques des requêtes. Nous présentons l'intégration de notre solution dans un produit d'intégration de données (DVS) de l'entreprise XCalia - Progress Software dans la section 5.5. Dans la section 5.6, une conclusion résume les résultats de validation d'un point de vue global.

5.2 Système de médiation pour validation

Dans cette section, nous montrons le système de médiation utilisé pour nos expérimentations.

5.2.1 Un système de médiation hétérogène réparti

Tous les systèmes de médiation ont trois aspects communs qui les caractérisent :

- *Hétérogénéité des données* : Le système intègre des sources de données de différents formats et différentes caractéristiques. Les adaptateurs doivent masquer cette hétérogénéité au médiateur.
- *Répartition des sources* : Les sources de données intégrées sont réparties sur différents sites, ce qui nécessite que le système (ou plus précisément, le médiateur) soit capable d'exécuter les opérations concernant plus d'une source de données, par exemple, les opérations binaires inter-sources.
- *Communication interne* : puisque l'hétérogénéité est masquée par les adaptateurs, l'environnement du médiateur doit garder l'uniformité de langage utilisé pour la communication entre les différents composants du système.

Notre système de médiation est un système hétérogène réparti et emploie un environnement semi-structuré. Nous présentons ci-dessous comment ce système réalise les trois aspects principaux d'un système de médiation.

Hétérogénéité

Le système intègre deux différents types de sources : Bases de données relationnelles et Services.

Les bases de données relationnelles incluent différentes bases commerciales telles que MySQL, Oracle, DB2, SQL Server, etc. Ces produits utilisent tous le modèle relationnel mais fournissent des fonctionnalités variées. Les données stockées dans ces bases sont de même format. L'hétérogénéité est interprétée également par le fait que des coûts d'exécution du même opérateur sur différentes bases peuvent avoir un écart considérable.

Les sources Service ne fournissent qu'un ensemble de fonctions, qui peuvent être appelées par les adaptateurs. L'adaptateur est chargé de transformer le sous-plan concernant la source Service en une série de fonctions disponibles définies par la source, et dans l'autre sens, il transforme le résultat fourni par la source Service dans un format commun au médiateur.

Le calcul de coût précis des sous-plans exécutés sur une source Service s'avère extrêmement difficile à cause du manque d'informations de coût. Mais comme le temps d'exécution d'un opérateur sur une source Service est en général plus long que sur une source relationnelle, il est important d'estimer le coût des sous-plans Service afin de mieux optimiser le plan global. En conséquence, la méthode historique est appliquée pour avoir une connaissance fiable sur le coût de ces sources Service.

Répartition

La répartition des sources de données augmente la difficulté de l'évaluation de requêtes. Les opérateurs binaires concernant deux sources différentes doivent s'exécuter sur le médiateur. Pour avoir une meilleure performance d'exécution, le médiateur dispose de plusieurs algorithmes pour exécuter ces opérateurs (cf. chapitre 4). L'optimiseur inclut aussi un modèle de coût précis pour les opérations exécutées par le médiateur, parce qu'il possède un contrôle complet du médiateur et peut se servir des informations de coût en cas de besoin.

Environnement semi-structuré

Toute hétérogénéité de données des sources est masquée par les adaptateurs. Au sein du médiateur, le langage de communication est sous format semi-structuré. La communication consiste en :

- la définition des méta-données et sa lecture par le parseur de requêtes ;
- la définition des fonctionnalités des sources de données (spécialement pour les sources Service) et sa lecture par le composant de construction du plan et l'optimiseur ;
- la définition et la communication de coût entre différents composants du système (cf. section 3.4.2) ;

L'uniformité de langage de communication facilite l'implémentation du système et réduit le temps de traitements sur le médiateur hors de l'exécution de requêtes.

5.2.2 Méta-données

Modèles de données

Les données sur les sources sont sous format objet-relationnel, mais comme nous l'avons expliqué dans le chapitre 3, ce modèle peut être transformé et exprimé dans le modèle semi-structuré. En effet, le médiateur utilise des fichiers XML pour exprimer ce modèle objet-relationnel en modèle semi-structuré. La figure 5.1 illustre le diagramme de classes qui présente les relations entre différentes tables/entités :

Méta-informations sur les sources de données

Le tableau 5.1 montre les méta-informations des sources de données sur lesquelles les données correspondant aux entités sont stockées (sauf *Person* qui est une entité abstraite). Les entités sont groupées par les sources de données sur lesquelles elles sont hébergées.

Il faut noter que deux entités peuvent être stockées dans le même type de source (ex. MySQL), elles sont considérés comme deux sources réparties par le système afin de tester les opérateurs inter-sites. Au total, nous avons six sources de données relationnelles de trois types de bases différentes (MySQL, Oracle et SQLServer), et nous avons trois sources de données de type Service.

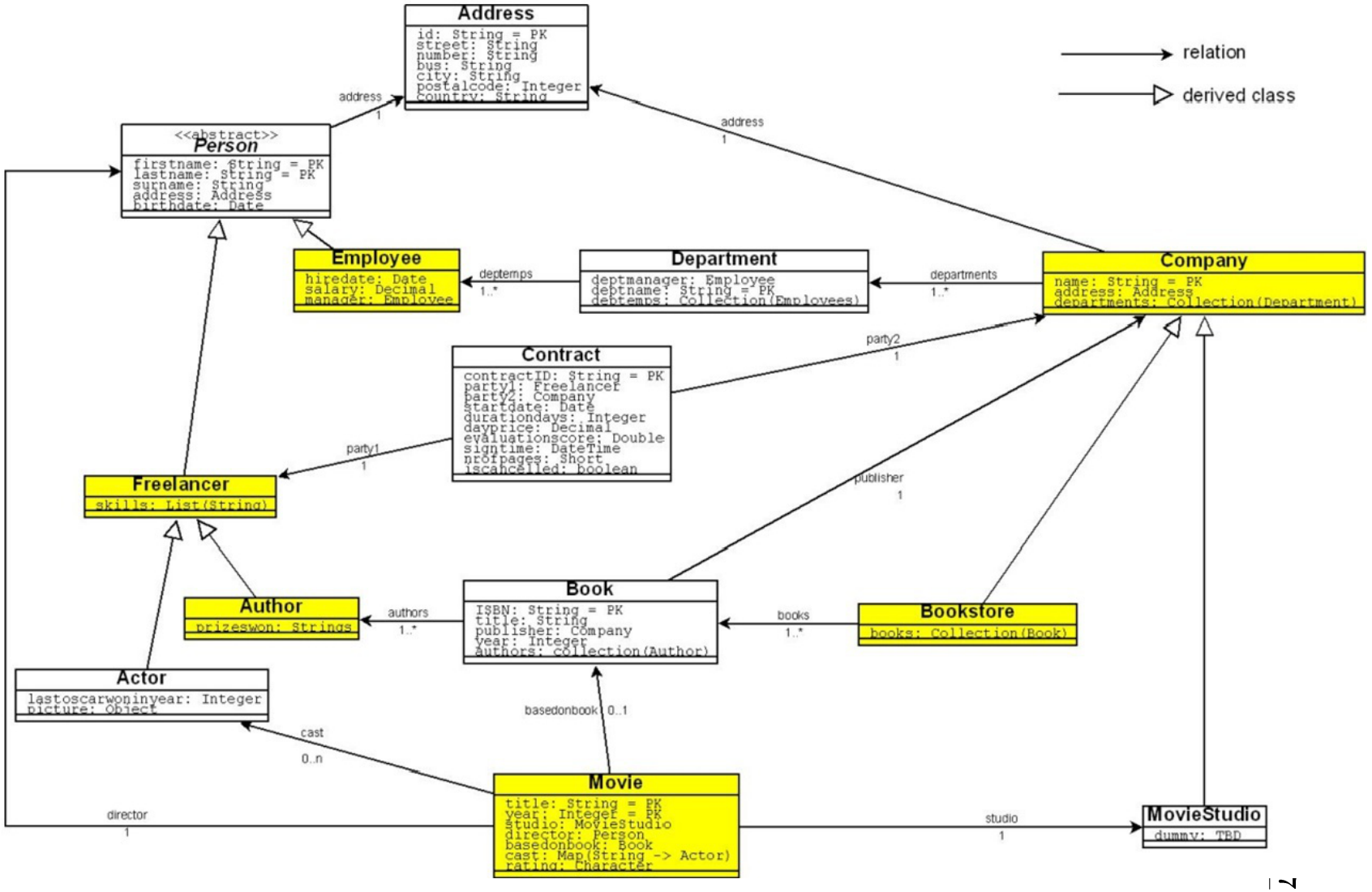

FIGURE 5.1 – Diagramme de classes des données pour validation

Entité	Localisation	type de source	Cardinalité	Modèle de coût
Employee	MySQL 1	Relationnel	5000	Calibration
Contract	MySQL 1	Relationnel	1000	Calibration
Freelancer	MySQL 2	Relationnel	2200	Calibration
Address	Oracle 1	Relationnel	10000	Calibration
MovieStudio	Oracle 1	Relationnel	20	Calibration
Book	Oracle 1	Relationnel	2000	Calibration
Company	Oracle 2	Relationnel	2000	Calibration
Department	Oracle 2	Relationnel	3500	Calibration
Bookstore	SQL Server 1	Relationnel	150	Calibration
Actor	Service 1	Service	Inconnue	Historique
Author	Service 2	Service	Inconnue	Historique
Movie	Service 3	Service	Inconnue	Historique

TABLE 5.1 – Méta-information sur les sources de données

À part les informations statistiques, nous pouvons également voir la méthode pour extraire les informations de coût. Pour les bases de données relationnelles, nous utilisons la méthode par calibration [Du *et al.* 1992] pour estimer le coût des sélections et des jointures sur ces sources. Pour les sources Service, puisqu'il manque des informations nécessaires (cardinalité, sélectivité etc), nous ne pouvons qu'estimer le coût par l'approche historique [Adali *et al.* 1996] pour avoir un coût basé sur l'exécution des opérations effectuées sur ces sources Service.

5.2.3 Requêtes pour validation

Afin de vérifier la précision du calcul de coût, nous utilisons un ensemble de requêtes couvrant les aspects de médiation : des opérations algébriques variées, opérations distribuées et coûteuses, de grands plans d'exécution à optimiser (jusqu'à 7 jointures inter-sites), etc. Le tableau 5.2 liste les requêtes expérimentées.

Bien qu'il existe des benchmarks utilisés pour tester l'efficacité des bases de données traitant des requêtes, ce qui nous intéresse dans cette validation, c'est l'intégration des différentes sources hétérogènes, y compris des sources spéciales comme Service. C'est pour cela nous avons choisi nous-même un ensemble de requêtes. De plus, ces requêtes représentent les requêtes les plus utilisées par les utilisateurs du système qui est un produit commercialisé (cf.

section 5.5).

Il faut noter que pour avoir plus de résultats expérimentaux en passant à l'échelle, certaines requêtes seront adaptées (par exemple, modifier le prédicat de restriction, ajouter le nombre de jointures inter-sites, etc.). Nous allons voir plus de descriptions de ces expériences dans les sections suivantes du chapitre.

Id Requête	Requête	Caractéristiques de la requête
Q1	SELECT b FROM Book b WHERE b.title = "The Name of the Rose"	1 sélection
Q2	SELECT b FROM Book b INNER JOIN b.author	1 jointure inter-site (exécutée sur un médiateur)
Q3	SELECT b FROM Book b INNER JOIN b.author WHERE b.title = "The Name of the Rose"	1 jointure inter-site et 1 sélection
Q4	SELECT b FROM Book b INNER JOIN b.author aut INNER JOIN aut.address addr INNER JOIN aut.skills sk	3 jointures inter-site
Q5	SELECT c FROM Company c INNER JOIN c.address addr INNER JOIN c.departments depts INNER JOIN depts.deptemps emps INNER JOIN emps.manager m	3 jointures inter-site et 1 jointure sur une source
Q6	SELECT e FROM Employee e INNER JOIN e.favoritebooks b WHERE b.year > 1980 ORDER BY e.lastname	1 jointure inter-site, 1 sélection et ordonnancement du résultat
Q7	SELECT DISTINCT comp FROM Company comp, Address addr WHERE comp.address = addr OR comp.name = 'Scottish Documentaries'	1 jointure inter-site et 1 sélection contenant OR (résolu par Union inter-site)
Q8	SELECT DISTINCT b FROM Book b INNER JOIN b.authors auth, Movie m INNER JOIN m."cast" act WHERE auth = act	1 jointure de type produit Cartésien et 1 sélection
Q9	SELECT firstname FROM Employee WHERE firstname = 'Fred' UNION SELECT firstname FROM Employee WHERE firstname = 'Arlette' EXCEPT SELECT firstname FROM Employee WHERE firstname = 'Sarah' UNION SELECT firstname FROM Employee WHERE firstname = 'Fred'	3 opérations ensemblistes inter-site
Q10	SELECT c FROM Company c INNER JOIN c.address addr INNER JOIN c.deptments depts INNER JOIN depts.emps emps INNER JOIN emps.favoritebooks b INNER JOIN b.authors aut INNER JOIN aut.address autAddr INNER JOIN aut.skilles sk INNER JOIN emps.manager m WHERE c.name = 'DataDirect' AND aut.firstname = 'Sharon'	7 jointures inter-site, 2 sélections et 1 jointure sur une source

TABLE 5.2 – Requêtes objet-relationnelles utilisées pour validation

5.3 Validation du calcul de coût par le modèle GSD

Le calcul de coût est précis si le coût calculé est proche du coût réel issu de l'exécution du plan. Notre modèle GSD permet d'intégrer diverses informations de coût pour calculer le coût des plans d'exécution. Dans cette section, nous montrons les résultats expérimentaux sur la comparaison entre le coût calculé et le coût réel.

