Une méthode d'optimisation hybride pour une évaluation robuste de requêtes

Chiraz Moumen

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse III - Paul Sabatier

Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Présentée et soutenue par : Chiraz MOUMEN

Le 29 Mai 2017

Titre : Une méthode d'optimisation hybride pour une évaluation robuste de requêtes

---

JURY

Agnès FRONT	Maitre de conférences HDR, Université Grenoble Alpes	Rapporteur
Abdelkader HAMEURLAIN	Professeur, Université Paul Sabatier	Examinateur
Philippe LAMARRE	Professeur, INSA de Lyon	Rapporteur
Franck MORVAN	Professeur, Université Paul Sabatier	Directeur de recherche
Michael MRISSA	Professeur, Université de Pau et des pays de l'Adour	Examinateur
Shaoyi YIN	Maitre de conférences, Université Paul Sabatier	Examinateur

---

Ecole doctorale : MITT : Mathématiques, Informatique, Télécommunications de Toulouse

Unité de recherche : Institut de Recherche en Informatique de Toulouse (UMR 5505)

Directeur de recherche : Professeur Franck MORVAN

Remerciements

Je tiens tout d'abord à remercier mon directeur de recherche Monsieur le Professeur Franck MORVAN, pour la confiance qu'il m'a accordée depuis mon Master de recherche, pour ses conseils, pour la rigueur et la précision scientifique qu'il m'a apprises et pour tous ce qu'il m'a apportés sur le plan professionnel aussi bien que sur le plan humain.

J'adresse ma profonde gratitude à Monsieur le Professeur Abdelkader HAMEURLAIN, responsable de l'équipe Pyramide au sein de laquelle j'ai effectué mes travaux de thèse, de m'avoir accueilli dans l'équipe me faisant ainsi vivre une expérience enrichissante, de ses conseils judicieux et d'avoir accepté de participer à ce jury.

Je remercie également Madame Agnès FRONT et Monsieur le Professeur Philippe LAMARRE de l'intérêt qu'ils ont porté à mon travail en acceptant d'en être les rapporteurs et de leurs remarques encourageantes et constructives.

Mes remerciements vont également à Madame Shaoyi YIN pour les discussions que j'ai eu la chance d'avoir avec elle et pour ses conseils judicieux. Je remercie Madame Shaoyi YIN et Monsieur le Professeur Michael MRISSA d'avoir consacré du temps à l'examen de ce mémoire et d'avoir accepté de participer à ce jury.

Je remercie aussi Monsieur Riad MOKADEM, maitre de conférences au sein de l'équipe Pyramide, pour ses encouragements et ses conseils.

J'adresse un grand merci à mes parents et à mon mari sans qui ce travail n'aurait pas abouti. Je remercie mes parents pour leur soutien moral et leur encouragement dans les moments de doute. Je remercie mon mari pour sa patience, son soutien et son encouragement quand je perdais confiance en moi.

Enfin, je remercie toute personne qui m'a aidé de près ou de loin à réussir cette thèse.

Chiraz Moumen

Résumé

La qualité d'un plan d'exécution engendré par un optimiseur de requêtes est fortement dépendante de la qualité des estimations produites par le modèle de coûts. Malheureusement, ces estimations sont souvent imprécises. De nombreux travaux ont été menés pour améliorer la précision des estimations. Cependant, obtenir des estimations précises reste très difficile car ceci nécessite une connaissance préalable et détaillée des propriétés des données et des caractéristiques de l'environnement d'exécution.
Motivé par ce problème, deux approches principales de méthodes d'optimisation ont été proposées. Une première approche s'appuie sur des valeurs singulières d'estimations pour choisir un plan d'exécution optimal. A l'exécution, des statistiques sont collectées et comparées à celles estimées. En cas d'erreur d'estimation, une ré-optimisation est déclenchée pour le reste du plan. A chaque invocation, l'optimiseur associe des valeurs spécifiques aux paramètres nécessaires aux calculs des coûts. Cette approche peut ainsi induire plusieurs ré-optimisations d'un plan, engendrant ainsi de mauvaises performances. Dans l'objectif d'éviter cela, une approche alternative considère la possibilité d'erreurs d'estimation dès la phase d'optimisation. Ceci est modélisé par l'utilisation d'un ensemble de points d'estimations pour chaque paramètre présumé incertain. L'objectif est d'anticiper la réaction à une sous-optimalité éventuelle d'un plan d'exécution. Les méthodes dans cette approche cherchent à générer des plans robustes dans le sens où ils sont capables de fournir des performances acceptables et stables pour plusieurs conditions d'exécution. Ces méthodes supposent souvent qu'il est possible de trouver un plan robuste pour l'ensemble de points d'estimations considéré. Cette hypothèse reste injustifiée, notamment lorsque cet ensemble

est important. De plus, la majorité de ces méthodes maintiennent sans modification un plan d'exécution jusqu'à la terminaison. Cela peut conduire à de mauvaises performances en cas de violation de la robustesse à l'exécution.
Compte tenu de ces constatations, nous proposons dans le cadre de cette thèse une méthode d'optimisation hybride qui vise deux objectifs : la production de plans d'exécution robustes, notamment lorsque l'incertitude des estimations utilisées est importante, et la correction d'une violation de la robustesse pendant l'exécution. Notre méthode s'appuie sur des intervalles d'estimations calculés autour des paramètres incertains, pour produire des plans d'exécution robustes. Ces plans sont ensuite enrichis par des opérateurs dits de contrôle et de décision. Ces opérateurs collectent des statistiques à l'exécution et vérifient la robustesse du plan en cours. Si la robustesse est violée, ces opérateurs sont capables de prendre des décisions de corrections du reste du plan sans avoir besoin de rappeler l'optimiseur. Les résultats de l'évaluation des performances de notre méthode indiquent qu'elle fournit des améliorations significatives dans la robustesse d'évaluation de requêtes.

Mots clés : bases de données, optimisation robuste, modèle de coûts, erreurs d'estimation.

Abstract

The quality of an execution plan generated by a query optimizer is highly dependent on the quality of the estimates produced by the cost model. Unfortunately, these estimates are often imprecise. A body of work has been done to improve estimate accuracy. However, obtaining accurate estimates remains very challenging since it requires a prior and detailed knowledge of the data properties and run-time characteristics.

Motivated by this issue, two main optimization approaches have been proposed. A first approach relies on single-point estimates to choose an optimal execution plan. At run-time, statistics are collected and compared with estimates. If an estimation error is detected, a re-optimization is triggered for the rest of the plan. At each invocation, the optimizer uses specific values for parameters required for cost calculations. Thus, this approach can induce several plan re-optimizations, resulting in poor performance. In order to avoid this, a second approach considers the possibility of estimation errors at the optimization time. This is modelled by the use of multi-point estimates for each error-prone parameter. The aim is to anticipate the reaction to a possible plan sub-optimality. Methods in this approach seek to generate robust plans, which are able to provide good performance for several run-time conditions. These methods often assume that it is possible to find a robust plan for all expected run-time conditions. This assumption remains unjustified. Moreover, the majority of these methods maintain without modifications an execution plan until the termination. This can lead to poor performance in case of robustness violation at run-time.

Based on these findings, we propose in this thesis a hybrid optimization method that

aims at two objectives : the production of robust execution plans, particularly when the uncertainty in the used estimates is high, and the correction of a robustness violation during execution. This method makes use of intervals of estimates around error-prone parameters. It produces execution plans that are likely to perform reasonably well over different run-time conditions, so called robust plans. Robust plans are then augmented with what we call check-decide operators. These operators collect statistics at run-time and check the robustness of the current plan. If the robustness is violated, check-decide operators are able to make decisions for plan modifications to correct the robustness violation without a need to recall the optimizer.
The results of performance studies of our method indicate that it provides significant improvements in the robustness of query processing.