5.3.1 Résultats expérimentaux par requêtes

Les figures 5.2 à 5.11 montrent pour chaque requête donnée dans la section 5.2.3, le coût calculé et le coût réel pour chaque plan d'exécution obtenu à l'issue des transformations appliquées par la stratégie de recherche exhaustive. Cette approche couvre tous les plans d'exécution possibles pour évaluer la requête et permet d'avoir une comparaison de coûts complète. Dans chaque schéma, l'axe X représente les plans d'exécution et l'axe Y le coût en millisecondes.

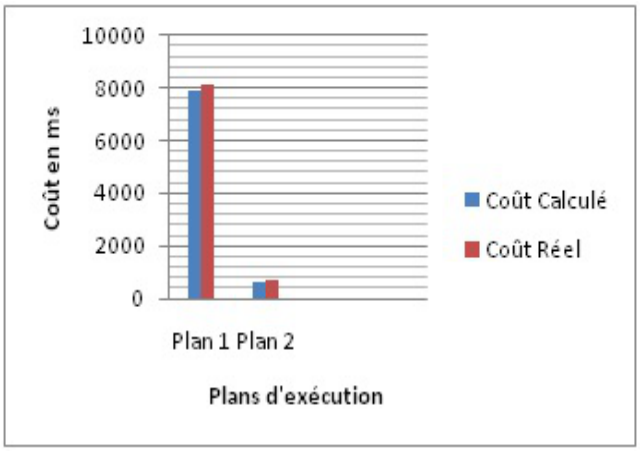

FIGURE 5.2 – Résultats de comparaison de coût pour la requête Q1

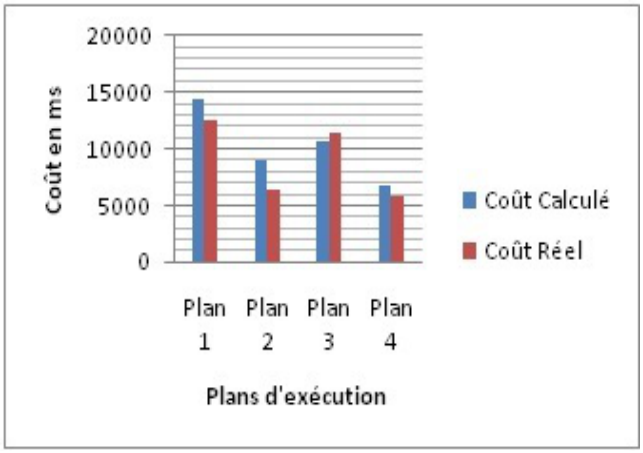

FIGURE 5.3 – Résultats de comparaison de coût pour la requête Q2

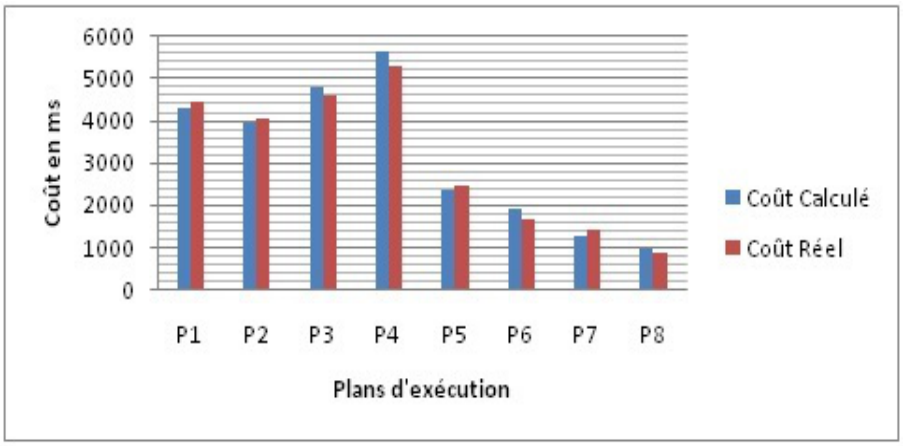

FIGURE 5.4 – Résultats de comparaison de coût pour la requête Q3

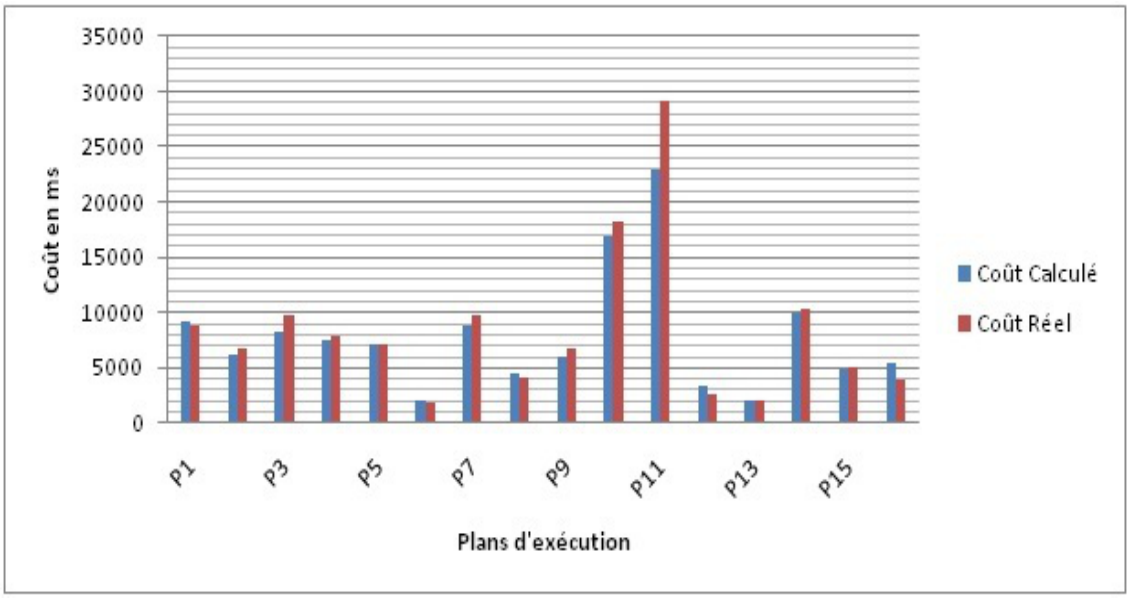

FIGURE 5.5 – Résultats de comparaison de coût pour la requête Q4

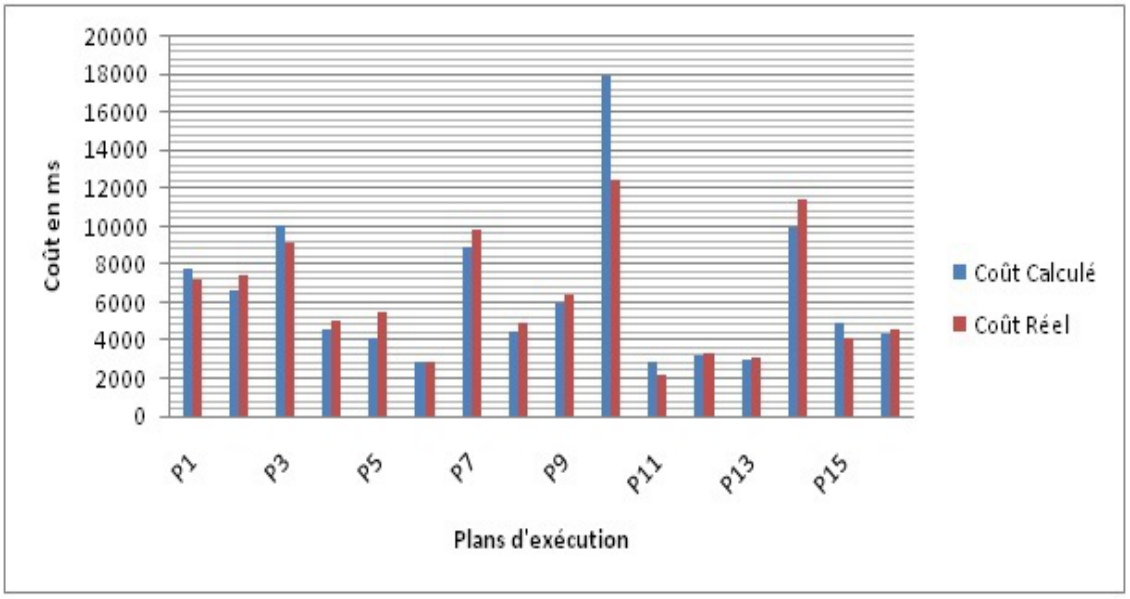

FIGURE 5.6 – Résultats de comparaison de coût pour la requête Q5

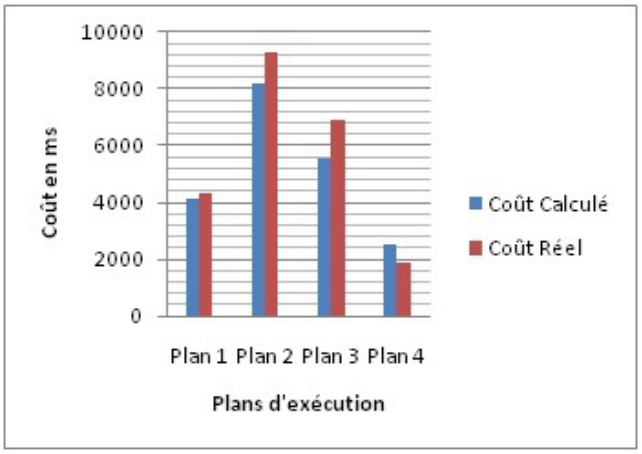

FIGURE 5.7 – Résultats de comparaison de coût pour la requête Q6

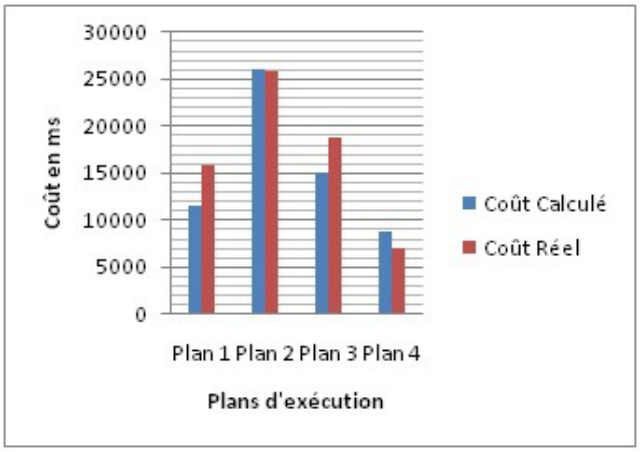

FIGURE 5.8 – Résultats de comparaison de coût pour la requête Q7

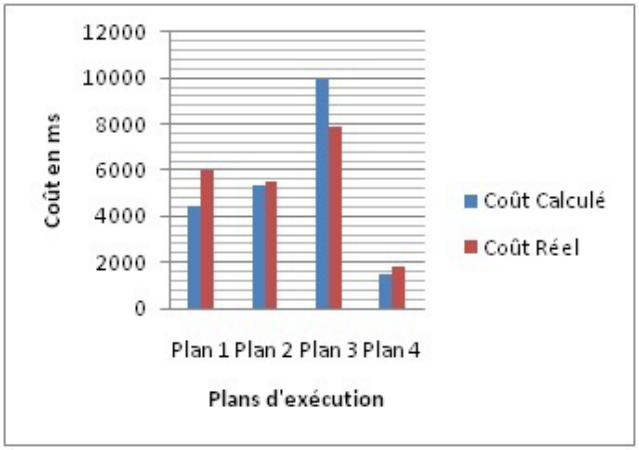

FIGURE 5.9 – Résultats de comparaison de coût pour la requête Q8

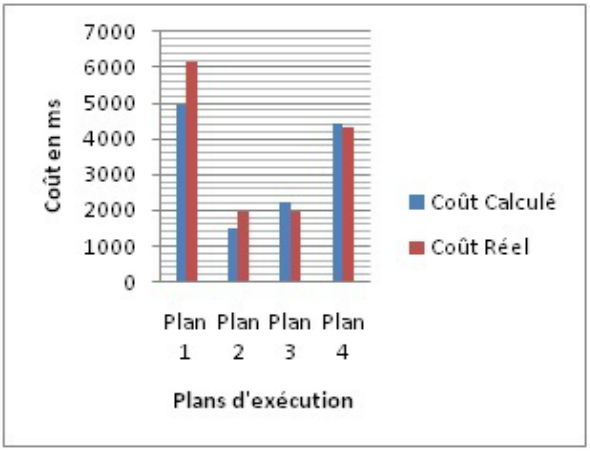

FIGURE 5.10 – Résultats de comparaison de coût pour la requête Q9

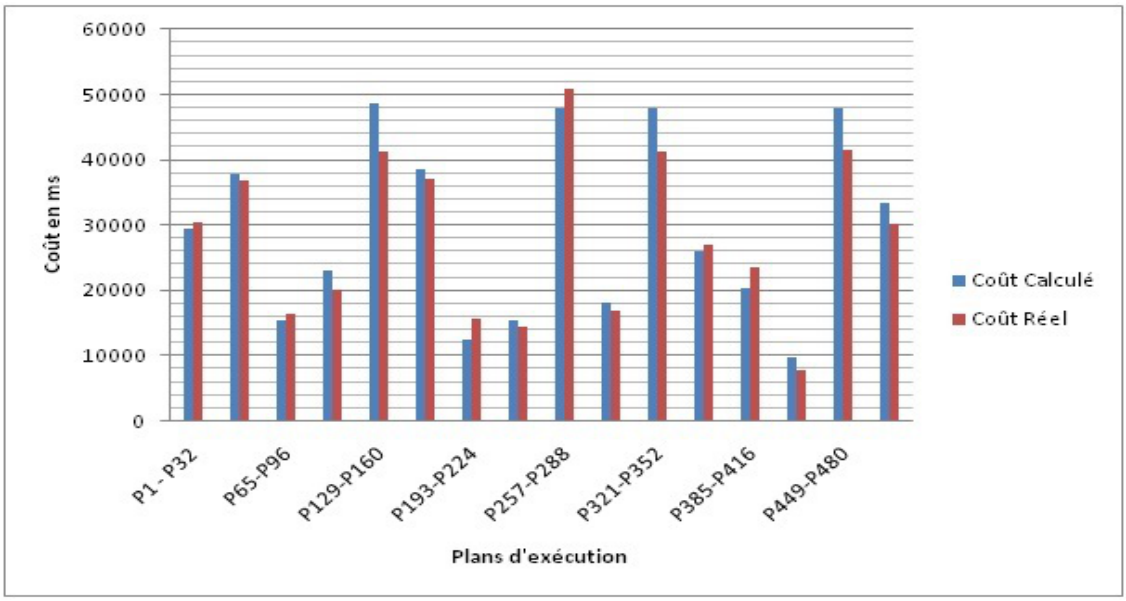

FIGURE 5.11 – Résultats de comparaison de coût pour la requête Q10

Note : L'optimiseur a essayé 512 plans d'exécution pour la requête Q10. Pour mieux présenter les résultats, les coûts sont regroupés par 32 plans et la valeur dans l'axe Y est la moyenne de coûts des 32 plans de chaque groupe.

Nous pouvons voir que pour la plupart des plans d'exécution, notre modèle de coût générique donne une estimation proche du coût réel (la précision de coût calculée sera montrée en détail dans la suite de cette section.). Pour la requête Q5, il existe une différence remarquable entre les 2 coûts pour le plan d'exécution P10, et c'est le même cas pour le plan P11 de la requête Q4. Dans ces plans, nous constatons que ce sont des plans faisant intervenir beaucoup de boucles imbriquées et par conséquent, ces plans utilisent beaucoup de mémoire. Nous allons faire une étude expérimentale ensuite dans la section 5.3.2

5.3.2 Précision de coût

Nous utilisons une notion **précision de coût** pour résumer les résultats expérimentaux sur le calcul de coût. La précision de coût peut être définie comme suit :

$$\text{Précision de coût} = 1 - \frac{|Cost_{real} - Cost_{calculated}|}{Cost_{real}}$$

Plus la précision est proche de 1, plus précis sera le modèle de coût.

Il n'est pas intéressant de mesurer la précision pour un seul plan d'exécution car la valeur obtenue n'est pas significative pour qualifier un modèle de coût. Cette précision est donc mesurée en se basant sur les résultats d'un ensemble de plans exécutés pour une requête. La figure 5.12 montre la précision de coût moyenne pour tous les plans de chaque requête.

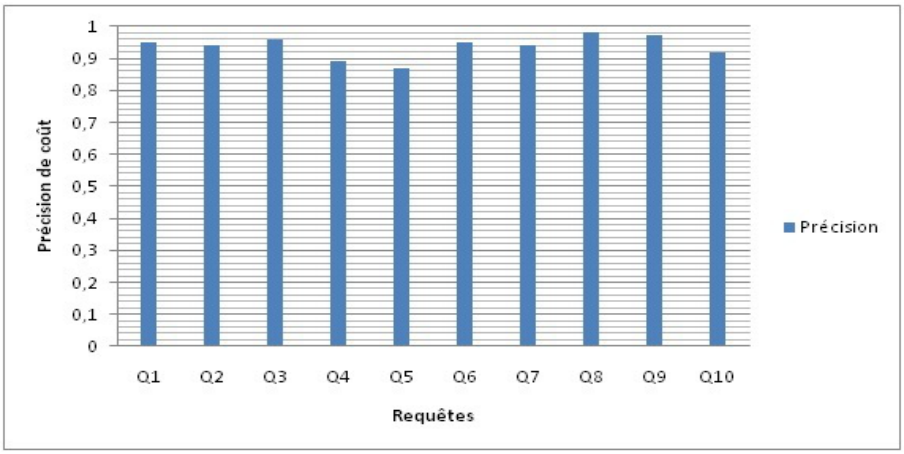

FIGURE 5.12 – Précision de coût moyenne pour les 10 requêtes expérimentées

Explication sur les divergences de coût constatées Lorsque la taille de données à traiter pour les jointures inter-sites augmente, l'impact de la mémoire devient plus important. Nous avons constaté que la divergence des coûts calculé/réel est importante pour les requêtes Q4 (0,89) et Q5 (0,86). Ceci est à cause d'un manque de précision de l'évolution de la charge de mémoire au cours de l'exécution des jointures inter-sources en appliquant l'algorithme boucle imbriquée. Pour expliquer cette divergence, nous avons fait varier la mémoire dédiée à l'exécution de requêtes, et la figure 5.13 montre l'évolution de la précision de coût en augmentant la mémoire utilisée. Nous pouvons effectivement constater que la précision est améliorée quand il y a plus de mémoire allouée pour l'exécution de requêtes.

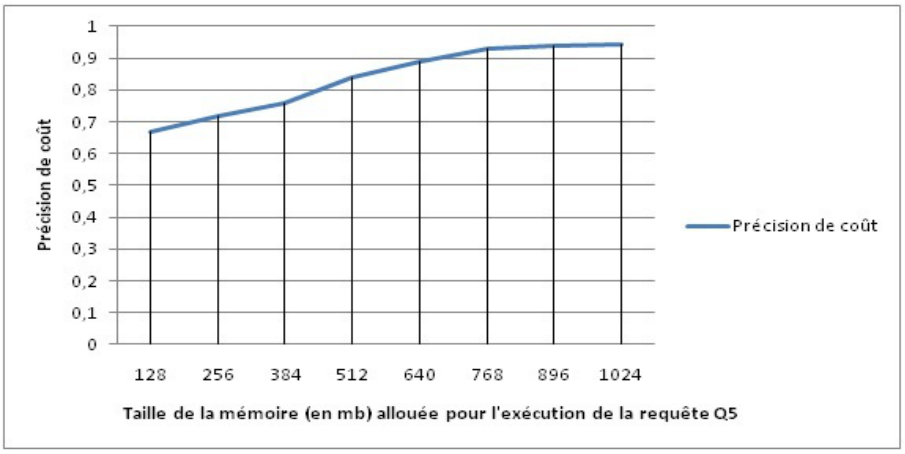

FIGURE 5.13 – Amélioration de précision de coût en fonction de la taille de mémoire allouée pour l'exécution de la requête Q5

5.3.3 Variation de caractéristiques de requêtes

Nous varions les caractéristiques cruciales des requêtes exécutées afin de vérifier si le modèle de coût est toujours assez précis pour estimer le temps d'exécution de la requête. Nous faisons varier deux paramètres : la cardinalité des sources interrogées et la sélectivité du prédicat de restriction.

Variation de cardinalité

Nous utilisons la requête suivante qui contient 3 jointures dont 2 sont entre différentes sources de données, et nous faisons varier la cardinalité de la table Company :

```
SELECT c FROM Company c INNER JOIN c.address addr
                            INNER JOIN c.departments depts
                                  INNER JOIN depts.emps emps
```

Pour le plan initial avec l'algorithme de jointure *boucle imbriquée* entre 2 sources de données, nous avons les résultats sur la comparaison entre coût calculé et coût réel dans la figure 5.14. L'axe X représente la cardinalité de la table Address et l'axe Y le coût du plan en milliseconde. Nous pouvons constater que la précision du coût reste stable au fil de l'évolution de la cardinalité.

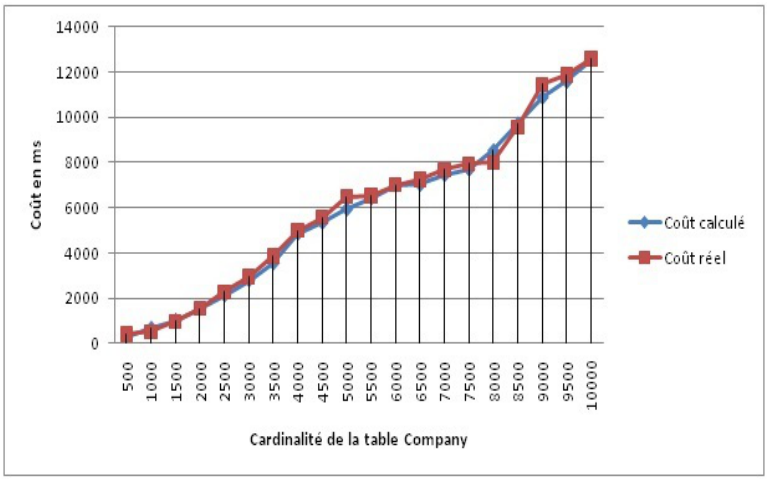

FIGURE 5.14 – Résultats de comparaison de coût en variant la cardinalité

Variation de sélectivité

Nous utilisons la requête suivante en faisant varier la sélectivité du prédicat.

```
SELECT c FROM Company c INNER JOIN c.address addr
INNER JOIN c.departments depts
         WHERE addr.city = 'Paris'
```

Si nous varions la sélectivité du prédicat addr.city = 'nom d'une ville', sachant que les sélectivités pour différentes valeurs de nom sont présentées dans le tableau 5.3, nous avons les résultats de comparaison de coûts dans la figure 5.15, avec l'axe X représentant la Sélectivité du prédicat et l'axe Y le coût en milliseconde. Nous pouvons constater que la précision de coût reste acceptable au fil de l'évolution de la sélectivité du prédicat.

Nom de ville	Sélectivité
Paris	0,001
Bruxelles	0,015
London	0,275
New York	0,0475
Boston	0,06
Hongkong	0,0795
Madrid	0,089
Houston	0,102
Washington	0,142
Los Angeles	0,189

TABLE 5.3 – Sélectivités du predicat addr.city = 'un nom d'une ville'

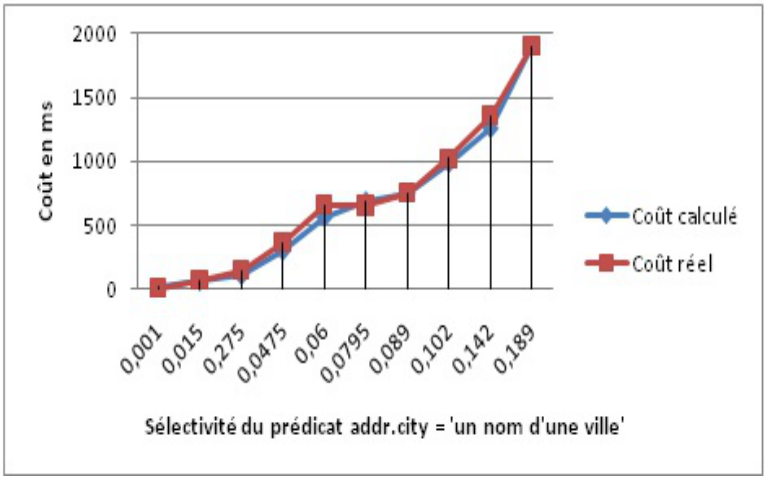

FIGURE 5.15 – Résultats de comparaison de coût en variant la sélectivité du prédicat

5.4 Validation du cadre générique d'optimisation

Notre cadre générique d'optimisation peut intégrer des stratégies de recherche issues des méta-heuristiques, sans modifier l'implémentation des fonctions de base. Pour montrer la flexibilité de notre cadre d'optimisation, nous avons implémenté l'application des stratégies de recherche mentionnées dans la section 4.4 et nous donnons dans cette section les résultats obtenus à l'issue de ces expérimentations.