Keywords : Databases, robust optimization, cost models, estimation errors.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Contexte

Un optimiseur de requêtes est un composant critique dans les systèmes de gestion de bases de données (SGBDs). Son rôle consiste à produire un plan d'exécution efficace pour une requête en entrée. Pour produire ce plan, l'optimiseur s'appuie généralement sur un modèle de coûts. Ce dernier permet d'estimer le coût d'un plan d'exécution associé à une requête. Le coût peut être une valeur proportionnelle à l'utilisation estimée de ressources nécessaires pour exécuter une requête avec un plan particulier. Il peut aussi correspondre à une estimation du temps de réponse de la requête pour ce plan. Les ressources incluent le nombre d'entrées/sorties disque, la quantité de mémoire vive nécessaire et le temps CPU. Le temps de réponse est le temps écoulé entre le moment où une requête est soumise par l'utilisateur et le renvoi du résultat complet de cette requête à l'utilisateur. Le temps de réponse se compose essentiellement d'un temps d'optimisation et d'un temps d'exécution. Le temps d'optimisation est le temps écoulé entre la soumission de la requête et la production d'un ou plusieurs plans d'exécution pour cette requête. Le temps d'exécution est le temps nécessaire pour fournir le résultat complet de l'exécution d'un ou des plans associés à la requête.

Les travaux de recherche portant sur l'optimisation de requêtes remontent aux années 70s, quand Selinger et al. [SAC+79] ont publié leur papier «*System R*». Dans ce papier, les auteurs ont introduit une approche d'optimisation de requêtes relationnelles. Cette approche peut être décrite comme suit : pour une requête en entrée, un optimiseur commence par choisir une parmi de nombreuses transformations -logiquement équivalentes- de la requête. Celles-ci sont représentées sous forme d'expressions algébriques. Elles sont appelées *plans logiques*. Des plans physiques sont ensuite énumérés à partir du plan logique choisi. Un plan physique est établi en ordonnançant les opérateurs du plan logique et en sélectionnant les algorithmes physiques pour implémenter les opérateurs de ce plan. A un plan logique correspondent plusieurs plans physiques possibles. Ces plans forment l'espace de recherche. L'optimiseur est responsable de sélectionner le meilleur plan dans l'espace de recherche. Pour cela, il s'appuie sur un modèle de coûts pour estimer le coût

relatif à l'exécution de chaque plan physique. Cette estimation permet de trouver le plan présumé le moins coûteux. Celui-ci est dit *optimal*. Ce plan constitue l'entrée du moteur d'exécution qui produit les résultats (Cf. Figure 1.1).

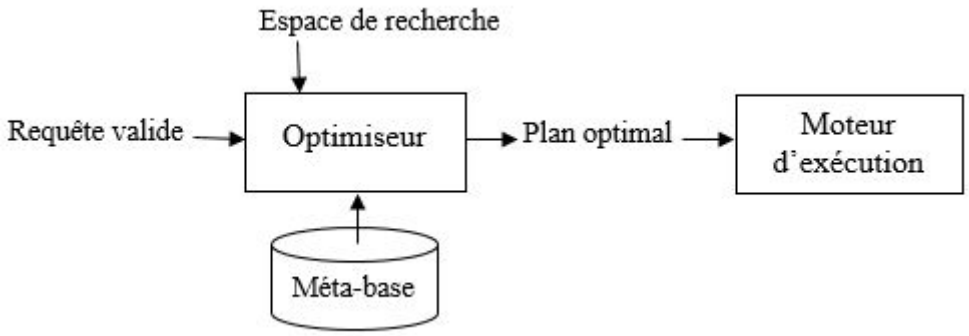


FIGURE 1.1 – Approche d'optimisation statique

Cette approche d'optimisation est dite statique du fait qu'un plan d'exécution engendré par l'optimiseur est maintenu sans modification jusqu'à la terminaison de l'exécution. Les méthodes proposées dans le cadre de cette approche supposent que les estimations de paramètres nécessaires aux calculs des coûts (e.g., tailles des relations temporaires, sélectivités des opérateurs, quantité de mémoire disponible, charge CPU) restent valides durant l'exécution. Par conséquent, le plan choisi reste optimal. Cependant, il existe plusieurs raisons pour lesquelles ces estimations peuvent devenir inexactes lors de l'exécution. Les principales raisons sont les suivantes [Chr84, PI97, GSW12, LGM+15] :