Nous avons réalisé l'implémentation des 6 stratégies de recherche : Exhaustif, Incrémental, Recuit Simulé, Programmation dynamique, Génétique et Colonies de fourmis. Les détails de ces algorithmes sont présentés dans le chapitre 2 et le chapitre 4.

5.4.1 Expérimentation sur la requête Q10

Nous avons choisi la requête représentative Q10 pour montrer nos résultats expérimentaux. Cette requête a sept opérateurs inter-sites et demande une optimisation indispensable pour mieux être évaluée.

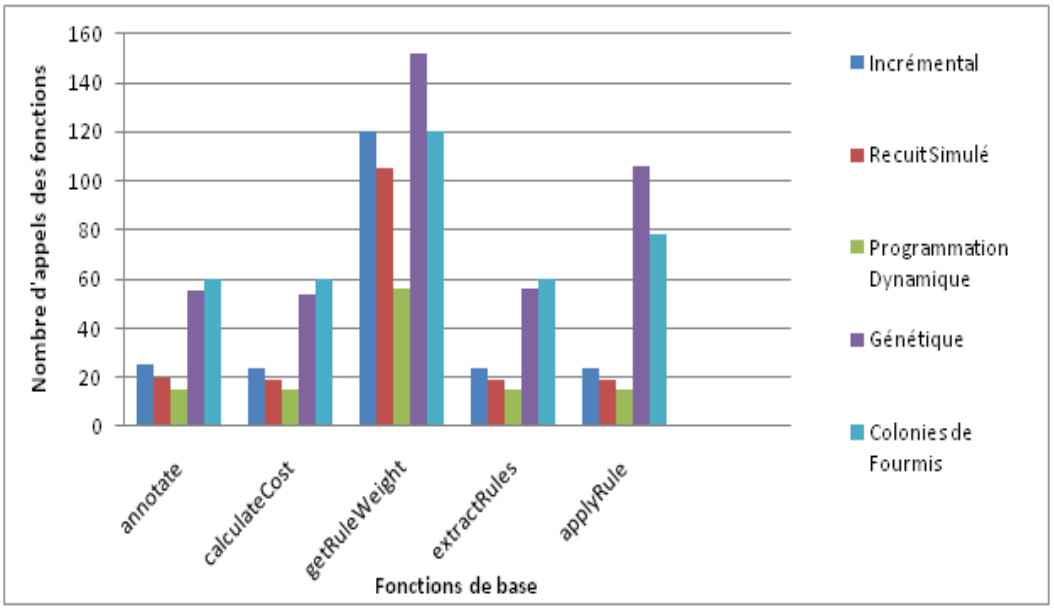

FIGURE 5.16 – Appels des fonctions pour différents stratégies pour Q10

La Figure 5.16 illustre pour chaque stratégie de recherche, le nombre d'appels des fonctions de base (cf. section 4.3) pour optimiser la requête Q10. La stratégie exhaustive a effectué des appels de fonctions de base plus de 1000 fois et donc il n'est pas intéressant de la comparer avec les autres stratégies.

Nous pouvons constater que les stratégies génétique et colonies de fourmis ont effectué plus d'appels que les autres stratégies parce qu'elles ont exploré plus de candidats dans l'espace de recherche. La fonction *getRuleWeight* est appelée beaucoup plus de fois que les autres fonctions, ceci est normal car dans la sélection des règles (fonction *chooseRules*), nous nous appuyons toujours sur les poids des règles.

Dans la suite de cette section, nous allons monter plus de résultats expérimentaux sur l'application des stratégies de recherche en augmentant le nombre des opérateurs inter-sites.

5.4.2 Variation de caractéristiques de requêtes

Afin de mieux connaître l'efficacité de l'optimiseur en appliquant les stratégies de recherche, nous faisons varier le nombre des opérateurs binaires inter-sites des requêtes pour voir l'évolution de performance des stratégies.

La figure 5.17 illustre les résultats de cette expérimentation. L'axe X représente le nombre d'opérateurs binaires présents dans le plan d'exécution et l'axe Y le temps d'optimisation (le temps utilisé pour trouver le plan optimal). La stratégie exhaustive donne un résultat exponentiel et donc n'apparaît pas dans le schéma de comparaison. Au contraire, pour toutes les autres stratégies, le temps d'optimisation reste linéaire par rapport à l'augmentation du nombre d'opérateurs binaires.

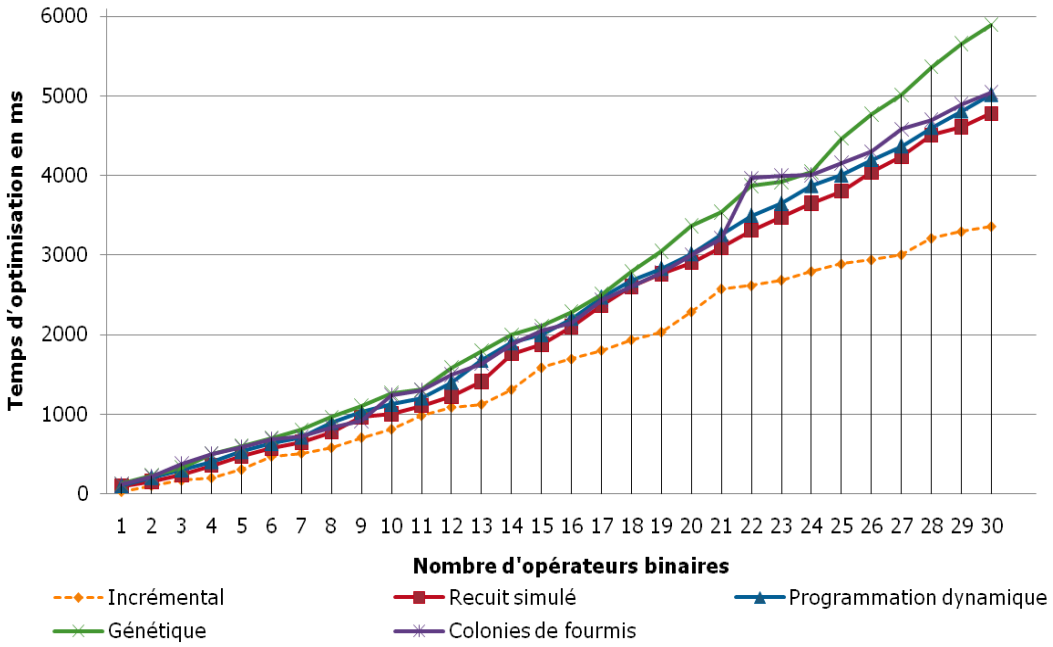

FIGURE 5.17 – Comparaison des stratégies en variant le nombre des opérations binaires

Si nous considérons que le plan optimal "idéal" est le plan avec le coût calculé minimal par la stratégie exhaustive, il est intéressant de comparer le coût de ce plan idéal avec le coût du plan obtenu par une stratégie de recherche autre que la stratégie exhaustive pour voir l'efficacité de cette stratégie.

La figure 5.18 montre cette comparaison pour les stratégies de recherche qui optimisent les requêtes avec dix, quinze et vingt opérateurs inter-sites. Avec les résultats dans la figure 5.17, nous pouvons constater que les stratégie génétique et colonies de fourmis utilisent en général plus de temps à optimiser un plan mais obtient un plan relativement meilleur que les autres stratégies. La stratégie programmation dynamique utilise un temps pour optimisation qui augmente linéairement lorsque le nombre de opérateurs inter-sites augmente, et obtiennent un plan "optimal" proche du plan idéal. La programmation dynamique est ainsi idéale pour optimiser les requêtes avec beaucoup d'opérateurs inter-sites.

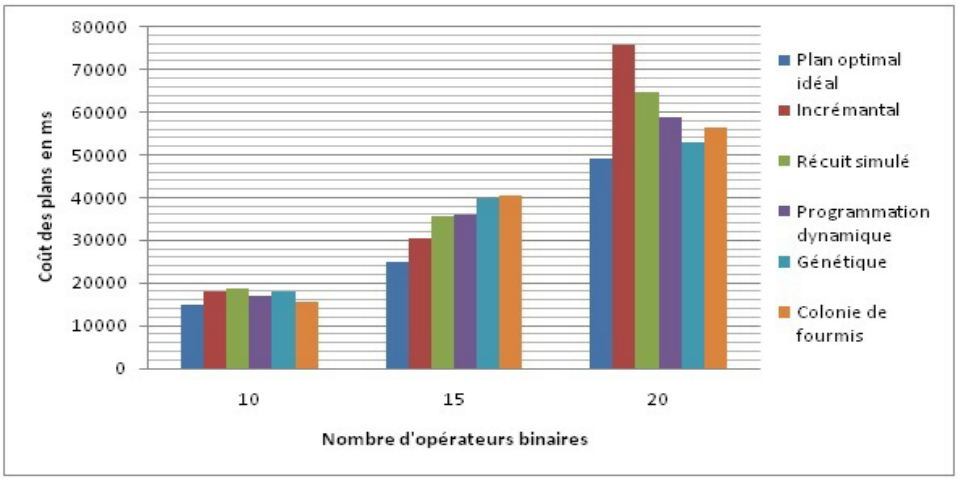

FIGURE 5.18 – Résultat de l'optimisation par les stratégies par rapport au plan " optimal " (calculé par la stratégie exhaustif)

5.5 Expérimentation dans l'entreprise

Dans le cadre d'une thèse CIFRE, l'approche proposée dans cette thèse a été mise en oeuvre et intégrée dans un produit d'intégration de données commercialisé par l'entreprise *XCalia - Progress Software Corporation*, et appelé **DVS** (*Data Virtualization Server*).

Système de médiation du DVS

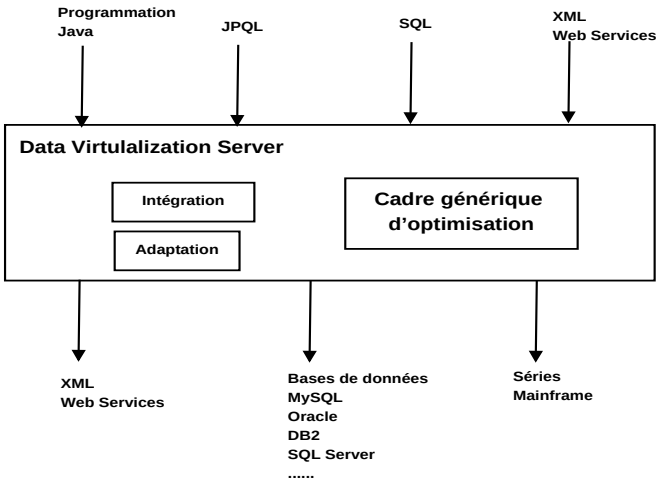

FIGURE 5.19 – Data Virtualization Server

DVS est un système de médiation qui intègre des sources de données réparties et hétérogènes. La figure 5.19 illustre l'architecture du DVS. Les sources de données intégrées incluent les principaux SGBD commercialisés : MySQL, Oracle, DB2, SQL Server etc, les Services Web et les séries mainframe. L'objectif du produit est de fournir au client une vue centralisée des données en masquant leur hétérogénéité et leur répartition. Les clients sont capables d'interroger ces sources de leurs propres façons : par la programmation Java, par un langage objet-relationnel JPQL ou relationnel SQL, et par des Services Web.

Vu la diversité et le volume très grand des données intégrées, ce système de médiation nécessite un traitement de requêtes très efficace, afin de répondre aux requêtes des clients en un temps raisonnable.

Application du cadre générique d'optimisation dans l'optimiseur DVS

L'auteur de cette thèse a été chargé de concevoir et développer l'optimiseur du produit DVS. L'implémentation de l'optimiseur a été une mise en oeuvre directe du cadre générique d'optimisation proposé dans cette thèse.