- *Utilisation d'hypothèses invalides* : des hypothèses comme l'uniformité de la distribution des valeurs d'attributs et/ou l'indépendance entre les valeurs d'attributs sont souvent utilisées pour simplifier les estimations des coûts de plans d'exécution associé à une requête. Leur utilisation satisfait rarement la nature réelle des données. Prenons l'exemple d'une requête évoquant deux attributs d'une même relation (e.g., pays='France' et ville='Paris'). Les coûts des plans d'exécution possibles pour cette requête sont proportionnels aux sélectivités résultantes de cette clause de sélection. Les estimations des sélectivités dépendent de la distribution des données

de ces attributs, i.e., la probabilité P(ville='Paris' / pays='France'). Afin d'éviter la complexité de l'estimation de ce paramètre, la majorité des SGBDs supposent l'indépendance des valeurs des attributs les uns des autres. Ainsi, la sélectivité est calculée comme le produit des sélectivités des attributs estimées séparément, i.e., P(pays='France' et ville='Paris') = P(pays='France') × P(ville='Paris'). Supposer l'indépendance entre les attributs dans un tel exemple s'oppose à la nature des données et peut entraîner des estimations erronées des sélectivités.

- *Imprécision des statistiques sur les données* : le processus d'estimation des coûts de plans d'exécution dépend de façon critique des estimations des cardinalités de différentes sous-expressions d'une requête. Malgré nombreux travaux (e.g., [TDJ11, TDJ13, WNS16]) proposés pour améliorer la précision des statistiques et les systèmes de collecte de données, de nombreux paramètres peuvent toujours être modélisés par des statistiques inexactes. Les corrélations entre prédicats peuvent faire en sorte que les estimations des tailles des relations temporaires soient erronées.

- *Absence de statistiques sur les données* : des erreurs d'estimations peuvent être dues à une indisponibilité de statistiques. Un système de bases de données peut avoir à exécuter une requête impliquant une ou plusieurs sources de données sur lesquelles aucune statistique n'est disponible [BB05], e.g., dans un environnement à grande échelle, il est difficile d'obtenir/maintenir à jour des statistiques sur les sites distants.

- *Nature dynamique de l'environnement d'exécution* : les caractéristiques d'un environnement d'exécution peuvent évoluer pendant une exécution. Par exemple, la quantité de mémoire disponible et la charge CPU sont deux paramètres dont les valeurs peuvent changer à tout instant. La charge CPU peut augmenter pendant l'exécution d'une requête. Cette augmentation entraine des variations dans les ressources disponibles (e.g., mémoire). Ainsi, les estimations de ces paramètres à l'optimisation peuvent devenir obsolètes à l'exécution.

Pour les raisons citées ci-dessus, les estimations utilisées par un optimiseur de requêtes pour générer un plan d'exécution, peuvent être erronées. Ioannidis et al. [IC91] ont montré qu'une erreur d'estimation se propage dans un plan d'exécution d'une manière exponentielle par rapport au nombre d'opérateurs. Cela mène à une sous-optimalité du plan et ainsi une dégradation significative des performances.

Le problème d'erreurs d'estimations et les sous-optimalités qui en résultent a reçu une attention considérable et a fait l'objet de nombreux travaux dans la littérature [BB05, DIR07, CK14, YHM15, MMH16]. Ces travaux incluent principalement (i) des méthodes pour une meilleure qualité des estimations (e.g., [PI97, GTK01, TDJ11, TDJ13, WNS16]), (ii) des méthodes permettant de contrôler l'exécution d'un plan et déclencher une ré-optimisation du reste du plan si une erreur d'estimation est détectée (e.g., [KD98, MRS$^+$04, BJZ13]), et (iii) des méthodes qui permettent d'engendrer des plans d'exécution robustes dans le sens où ils sont capables de fournir des performances acceptables et stables pour différentes conditions d'exécution (e.g., [BC05, ABD$^+$10, AHW15a]).

Malgré les efforts considérables qui ont été déployés pour une meilleure qualité des estimations, les méthodes proposées (e.g., [GTK01, TDJ13, WNS16]) ne permettent qu'une amélioration de la précision des estimations. Obtenir des estimations précises reste néanmoins très difficile puisque ceci nécessite une connaissance préalable et détaillée des propriétés des données (e.g., distribution des valeurs d'un attribut) et des caractéristiques de l'environnement d'exécution (e.g., charge CPU, quantité de mémoire disponible). Pour les méthodes (e.g., [KD98, MRS$^+$04, BJZ13]) proposées pour contrôler l'exécution d'un plan associé à une requête et déclencher -si nécessaire- une correction de ce plan, elles présentent un risque d'une dégradation des performances due aux sur-coûts relatifs à de multiples corrections possibles d'un plan d'exécution. Quant aux méthodes d'optimisation (e.g., [BC05, ABD$^+$10, AHW15a]) qui visent à produire des plans d'exécution robustes, elles opèrent généralement sous l'hypothèse qu'il est toujours possible de trouver un plan d'exécution robuste à toutes les circonstances prévues. Cette hypothèse reste injustifiée, notamment lorsque l'incertitude concernant les caractéristiques de l'exécution est impor-

tante. En outre, la majorité de ces méthodes suggèrent de maintenir sans modification un plan d'exécution jusqu'à la terminaison. Lorsque les conditions d'exécution sont peu prévisibles, des plans estimés robustes peuvent résulter en une violation de la robustesse à l'exécution. Maintenir l'exécution de ces plans peut induire de mauvaises performances.

Compte tenu des limites des méthodes existantes, nous proposons dans ce document une méthode d'optimisation hybride qui vise deux objectifs : la production de plans d'exécution robustes qui gèrent l'incertitude des estimations utilisées à l'optimisation, et la détection et la correction d'une violation de la robustesse pendant l'exécution. Dans la sous-section suivante, nous détaillons notre problématique de recherche.

1.2 Définition de la problématique de recherche

Une solution pour tenir compte de la possibilité d'erreurs dans les estimations utilisées à l'optimisation, est de considérer des intervalles autour de ces estimations plutôt que des valeurs singulières. La littérature de recherche propose plusieurs techniques de calcul de tels intervalles. Par exemple, dans [SB99], les auteurs déterminent un intervalle d'estimations d'un paramètre en calculant les limites supérieure et inférieure de la valeur de ce paramètre. Dans [BBD05], des intervalles sont calculés en fonction des estimations singulières obtenues. Dans [BBD05], un intervalle d'estimations est calculé comme suit :

$$\begin{aligned} h_i &= E \times (1 + 0.2 \times U) \\ l_o &= E \times (1 - 0.1 \times U) \end{aligned} \tag{1.1}$$

h_i (respectivement l_o) définit la borne supérieure (respectivement inférieure) de l'intervalle. E réfère à une estimation spécifique de la valeur du paramètre en question. Finalement, U représente le niveau d'incertitude concernant la précision de l'estimation E. U peut prendre une valeur entière entre 0 (pas d'incertitude) et 6 (incertitude très élevée). Cette valeur est déterminée en se basant sur la manière dont l'estimation E a été obtenue.

Le problème de calcul d'intervalles d'estimations est hors de portée de ce document. Nous supposons que les intervalles requis sont disponibles. Aussi, nous supposons qu'un intervalle d'estimations utilisé dans notre travail délimite toutes les valeurs possibles du paramètre en question.

Les intervalles d'estimations sont utilisés pour générer des plans d'exécution présumés robustes. Dans le contexte d'évaluation de requêtes, la robustesse est généralement liée à la résistance face aux erreurs d'estimations. Un plan d'exécution est qualifié de robuste si ses performances ne se dégradent pas d'une manière significative en cas d'erreurs dans les estimations utilisées. Bien qu'il y ait eu un nombre important de travaux de recherche portant sur la robustesse dans le cadre d'évaluation de requêtes, il n'existe pas encore une définition formelle et universelle de la robustesse dans ce contexte. Récemment, Graefe et al. [Gra11] ont différencié l'*optimisation robuste* de l'*exécution robuste* de requêtes. Une optimisation robuste réfère à la capacité d'un optimiseur de générer un plan qui reste efficace pour différentes conditions d'exécution. Une exécution robuste refère à la capacité du moteur d'exécution de requêtes d'exécuter un plan spécifique d'une manière efficace dans des conditions d'exécution imprévues. Une revue sur les travaux relatifs à la robustesse dans le contexte d'évaluation de requêtes est fournie dans [YHM15].