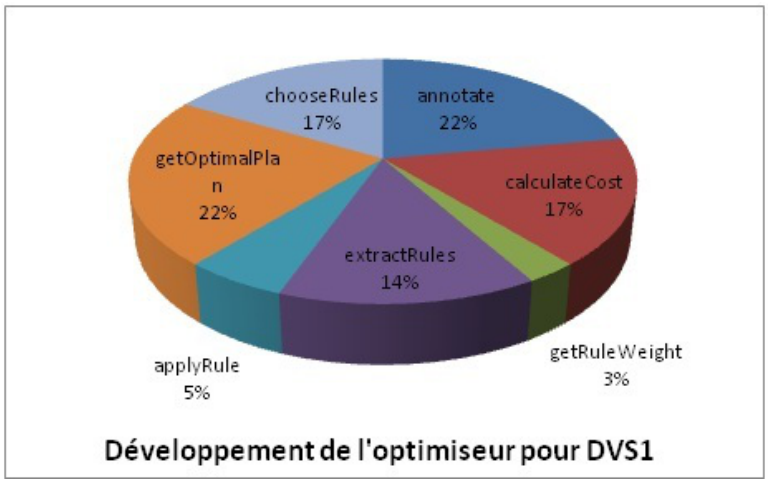

FIGURE 5.20 – Développement des fonctions de l'optimiseur pour DVS1

La figure 5.20 illustre la répartition du temps de développement de l'optimiseur pour la première version DVS. Dans la figure 5.21, qui illustre la répartition du temps pour la deuxième version de l'optimiseur pour DVS, nous

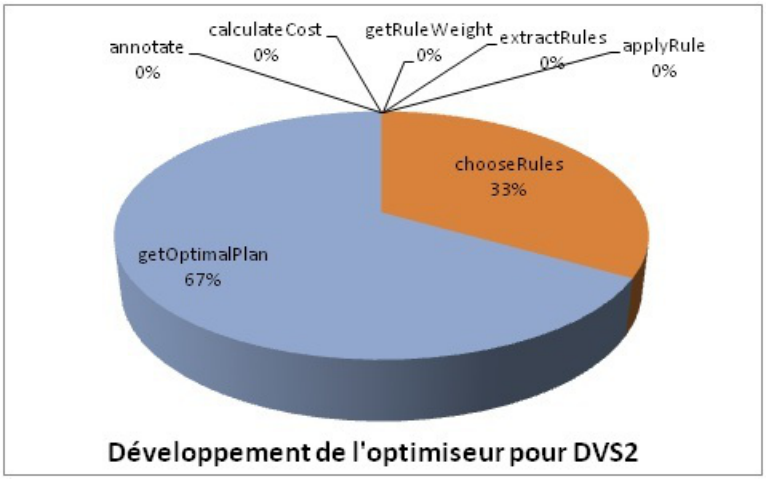

FIGURE 5.21 – Développement des fonctions de l'optimiseur pour DVS2

pouvons voir que le travail se focalise entièrement sur les fonctions *chooseRules* et *getOptimalPlan*. Cela est normal car l'implémentation de nouvelles stratégies de recherche ne nécessite que le travail dans ces deux fonctions (cf. section 4.3).

De plus, l'implémentation de la nouvelle version de l'optimiseur de DVS a utilisé deux fois moins de temps que la première version grâce à la réutilisation de l'implémentation des fonctions de base.

Performance de l'optimiseur DVS

Pour commercialiser le produit DVS, beaucoup de tests sur les benchmarks de requêtes ont été réalisés. Ces tests ont fait varier les caractéristiques cruciales des sources et des requêtes pour montrer la performance du système. Nous avons eu deux versions d'optimiseur : la première version contient un ensemble de règles de transformation restreint par rapport à la deuxième version ; de plus, plus de stratégies de recherche ont été implémentées pour la deuxième version,ce qui a permis de réduire largement le temps pour calculer le meilleur plan d'exécution.

Le tableau 5.4 illustre certaines améliorations en temps d'optimisation et temps d'exécution des requêtes sur les sources de données de petites et grandes cardinalités, avec le nombre de jointures allant de 3 jusqu'à 15. Nous pouvons voir que pour la première version de DVS, le temps de calcul du plan optimal et de l'exécution pour les grosses requêtes sur un grand volume

de données était inacceptable, mais que la deuxième version de DVS résout ces problèmes avec un meilleur optimiseur de requêtes. **Nous pouvons voir que pour la requête avec 15 jointures inter-site sur les sources de données de cardinalité 30000, le temps d'optimisation pour calculer le plan optimal est réduit de 10 minutes à 2 secondes (300 fois). Le temps d'exécution de la requête est réduit de 35 minutes à 1 minutes 15 seconde (28 fois).**

Cardinalité de chaque source	Nombre de jointures	Temps d'optimisation DVS 1	Temps d'exécution DVS 1	Temps d'optimisation DVS 2	Temps d'exécution DVS 2
1000	3	0.1s	0.2s	0.08s	0.2s
5000	5	38s	3mn	0.2s	10s
30000	15	10mn	35mn	2s	1mn15s

TABLE 5.4 – Comparaison de performance dans différentes versions de DVS

5.6 Conclusion

Dans ce chapitre, nous avons introduit notre système de médiation utilisé pour valider le calcul de coût à l'aide du GSD et du cadre générique d'optimisation. Nous avons illustré les résultats expérimentaux via l'exécution d'un benchmark de requêtes sur ce système de médiation. De plus, nous avons montré également les résultats d'application du cadre d'optimisation dans un produit commercialisé.

Pour le calcul de coût, les résultats de coût calculé/réel montre que la précision du modèle est satisfaisante pour bien estimer et prévoir le temps d'exécution des plans. Du plan le plus simple de la requête Q1 au plan le plus complexe de la requête Q10, l'estimation du coût par le modèle maintient sa précision. Il existe des cas particuliers où l'estimation est plus précise que les autres cas, à cause de la surcharge de la mémoire. Toutefois, cela n'empêche pas l'optimiseur de trouver toujours un plan efficace pour l'exécution de la requête, car les plans avec un coût relativement plus élevé que les autres sont rejetés par l'optimiseur.

Les résultats expérimentaux pour l'application des stratégies de recherche montrent la flexibilité de notre cadre générique d'optimisation : il peut intégrer ces stratégies sans modifier l'essentiel de l'optimiseur. Notre cadre générique d'optimisation a été mis en oeuvre et intégré dans un produit d'intégration de données appelé DVS, et commercialisé par l'entreprise Xcalia - Progress Software Corporation.

Cette expérimentation a validé notre approche d'une proposition d'un cadre générique d'optimisation. Pour des requêtes contenant beaucoup de jointures inter-site et interrogeant des sources de grand volume, le temps de calcul du plan optimal est de l'ordre de 2 secondes et le temps d'exécution du plan optimal est réduit de **28 fois** par rapport au plan initial non optimisé.

Chapitre 6

Conclusion

Le traitement de requêtes dans un système de médiation est une tâche complexe et nécessite une performance stable. La performance s'appuie sur les technologies d'optimisation de requêtes qui jouent un rôle très important parmi les trois étapes essentielles du traitement de requêtes : analyse, optimisation et évaluation. Les systèmes de médiation forment un environnement contenant des sources de données hétérogènes et réparties, ce qui ajoute plus de difficultés sur l'optimisation de requêtes interrogeant ces sources de données. Dans le chapitre 2, nous avons résumé les principaux travaux liés à l'optimisation de requêtes dans les systèmes de médiation. Afin de soulager les limitations dans ces solutions existantes et d'intégrer certaines technologies dans un processus générique, dans cette thèse, nous proposons un cadre générique d'optimisation de requêtes dans les systèmes de médiation intégrant des sources de données hétérogènes et réparties.

Dans ce chapitre, nous résumons les contributions de cette thèse en comparaison avec les solutions existantes dans la section 6.1 et nous présentons des perspectives de recherche à l'issue du travail réalisé dans cette thèse à court et long termes dans la section 6.2.

6.1 Contributions de la thèse

Dans cette thèse, nous proposons un cadre générique d'optimisation de requêtes basé sur le coût, basé sur un modèle générique de description des sources (GSD), dans les systèmes de médiation intégrant des sources de données hétérogènes et réparties. Ce cadre permet d'intégrer différentes technologies sur l'optimisation de requêtes et fournit une possibilité d'implémentation facile de l'optimiseur pour tout type de systèmes de médiation. Nous résu-

mons ci-dessous les contributions menées par cette thèse en les comparant avec les travaux existants.

Description générique des sources

Nous avons proposé un modèle générique (GSD) pour la description des sources de données. Ce modèle permet de décrire tous les types d'informations pour tous les types de systèmes gérant des sources. Nous avons défini les différents composants du GSD d'une manière formelle. Ces composants permettent de décrire les informations de source et de structurer ces informations.

Le modèle GSD utilise le format de graphe pour décrire les schémas, ceci résout l'hétérogénéité des sources parce que tout peut être représenté par des graphes. Les opérateurs sont associés à des opérandes (paramètres) référençant aux schémas. Les opérateurs d'une source ou inter-sites sont distingués. La description des informations supplémentaires est classifiée par couches d'annotation, ce qui facilite l'extraction des informations pertinentes. Les schémas et les opérateurs peuvent être annotés selon n'importe granularité, ce qui permet de décrire les informations de n'importe précision.

Avec ce modèle GSD, nous pouvons décrire des informations de coût. Avec la description de coût, nous pouvons calculer le coût des plans d'exécution. Ceci est utile dans le cadre de l'optimisation de requêtes.

Cadre générique d'optimisation

Nous avons vue qu'il existe différentes technologies liées à l'optimisation de requêtes dans un système de médiation. Chaque technologie est dédiée pour résoudre une certaine problématique d'optimisation : stratégies de recherche, espace de recherche formé grâce aux règles de transformation, prise en compte des capacités fonctionnelles réduites des sources.

Notre cadre générique fournit un certain nombre de fonctions unitaires pour construire des optimiseurs appliquant différentes stratégies de recherche. Les fonctions unitaires prédéfinies par le cadre permettent de manipuler les règles de transformation afin de faire évoluer le plan d'exécution d'une manière définie par la stratégie de recherche. Pour différentes stratégies de recherche, il suffit de développer l'algorithme qui pilote l'évolution du plan d'exécution (uniquement deux fonctions getOptimalPlan et chooseRules) en réutilisant les mêmes fonctions unitaires.

Validation de la solution proposée

Nous avons réalisé les expérimentations de notre cadre générique d'optimisation sur un système de médiation intégrant des sources de données hétérogènes et réparties. Nous avons exécuté un benchmark de requêtes sur ce système et nous avons mesuré les informations concernant l'optimisation et l'exécution des requêtes. Nous avons montré qu'avec le modèle GSD proposé, le coût d'exécution des plans d'exécution peut être calculé par l'optimiseur avec une précision satisfaisante.

Pour le cadre générique d'optimisation, nous avons expérimenté sur le même benchmark de requêtes en paramétrant le cadre d'optimisation avec différentes stratégies de recherche. Nous avons montré qu'avec chaque stratégie de recherche l'optimiseur a réussi à optimiser les requêtes dans un délai raisonnable, ce qui montre la généralité de notre cadre d'optimisation.

Dans le cadre de cette thèse CIFRE, notre cadre générique d'optimisation a été mis en oeuvre et intégré dans un produit d'intégration de données commercialisé (DVS) par l'entreprise Xcalia - Progress Software Corporation. Pour des requêtes contenant beaucoup de jointures inter-site et interrogeant des sources de grand volume, le temps de calcul du plan optimal est de l'ordre de 2 secondes et le temps d'exécution du plan optimal est réduit de **28 fois** par rapport au plan initial non optimisé.Cette expérimentation a validé notre approche.

6.2 Perspectives de recherche

Nous pensons étendre le travail réalisé dans cette thèse dans les directions suivantes :

Estimation de coût avec des caches

Dans la section 3.4, nous avons proposé un processus de calcul de coût d'un arbre algébrique physique. Pendant l'optimisation d'une requête, l'espace de recherche contenant un certain nombre d'arbres algébriques candidats est exploré par une stratégie de recherche. Le processus de calcul de coût est appliqué à chaque arbre algébrique. Il est possible que la stratégie de recherche implique que le même arbre algébrique soit considéré plusieurs fois. Pour éviter de calculer le coût du même arbre algébrique, il est intéressant de réutiliser les informations des coûts déjà rencontrées.

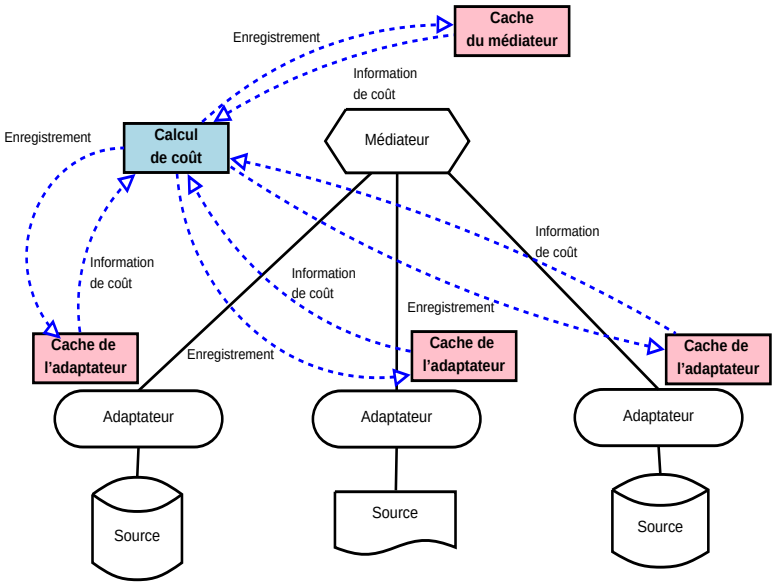

FIGURE 6.1 – Module d'estimation de coût dynamique

Pour ce faire, nous proposons d'utiliser un système de cache à deux niveaux pour enregistrer les informations des coûts des arbres (ou sous-arbres) déjà calculés (figure 6.1). Nous aurons un cache au niveau du médiateur et un cache pour chaque adaptateur de source de données. Le cache du médiateur enregistre les coûts calculés des arbres algébriques entiers correspondants aux requêtes. Les caches des adaptateurs enregistrent les coûts des sous-arbres exécutés par chaque source de données. Quand le coût d'un arbre algébrique doit être calculé, nous cherchons d'abord dans le cache du médiateur pour voir s'il existe déjà le coût du même arbre. Si c'est le cas, nous réutilisons directement le coût ; dans le cas contraire, nous essayons de trouver les coûts des sous-arbres dans les caches des adaptateurs. Si aucune information n'est trouvée, le coût sera calculé et enregistré dans les caches correspondants avec l'arbre (ou sous-arbres). Cette extension est envisageable à court terme.

Optimisation de requêtes dans Cloud Computing

Le Cloud Computing [Divyakant Agrawal et Abbadi 2010] est un concept qui consiste à déporter sur des serveurs distants des traitements informatiques traditionnellement localisés sur des serveurs locaux ou sur le poste Client de l'utilisateur. Dans ce contexte, les utilisateurs ou les entreprises ne sont plus gérants de leurs serveurs informatiques, mais peuvent ainsi accéder de manière évolutive à de nombreux services en ligne sans avoir à gérer l'infrastructure sous-jacente, souvent complexe. Les applications et les don-

nées ne se trouvent plus sur l'ordinateur local, mais dans un nuage (cloud) composé d'un certain nombre de serveurs distants interconnectés au moyen d'une excellente bande passante indispensable à la fluidité du système.

Un aspect important dans ce contexte est que nous considérons les ressources du système (CPU, mémoire, réseaux, etc.) *infinies* : le temps de traitement des opérations est rapide et n'a jamais de problème de performance, les réseaux n'ont jamais de congestion, etc. En conséquence, le critère d'optimisation n'est plus le temps d'exécution. En revanche, ces ressources sont payantes et coûteuses, ce qui implique que désormais l'optimisation a pour objectif de réduire le coût monétaire des requêtes. Dans la section 4.5, nous avons montré qu'en appliquant l'approche proposée dans cette thèse, il suffira de

- ajouter une couche d'annotation de coût monétaire dans la description de sources par GSD ;
- adapter l'implémentation de la seule fonction *getRuleWeight* en prenant en compte le nouveau modèle de coût monétaire pour concevoir un nouvel optimiseur dans le contexte de Cloud Computing.

Ainsi, aucune modification du modèle de description de sources et du cadre d'optimisation n'est nécessaire. Ce travail est donc envisageable à très court terme.

Modèles de coût pour les flux de données

Un flux de données est une séquence infinie d'éléments générés de façon continue à un rythme rapide. Pour concevoir un système efficace de traitement des flux de données, nous devons modifier la plupart des composants d'un SGBD traditionnel [Golab et Özsu 2003]. Le traitement des requêtes sur les flux de données est continu, le modèle de coût traditionnel, basé sur la sélectivité de prédicat et la cardinalité des résultats intermédiaires, ne s'applique plus à l'environnement des flux de données infinis. Différents modèles de coût ont été proposés dans [B. Babcock 2003] [S. Madden 2003] [Viglas et Naughton. 2002] sur la charge de mémoire, le taux de génération des résultats intermédiaires et la consommation d'énergie.

Nous pensons travailler sur l'intégration des données hétérogènes, y compris les flux de données. Il serait donc intéressant d'intégrer dans notre modèle de coût générique les modèles de coût pour les flux de données afin de mieux estimer le coût des requêtes et de les optimiser. Nous pourrions également

travailler à la proposition d'un nouveau modèle de coût pour les flux de données.

Intégration des flux de données en temps réel

Les systèmes informatiques en temps réel sont différents des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude du résultat. Le système doit strictement délivrer des résultats exacts dans des délais imposés. Le développement de systèmes temps réel nécessite donc que chacun des éléments du système soit lui-même temps réel, c'est-à-dire permette de prendre en compte des contraintes temporelles. La précision du modèle de coût dans ce type de systèmes devient donc très importante. Des travaux liés à l'intégration des données en temps réel ont été réalisés dans [Adelberg *et al.* 1996] [Carney *et al.* 2002]. Plus récemment, dans [Nehme *et al.* 2009], une solution pour le traitement des flux de données en temps réel a été proposée en utilisant un système de méta-données de *Tag*.

Pourtant, dans les environnements hétérogènes répartis, la conception d'un système en temps réel demeure une problématique majeure à résoudre. Nous pourrions étudier ce domaine pour établir un modèle de coût précis pour satisfaire aux besoins d'un système en temps réel dans le cadre de l'intégration de données hétérogènes réparties avec une prévision exacte en assurant un temps d'exécution réduit. Afin d'étendre notre cadre générique d'optimisation pour traiter les requêtes en temps réel, il suffirait d'y ajouter des composants qui gèrent le traitement des requêtes en temps réel. Ces composants permettront de prendre la décision de l'évaluation de requêtes en vérifiant si le coût d'exécution satisfait aux délais envisagés.

Modèle de coût pour les opérateurs FPGA

Il existe des limitations aux architectures informatiques existantes pour l'intégration des données : charge de mémoire et de *CPU*, consommation d'énergie, congestion de réseaux, etc. Une solution possible pour surmonter ces limitations est d'utiliser des matériels configurables (ex. des circuits logiques) pour effectuer le traitement des données *physiquement*, à des débits très élevés et avec une latence extrêmement faible. Dans [Mueller *et al.* 2009], un compilateur *Glacier* a été proposé, qui permet de traiter des opérations d'une requête envoyée aux bases de données avec des circuits *FPGA* (*Field-Programmable Gate Arrays*). Cette solution matérielle surpasse de manière

significative des solutions traditionnelles logicielles connues en termes d'économie de consommation des ressources.

Nous pensons travailler sur l'intégration des données en profitant des opérateurs FPGA pour accélérer les accès aux données. Il serait donc intéressant d'établir un modèle de coût pour les traitements des opérateurs FPGA, ce qui manque dans les travaux existants [René Müller et Alonso 2009]. Nous pourrons alors intégrer ce modèle de coût spécifique dans notre modèle générique proposé dans la section 3.4. Il suffit de définir les formules de coût et les statistiques avec le langage générique descriptif (cf. section 3.4.2).

Optimisation multi-critères de requêtes

Dans la section 2.3, nous avons montré que l'objectif de l'optimisation de requêtes peut varier selon différents systèmes d'intégration de données : temps de réponse, consommation de l'énergie, etc. L'optimisation traditionnelle de requêtes ne concerne chaque fois qu'un seul objectif. Néanmoins, pour un système d'intégration de données il est souvent nécessaire de prendre en compte plusieurs de ces objectifs d'optimisation afin d'assurer un bon déroulement du traitement des requêtes. Il s'agit donc de l'*optimisation multi-critères* de requêtes, dont les technologies sont résumées dans [Mahalingam et Candan 2004]. Il serait intéressant d'adapter notre cadre générique d'optimisation pour qu'il puisse gérer le cas d'optimisation multiple-critères. La difficulté est ici l'intégration des stratégies de recherches et des modèles de coût pour l'exploration de l'espace de recherche avec plus d'une fonction d'optimisation. Cette problématique non triviale constitue l'objectif de nos travaux à réaliser à long terme.

Bibliographie

[A. Elmagarmid et Sheth 1998] M. Rusinkiewicz A. Elmagarmid et A. Sheth. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann, 1998.

[Abiteboul 1997] S. Abiteboul. Querying Semistructured Data. In *Proceeding of the 6th International Conference on Database Theory*, Delphi, Greece, 1997.

[Abiteboul 1999] Serge Abiteboul. On Views and XML. In *PODS*, pages 1–9, Philadelphia, Pennsylvania, 1999.

[Aboulnaga *et al.* 2001] A. Aboulnaga, A. Alameldeen, et J. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In *27th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 591–600, Roma, Italy, 2001.

[Adali *et al.* 1996] S. Adali, K. Candan, et Y. Papakonstantinou. Query Caching and Optimization in Distributed Mediator Systems. In *ACM SIGMOD Int'l Conf.on Management of Data*, pages 137–148, Montreal, Canada, 1996.

[Adali 1998] S. Adali. *Query Processing in Heterogeneous Mediated Systems*. PhD thesis, University of Maryland, College Park, 1998.

[Adelberg *et al.* 1996] Brad Adelberg, Ben Kao, et Hector Garcia-Molina. Overview of the stanford real-time information processor (strip). *SIGMOD Rec.*, 25(1) :34–37, 1996.

[Ali et Moerkotte 2004] Robin Ali et Guido Moerkotte. *Query Rewriting with Coko-Kola*. Technical report, April 2004.

[Ambite et Knoblock 1998] J.L. Ambite et C.A. Knoblock. Flexible and scalable query planning in distributed and heterogeneous environments. In *the Fourth International Conference on Artificial Intelligence Planning Systems*, 1998.

[Amer-Yahia *et al.* 2001] S. Amer-Yahia, S. Cho, L.V.S. Lakshmanan, et D. Srivastava. Minimization of Tree Pattern Queries. In *SIGMOD Conference*, 2001.

[Amsaleg *et al.* 1996] L. Amsaleg, M. J. Franklin, A. Tomasic, et T. Urhan. Scarambling Query Plans to Cope With Unexpected Delays. In *Int'l Conf. on Parallel and Distributed Information Systems (PDIS)*, Miami Beach, Florida, 1996.

[Apers *et al.* 1983] P. Apers, A. Hevner, et S. Yao. Optimization Algorithms for Distributed Queries. *IEEE Transactions on Software Engineering*, 9(1) :57–68, 1983.

[B. Babcock 2003] M. Datar R. Motwani J. Widom. B. Babcock, S. Babu. Models and issues in data streams. In *In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, 2003.

[Baru *et al.* 1999] C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinou, et P. Velikhov. XML-Based Information Mediation with MIX. In *ACM-SIGMOD*, Philadelphia, USA, 1999.

[Bernstein *et al.* 1981] P.A. Bernstein, N. G. Goiodman, E. Wong, C. L. Reeve, et J. B. R. Jr. Query Processing in a System for Distributed Databases (SDD-1). *ACM Transactions on Database Systems (TODS)*, 6(4) :602–625, 1981.

[Bernstein et Chiu 1981] P. A. Bernstein et D.-M. W. Chiu. Using Semi-Joins to Solve Relational Queries. *Journal of the ACM*, 28(1) :25–40, 1981.

[Bjørnestad] S. Bjørnestad. The framework for EPOQ - an Extensible Object-Oriented Query Optimizer. citeseer.ist.psu.edu/310683.html.

[Blakeley *et al.* 1993] J. A. Blakeley, W.J. McKenna, et G. Graefe. Experiences Building the Open OODB Query Optimizer. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 287–296, Washington D.C., 1993.

[Bodorik *et al.* 1992] P. Bodorik, J. S. Riordon, et J. S. Pyra. Deciding on Correct Distributed Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 4(3) :253–265, 1992.

[Bodorik et Riordon 1988] P. Bodorik et J. S. Riordon. Distributed Query Processing Optimization Objectives. In *Fourth Int'l Conf. on Data Engineering*, pages 320–329, Los Angeles, California, 1988.

[Bodorik et Riordon 1991] P. Bodorik et J. S. Riordon. Threshold Values for Processing Distributed Queries. *The Computer Journal*, pages 551–558, 1991.

[Bonnet et Tomasic 1998a] P. Bonnet et A. Tomasic. *Parachute Queries in the Presence of Unavailable Data Sources.* Technical Report 3429, INRIA, 1998.

[Bonnet et Tomasic 1998b] P. Bonnet et A. Tomasic. Partial Answer for Unavailable Data Sources. In *Conf. on Flexible Query Answering Systems*, Roskilde, Denmark, 1998.

[Bonnet et Tomasic 1998c] P. Bonnet et A. Tomasic. Unavailable Data Sources in Mediator Based Applications. In *1st Int'l Workshop on Practical Information Mediation and Brokering and the Commerce of Information on the Internet (I'MEDIAT'98)*, Tokyo, Japan, 1998.

[Bonnet 1999] P. Bonnet. *Prise en compte des sources de données indisponibles dans les systèmes de médiation.* PhD thesis, University of Savoie, 1999.

[Braga et Campi 2003] D. Braga et A. Campi. A Graphical Environment to Query XML Data with XQuery. In *WISE*, pages 31–40, 2003.

[Bray *et al.* 1998] T. Bray, J. Paoli, et C. Sperberg-MacQueen. Extensible Markup Language (XML) 1.0 (W3C Recommendation), 1998.

[C. Collet et Shen 1996] M.N. Huhns C. Collet et W. Shen. Resource integration using a large knowledge base in carnot. 1996.

[C. Reynaud et Gagliardi 2005] B. Safar C. Reynaud et H. Gagliardi. Une expérience de représentation d'une ontologie dans le médiateur picsel. 2005.

[Carey *et al.* 1990] M. J. Carey, D. J. DeWitt, G. Graefe, D. M. Haight, J. E. Richardson, D. T. Schuh, E. J. Shekita, et S. Vandenberg. The EXODUS Extensible DBMS Project : An Overview. In D. Maier and S. Zdonik, editor, *Readings on Object-Oriented Database Sys.* Morgan Kaufmann, San Mateo, CA, 1990.

[Carney *et al.* 2002] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, et Stan Zdonik. Monitoring streams : a new class of data management applications. In *VLDB '02 : Proceedings of the 28th international conference on Very Large Data Bases*, pages 215–226, Hong Kong, China, 2002.

[Chen *et al.* 2003] Z. Chen, H.V. Jagadish, L.V.S. Laksmanan, et S. Paparizos. From Tree Patterns to Generalized Tree Patterns : On efficient Evaluation of XQuery. In *VLDB*, 2003.

[Chen 2004] Zhiming Chen. From tree patterns to generalized tree patterns : On efficient evaluation of xquery. University of British Columbia, MSc Thesis, 2004.

[Cherniack et Zdonik 1998] Mitch Cherniack et Stanley B. Zdonik. Changing the rules : Transformations for rule-based optimizers. In *SIGMOD Conference*, 1998.

[Christian Y. A. Brenninkmijer et Paton 2009] Alvaro A. A. Fernandes Christian Y. A. Brenninkmijer, Ixent Galpin et Norman W. Paton. Validated cost models for sensor network queries. In *Proceedings of the Sixth International Workshop on Data Management for Sensor Networks*, 2009.

[Christophides *et al.* 2000] V. Christophides, S. Cluet, et J. Siméon. On Wrapping Query Languages and Efficient XML Integration. In *SIGMOD Conference*, pages 141–152, 2000.

[Cluet *et al.* 1998] S. Cluet, C. Delobel, J. Siméon, et K. Smaga. Your Mediators Need Data Conversion ! In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 177–188, Seattle, Washington, 1998.

[Cluet et Delobel 1992] S. Cluet et C. Delobel. A General Framework for the Optimization of Object-Oriented Queries. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 383–392, San Diego, California, 1992.

[Codd 1970] E.F. Codd. A Relational Model of Data for Large Shared Data Banks. In *Communications of the ACM*, pages 377–387, 1970.

[Council 1998] Transaction Processing Performance Council. The TPC-D Benchmark (Decision Support) Standard Specification, 1998. http ://www.tpc.org/dspec.htm.

[Dang-Ngoc *et al.* 2004] T.-T. Dang-Ngoc, G. Gardarin, et N. Travers. Tree graph view : On efficient evaluation of xquery in an xml mediator. In *Proc. of BDA*, 2004.

[Dang-Ngoc *et al.* 2005] T.T. Dang-Ngoc, C. Jamard., et N. Travers. XLive : An XML Light Integration Virtual Engine. In *Bases de Données Avancées (BDA)*, 2005.

[Dang-Ngoc et Gardarin 2003] T.-T. Dang-Ngoc et G. Gardarin. Federating Heterogeneous Data Sources with XML. In *Proc. of IASTED IKS Conf.*, 2003.

[Dang-Ngoc et Travers 2007] Tuyêt-Trâm Dang-Ngoc et Nicolas Travers. Tree graph views for a distributed pervasive environment. In *the 1st International Conference on Network-Based Information Systems (NBIS)*, Regensburg, Germany, September 2007.

[Dang-Ngoc 2003] Tuyêt-Trâm Dang-Ngoc. *Fédération de données semi-structurées avec XML*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, Versailles, France, 10 Juin 2003.

[Daniels *et al.* 1982] D. Daniels, P.G. Selinger, L. M. Haas, B. G. Lindsay, C. Mohan, A. Walker, et P. F. Wilms. An Introduction to Distributed Query Compilation in R*. In *2nd Int'l Symposium on Distributed Data Bases*, pages 291–309, Berlin, Germany, 1982.

[D.Chamberlin *et al.* 2005] D.Chamberlin, P.Fankhauser, D.Florescu, M.Marchiori, et J.Robie. XML Query Use Cases, september 2005. W3C. http ://www.w3.org/TR/xquery-use-cases.

[Divyakant Agrawal et Abbadi 2010] Sudipto Das Divyakant Agrawal et Amr El Abbadi. Big data and cloud computing : New wine or just new bottles. In *Proceedings of the VLDB Endowment*, 2010.

[Dogac *et al.* 1994] A. Dogac, C. Ozkan, B. Arpinar, T. Okay, et C. Evrendilek. *Advances in Object-Oriented Database Systems*, chapter METU Object-Oriented DBMS. Springer-Verlag, 1994.

[Du *et al.* 1992] W. Du, R. Krishnamurthy, et M.C. Shan. Query Optimization in a Heterogeneous DBMS. In *18th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 277–291, Vancouver, Canada, 1992.

[Du *et al.* 1994] W. Du, M.-C. Shan, et J. Davis. *Optimization and Execution Strategy for Multidatabase Queries.* Technical report, Hewlett-Packard Labs, 1994.

[Du *et al.* 1995] W. Du, M.-C. Shan, et U. Dayal. Reducing Multidatabase Query Presponse Time by Tree Balancing. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 293–303, San Jose, California, 1995.

[Du et Elmagarmid 1989] W. Du et A. K. Elmagarmid. Quasi Serializability : a Correctness Criterion for Global Concurrency Control in InterBase. In *15th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 347–355, Amsterdam, The Netherlands, 1989.

[Du et Shan 1995] W. Du et M.-C. Shan. Query processing in Pegasus. *Object-Oriented Multidatabase Systems : A Solution for Advanced Applications*, pages 449–468, 1995.

[Evrendilek *et al.* 1997] C. Evrendilek, A. Dogac, S. Nural, et F. Ozcan. Multidatabase Query Optimization. *Distributed and Parallel Databases*, 5(1) :77–114, 1997.

[eXist 2007] eXist. Open Source Native XML Database, 2007. http ://exist.sourceforge.net/.

[Fernandez *et al.* 2001] M.F. Fernandez, A. Morishima, et D. Suciu. Efficient Evaluation of XML Middleware Queries. In *SIGMOD '01*, 2001.

[Florescu 1996] D. Florescu. *Espace de Recherche pour l'Optimisation de Requêtes Objet.* PhD thesis, University of Paris IV, 1996.

[G. Gardarin et Tang. 1995] J.-R. Gruser G. Gardarin et Z.-H. Tang. A cost model for clustered object-oriented databases. In *VLDB*, 1995.

[Gardarin *et al.* 1996a] G. Gardarin, J.R. Gruser, et Z.H. Tang. Cost-based Selection of Path Expression Algorithms in Object-Oriented Databases.

In *22nd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 390–401, Bombay, India, 1996.

[Gardarin *et al.* 1996b] G. Gardarin, F. Sha, et Z.H. Tang. Calibrating the Query Optimizer Cost Model of IRO-DB. In *22nd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 378–389, Bombay, India, 1996.

[Gardarin *et al.* 1997] G. Gardarin, B.Finance, et P. Fankhauser. Federating Object-Oriented and Relational Databases : The IRO-DB Experience. In *2nd IFCIS Int'l Conf. on Cooperative Information Systems (CoopIS)*, Kiawah Island, South Carolina, 1997.

[Gardarin 1997] G. Gardarin. Multimedia Federated Databases on Intranets : Web-Enabling IRO-DB. In *8th Int'l Conf. on Database and Expert Systems Applications, (DEXA '97)*, pages 16–27, Toulouse, France, 1997.

[Gardy et Puech 1989] D. Gardy et C. Puech. On the Effects of Join Operations on Relation Sizes. *ACM Transactions on Database Systems (TODS)*, 14(4) :574–603, 1989.

[Golab et Özsu 2003] Lukasz Golab et M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2) :5–14, 2003.

[Goldman et Widom 1997] R. Goldman et J. Widom. DataGuides : Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB*, 1997.

[Graefe et McKenna 1993] G. Graefe et W.J. McKenna. The Volcano Optimizer Generator : Extensibility and Efficient Search. In *ICDE*, pages 209–218, 1993.

[Gruser 1996] J.R. Gruser. *Modèle de Coût pour l'Optimisation de Requêtes Objet.* PhD thesis, University of Paris IV, 1996.

[Haas *et al.* 1997a] L. M. Haas, D. Kossmann, E. L. Wimmers, et J. Yang. Optimization Queries Across Diverse Data Sources. In *VLDB*, 1997.

[Haas *et al.* 1997b] L. M. Haas, D. Kossmann, E. L. Wimmers, et J. Yang. Optimization Queries Across Diverse Data Sources. In *23rd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 276–285, Athens, Greece, 1997.

[Haas *et al.* 2002] L. M. Haas, E. T. Lin, et M. A. Roth. Data integration through database federation. *IBM Syst. J.*, 41(4) :578–596, 2002.

[Hacid et Reynaud 2004] Mohanad Said Hacid et Chantal Reynaud. L'intégration de sources de données. *Revue Information - Interaction - Intelligence*, (R I3), 2004.

[Hevner et Yao 1979] A. R. Hevner et S. B. Yao. Query Processing in Distributed Database Systems. *IEEE Transactions on Software Engineering*, 5(3) :177–187, 1979.

[Ioannidis et Christodoulakis 1991] Y.E. Ioannidis et S. Christodoulakis. On the propagation of errors in the size of join results. In *SIGMOD*, pages 268–277, Denver, Colorado, 1991.

[Ioannidis 1993] Y.E. Ioannidis. Universality of serial histograms. In *VLDB*, page 256, Dublin, Irlande, 1993.

[Ioannidis 1996] Yannis E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1) :121–123, 1996.

[Kim 1995] W. Kim. *Modern Database Systems : The Object Model, Interoperability, and Beyond*. Addison-Wesley, 1995.

[Kirkpatrick *et al.* 1983] S. Kirkpatrick, C. D. Gelatt Jr., et M. P. Vecchi. Optimization by simulated annealing. In *Science*, 1983.

[L. Bouganim 2004] F. Dang Ngoc et P. Pucheral L. Bouganim. Client-based access control management for xml documents. In *International Conference on Very Large Databases (VLDB)*, Toronto, 2004.

[Levy *et al.* 1995] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, et D. Srivastava. Answering Queries Using Views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, San Jose, California, 1995.

[Levy *et al.* 1996a] A. Levy, A. Rajaraman, et J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *22nd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 251–262, Bombay, India, 1996.

[Levy *et al.* 1996b] A. Y. Levy, A. Rajaraman, et J. D. Ullman. Answering Queries Using Limited External Processors. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 227–237, Montreal, Canada, 1996.

[Liu *et al.* 2007] T. Liu, T. T. Dang-Ngoc, et D. Laurent. Cost framework for a distributed semi-structured environment. In Springer, editor, *in the proceedings of the International workshop Database Management and Application over Networks - DBMAN (APWeb/WAIM Workshop)*, pages 1–11, Huangshan, China, June 2007.

[Liu 2007] T. Liu. Costbased query optimization in a heterogeneous distributed semistructured environment. In *VLDB PhD Workshop*, Vienna, Austria, 2007.

[Mackert et Lohman 1986] L. F. Mackert et G. M. Lohman. R* Optimizer Validation and Performance Evaluation for Local Queries. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 84–95, Washington D.C., 1986.

[Mahalingam et Candan 2004] Lakshmi Priya Mahalingam et K. Selçuk Candan. Multi-criteria query optimization in the presence of result size and quality tradeoffs. *Multimedia Tools Appl.*, 23(3) :167–183, 2004.

[McHugh et Widom 1999a] J. McHugh et J. Widom. *Query Optimization for SemiStructured Data.* Technical report, Stanford University Database Group, 1999.

[Mchugh et Widom 1999b] Jason Mchugh et Jennifer Widom. Query Optimization for XML. In *25th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 315–326, Edinburgh, Scotland, 1999.

[McHugh et Widom February 1999] J. McHugh et J. Widom. *Query Optimization for SemiStructured Data.* Technical report, Stanford University Database Group, February, 1999.

[Michael Genesereth et Duschka 1997] Arthur Keller Michael Genesereth et Oliver Duschka. Infomaster : an information integration system. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 539–542, Tucson, AZ, 1997.

[Mueller *et al.* 2009] Rene Mueller, Jens Teubner, et Gustavo Alonso. Streams on wires : A query compiler for fpgas. In *Proc. VLDB Endowment*, Lyon, France, August 2009.

[Naacke *et al.* 1998] H. Naacke, G. Gardarin, et A. Tomasic. Leveraging Mediator Cost Models with Heterogeneous Data Sources. In *14th Int'l Conf. on Data Engineering (ICDE)*, Orlando, Florida, 1998.

[Naacke *et al.* 1999] H. Naacke, A. Tomasic, et P. Valduriez. Validating Mediator Cost Models with DISCO. *Networking and Information Systems Journal*, 1999.

[Nahar *et al.* 1986] S. Nahar, S. Sahni, et E. Shragowitz. Simulated annealing and combinatorial optimization. In *23rd Design Automation Conference*, 1986.

[Nehme *et al.* 2009] Rimma V. Nehme, Elke A. Rundensteiner, et Elisa Bertino. Tagging stream data for rich real-time services. *Proc. VLDB Endow.*, 2(1) :73–84, 2009.

[OGC 2007] OGC. Sensor model language (sensorml), 2007. http ://www.opengeospatial.org/standards/sensorml.

[Ouzzani et Bouguettaya 2004] Mourad Ouzzani et Athman Bouguettaya. Query processing and optimization on the web. *Distrib. Parallel Databases*, 15(3) :187–218, 2004.

[Ozcan *et al.* 1996] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, et A. Dogac. Dynamic Query Optimization on a Distributed Object Management

Plateform. In *Int'l Conf. on Information and Knowledge Management (CIKM)*, pages 117–124, Rockville, Maryland, 1996.

[Ozcan *et al.* 1997] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, et A. Dogac. Dynamic Query Optimization in Multidatabases. *Data Engineering Bulletin*, 20(3) :38–45, 1997.

[P. Laublet et Charlet 2002] C. Reynaud P. Laublet et J. Charlet. Sur quelques aspects du web sémantique. 2002.

[Papakonstantinou *et al.* 1993] Y. Papakonstantinou, A. Gupta, et L. M. Haas. Capabilities-based Query Rewriting in Mediator Systems. *Distributed and Parallel Databases*, 6(1) :73–110, 1993.

[Papakonstantinou *et al.* 2003] Y. Papakonstantinou, V. Borkar, M. Orgiyan, K. Stathatos, L. Suta, V. Vassalos, et P. Velikhov. XML Queries and Algebra in the Enosys Integration Platform. *Data Knowledge Engineering*, 44(3) :299–322, 2003.

[Papakonstantinou 1997] Y. Papakonstantinou. *Query Processing in Heterogeneous Information Sources.* PhD thesis, Stanford University, 1997.

[Pellenkoft 1997] A. J. Pellenkoft. *Probabilistic and Transformation Based Query Optimization.* PhD thesis, University of Amsterdam, 1997.

[Perriezo 1958] W. Perriezo. A distributed query processing algorithm yielding a least upper bound response time semijoin strategy. In *Supercomputing Systèmes*, 1958.

[Piatetsky-Shapiro et Connell 1984] G. Piatetsky-Shapiro et C. Connell. Accurate Estimation of the Number of Tuples Satisfying a Condition. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 256–276, Boston, Massachusetts, 1984.

[Ramakrishanan 1998] R. Ramakrishanan. *Database Management Systems.* McGraw-Hill, 1998.

[René Müller et Alonso 2009] Jens Teubner René Müller et Gustavo Alonso. Data processing on fpgas. In *Proc. VLDB Endowment*, Lyon, France, August 2009.

[Roth *et al.* 1999] M. T. Roth, F. Ozcan, et L. M. Haas. Cost Models DO Matter : Providing Cost Information for Diverse Data Sources in a Federated System. In *In 25th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 599–610, Edinburgh, Scotland, 1999.

[Roth et Schwarz 1997] M. T. Roth et P.M. Schwarz. Don't Scrap It, Wrap it ! A Wrapper Architecture for Legacy Data Sources. In *In 23th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 266–275, Athens, Greece, 1997.

[S. Madden 2003] J. M. Hellerstein W. Hong. S. Madden, M. J. Franklin. The design of an acquisitional query processor for sensor networks. In *In Proc. ACM Int. Conf. on Management of Data*, 2003.

[Sannella 1994] Michael Sannella. *Constraint Satisfaction and Debugging for Interactive User Interfaces.* PhD thesis, University of Washington, Seattle, WA, 1994.

[Selinger *et al.* 1979a] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, et T. G. Price. Access path selection in a relational database management system. In *In Proc. ACM-SIGMOD Conf. on the Management of Data*, pages 23–34, Boston, MA, June 1979.

[Selinger *et al.* 1979b] P.G. Selinger, M. Astrahan, D. Chamberlin, R.A. Lorie, et T.G. Price. Access Path Selection in a Relational Database Management System. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 23–24, Boston, Massachusetts, 1979.

[Selinger *et al.* 1979c] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, et T.G. Price. Access Path Selection in a Relational Database Management System. In *ACM-SIGMOD*, 1979.

[Selinger et Adiba 1982] P.G. Selinger et M. Adiba. Access path selection in distributed database management systems. In *Int'l Conf. On Databases (ICOD)*, pages 204–215, Aberdeen, Scotland, 1982.

[Sellis 1988] T. K. Sellis. Multiple-Query Optimization. *ACM Transactions on Database Systems (TODS)*, 13(1) :23–52, 1988.

[S.Epstein *et al.* 1978] R. S.Epstein, M. Stonebraker, et E.Wong. Distributed Query Processing in a Relational Data Base System. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 169–180, Austin, Texas, 1978.

[Sheth et Larson 1990] A. P. Sheth et J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys*, 22(3) :183–236, 1990.

[Sihem *et al.* 2002] Amer-Yahia Sihem, Cho SungRan, Lakshmanan V. S. Laks, et Srivastava Divesh. Tree Pattern Query Minimization. *VLDB Journal*, 11(4) : :315–331, 2002.

[Subrahmanian et Venkataraman 1998] S. N. Subrahmanian et S. Venkataraman. Cost-Based Optimization of Decision Support Queries Using Transient Views. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 319–330, Seattle, Washington, 1998.

[T.Landers et Rosenberg 1982] T.Landers et R. L. Rosenberg. An Overview of MULTIBASE. In H.-J. Schneider, editor, *Second International Symposium on Distributed Data Bases*, pages 153–184, Berlin, Germany, 1982. North-Holland.

[Tomasic *et al.* 1997] A. Tomasic, R. Amouroux, P. Bonnet, O. Kapitskaia, H. Naacke, et L. Raschid. The Distributed Information Search Component (Disco) and the World Wide Web. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 546–548, Tucson, Arizona, 1997.

[Tomasic *et al.* 1998] A. Tomasic, L. Raschid, et P. Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5) :808–823, 1998.

[Travers *et al.* 2006a] N. Travers, T.-T. Dang-Ngoc, et T. Liu. TGV : an efficient Model for XQuery Evaluation within an Interoperable System. *IBIS journal*, 3, 2006.

[Travers *et al.* 2006b] N. Travers, T.T. Dang-Ngoc, et T. Liu. TGV : An Efficient Model for XQuery Evaluation within an Interoperable System. *Int. Journal of Interoperability in Business Information Systems (IBIS)*, 3, 2006.

[Travers *et al.* 2007a] N. Travers, T.-T. Dang-Ngoc, et T. Liu. An Efficient Evaluation of XQuery with TGV (demonstration). In *WebIST*, 2007.

[Travers *et al.* 2007b] Nicolas Travers, Tuyêt-Trâm Dang-Ngoc, et Tianxiao Liu. Full untyped xquery canonisation. In Springer, editor, *in the proceedings of the International workshop on Emerging Trends of Web Technologies and Applications -WebETrends (APWeb/WAIM Workshop)*, pages 358–371, Huangshan, China, June 2007.

[Travers *et al.* 2007c] Nicolas Travers, Tuyêt-Trâm Dang-Ngoc, et Tianxiao Liu. Tgv : a tree graph view for modelling untyped xquery. In *the 12th International Conference on Database Systems for Advanced Applications (DASFAA)*, Bangkok, Thailand, April 2007.

[Travers et Dang-Ngoc 2007] Nicolas Travers et Tuyêt-Trâm Dang-Ngoc. An extensible rule transformation model for xquery optimization. In *The 9th International Conference on Enterprise Information Systems (ICEIS)*, Madeira, Portugal, June 2007.

[Travers 2006] Nicolas Travers. *Optimisation Extensible dans un Médiateur de Données XML*. PhD thesis, University of Versailles, 2006.

[Urhan *et al.* 1998] T. Urhan, M. J. Franklin, et L. Amsaleg. Cost Based Query Scrambling for Initial Delays. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 130–141, Seattle, Washington, 1998.

[Valduriez 1982] P. Valduriez. Semi-Join Algorithms for Multiprocessor Systems. In *ACMSIGMOD Int'l Conf.on Management of Data*, pages 225–233, Orlando, Florida, 1982.

[Valuduriez 1987] P. Valuduriez. Join indices. *ACM Transactions on Database Systems (TODS)*, 12(2) :218–246, 1987.

[Vassalos et Papakonstantinou 1997] V. Vassalos et Y. Papakonstantinou. Describing and Using Query Capabilities of Heterogeneous Sources. In *23rd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 256–265, Athens, Greece, 1997.

[Viglas et Naughton. 2002] S. Viglas et J. Naughton. Rate-based query optimization for streaming information sources. In *In Proc. ACM Int. Conf. on Management of Data*, 2002.

[V.S. Subrahmanian et Ward 1995] Anne Brink Ross Emery James J. Lu Adil Rajput Timothy J. Rogers Robert Ross V.S. Subrahmanian, Sibel Adah et Charles Ward. Hermes : A heterogeneous reasoning and mediator system. 1995.

[W3C 1999] W3C. Xml path language (xpath) version 1.0, novembre 1999. http ://www.w3.org/TR/1999/REC-xpath-19991116.

[W3C 2001] W3C. Web services description language (wsdl) 1.1, 2001. http ://www.w3.org/TR/wsdl.

[W3C 2003] W3C. Mathematical Markup Language (Mathml TM) Version 2.0, 2003. http ://www.w3.org/TR/REC-MathML.

[W3C 2004] W3C. Resource description framework (rdf), 2004. http ://www.w3.org/RDF/.

[W3C 2005] W3C. An XML Query Language (XQuery 1.0), 2005. http ://www.w3.org/TR/xquery/.

[Widom 1996] J. Widom. The Starburst Active Database Rule System. *Knowledge and Data Engineering*, 8(4) :583–595, 1996.

[Wiederhold 1992] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3) :38–49, March 1992.

[Y. Papakonstantinou et Ullman 1995] H. Garcia-Molina Y. Papakonstantinou, A. Gupta et J. Ullman. A query translation scheme for rapid implementation of wrappers. In *Proceedings of ACM PODS 1995*, 1995.

[Yao 1977] S. B. Yao. Approximating the Number of Accesses in Database Organizations. *Communications of ACM (CACM)*, 20(4) :260–261, 1977.

[Zhang *et al.* 2005] N. Zhang, P. J. Haas, V. Josifovski, et C. Zhang G. M. Lohman. Statistical Learning Techniques for Costing XML Queries. In *31th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 289–300, Trondheim, Norway, 2005.

[Zhu et Larson 1998] Q. Zhu et P. A. Larson. Solving Local Cost Estimation Problem for Global Query Optimization in Multidatabase Systems. *Distributed and Parallel Databases*, 1998.

[Zhu 1993] Q. Zhu. An Integrated Method for Estimating Selectivities in a Multidatabase System. In *Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, pages 832–847, Toronto, Ontario, Canada, 1993.

[Zhu 1995] Q. Zhu. *Estimating Local Cost Parameters for Global Query Optimization in a Multidatabase System*. PhD thesis, University of Waterloo, 1995.

[Zhu 1996] Q. Zhu. Developing Regression Cost Models for Multidatabase Systems. In *Int'l Conf. on Parallel and Distributed Information Systems (PDIS)*, pages 220–231, Miami Beach, Florida, 1996.

[Zloof 1977] M. Zloof. QBE : Query By Example, 1977.

[Özsu et Valduriez 1999] M.T. Özsu et P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

Résumé Dans cette thèse, nous proposons un cadre générique d'optimisation de requêtes dans les environnements hétérogènes répartis. Nous proposons un modèle générique de description de sources (GSD), qui permet de décrire tous les types d'informations liées au traitement et à l'optimisation de requêtes. Avec ce modèle, nous pouvons en particulier obtenir les informations de coût afin de calculer le coût des différents plans d'exécution. Notre cadre générique d'optimisation fournit les fonctions unitaires permettant de mettre en oeuvre les procédures d'optimisation en appliquant différentes stratégies de recherche. Nos résultats expérimentaux mettent en évidence la précision du calcul de coût avec le modèle GSD et la flexibilité de notre cadre générique d'optimisation lors du changement de stratégie de recherche. Notre cadre générique d'optimisation a été mis en oeuvre et intégré dans un produit d'intégration de données (DVS) commercialisé par l'entreprise Xcalia - Progress Software Corporation. Pour des requêtes contenant beaucoup de jointures inter-site et interrogeant des sources de grand volume, le temps de calcul du plan optimal est de l'ordre de 2 secondes et le temps d'exécution du plan optimal est réduit de 28 fois par rapport au plan initial non optimisé.

Abstract This thesis proposes a generic framework for query optimization in heterogeneous and distributed environments. We propose a generic source description model (GSD), which allows describing any type of information related to query processing and optimization. With GSD, we can use cost information to calculate the costs of execution plans. Our generic framework for query optimization provides a set of unitary functions used to perform optimization by applying different search strategies. Our experimental results show the accuracy of cost calculus when using GSD, and the flexibility of our generic framework when changing search strategies. Our proposed approach has been implemented and integrated in a data integration product (DVS) licensed by Xcalia - Progress Software Corporation. For queries with many inter-site joins accessing large size data sources, the time used for finding the optimal plan is in the order of 2 seconds, and the execution time of the optimized plan is reduced by 28 times, as compared with the execution time of the non optimized original plan.