Afin d'éviter toute confusion avec d'autres notions de la robustesse dans la littérature, nous proposons la définition suivante : *soient V_e une estimation de la valeur d'un paramètre jugée sujette à une erreur, I un intervalle défini autour de V_e représentant ainsi l'incertitude concernant la précision de cette estimation, $P_{optimal}$ le plan optimal pour une valeur $V_i \in I$, λ la sous-optimalité acceptée (exprimée en pourcentage) d'un plan P_{alt} par rapport au plan optimal. P_{alt} est dit robuste si :*

$$\forall\ V_i \in I, \frac{coût(P_{alt})}{coût(P_{optimal})} \leq 1 + \frac{\lambda}{100} \tag{1.2}$$

Par exemple, si les utilisateurs tolèrent une augmentation mineure (λ) des coûts de 20%, le coût du plan P_{alt} est au plus égal à 1.2 fois le coût de $P_{optimal}$. Dans le reste de ce document, nous nous basons sur cette définition pour qualifier un plan de robuste.

L'utilisation d'intervalles d'estimations plutôt que des valeurs singulières modélise parfaitement l'incertitude des estimations utilisées par un optimiseur. Ceci permet aussi un choix prudent d'un plan d'exécution. Les méthodes existantes faisant usages d'intervalles d'estimations supposent qu'il est toujours possible de trouver un plan robuste sur l'intégralité d'un intervalle. Cette hypothèse n'est pas fondée. Un intervalle large est signe d'une incertitude élevée, tandis qu'un intervalle étroit est signe d'une incertitude peu importante. Trouver un plan robuste sur un intervalle quand celui-ci est large n'est toujours pas évident. Des plans d'exécution peuvent être robustes seulement sur des sous-intervalles de cet intervalle. En outre, quand l'incertitude est peu importante et qu'un plan robuste ait pu être trouvé par les méthodes existantes, ce plan est généralement maintenu sans modification jusqu'à la terminaison de l'exécution. En effet, la majorité des méthodes existantes dans ce contexte, supposent que les valeurs qui seront observées à l'exécution coïncideront forcément avec les valeurs prévues à l'optimisation. Cette hypothèse peut être souvent violée. Son application peut alors induire de mauvaises performances.
Dans ce document, nous abordons ces problèmes. Nous proposons une méthode d'optimisation qui découpe -si nécessaire- un intervalle d'estimations en sous-intervalles, auxquels sont associés des plans robustes. Le plan associé au sous-intervalle le plus susceptible de contenir la valeur à l'exécution est choisi pour commencer l'exécution. Pour que notre méthode soit capable de réagir à une violation de la robustesse à l'exécution, des opérateurs de contrôle et de décision sont insérés dans le plan choisi. Ces opérateurs permettent de détecter et corriger une violation de la robustesse à l'exécution. Cela permet d'éviter une dégradation des performances due au maintien d'un mauvais choix de plan d'exécution jusqu'à la fin de l'exécution. Nous détaillons notre travail dans la section suivante.

1.3 Contributions

Dans ce document, après établir un état de l'art sur les principales méthodes existantes ayant adressé le problème d'optimisation du aux erreurs d'estimations, nous introduisons

notre méthode d'optimisation hybride. Comme mentionné précédemment, notre méthode vise deux objectifs : (1) la production de plans d'exécution robustes qui gèrent l'incertitude dans les estimations utilisées lors de l'optimisation, et (2) la détection et la correction d'une violation de la robustesse pendant l'exécution.
Notre méthode comporte deux modules. Le premier module s'intitule *génération de plans d'exécution robustes*. Il aborde la difficulté d'engendrer un plan de requête robuste sur l'intégralité d'un intervalle d'estimations utilisé. Ce module est constitué de trois éléments :

- *Sélection d'un ordonnancement robuste d'opérateurs* : nous concentrons notre travail sur l'incertitude des tailles de relations temporaires produites à des étapes intermédiaires dans un plan d'exécution. Nous utilisons des intervalles d'estimations des tailles de ces relations. L'objectif de cet élément est d'ordonnancer les opérateurs d'une requête en tenant compte de l'incertitude sur les estimations des tailles des relations opérandes.
- *Identification d'algorithmes physiques robustes* : ensuite, des algorithmes physiques sont énumérés pour chaque opérateur de la requête. Les coûts d'exécution de ces algorithmes sont comparés pour les valeurs/intervalles d'estimations des tailles des relations opérandes. L'objectif est d'identifier des algorithmes physiques robustes pour chaque opérateur.
- *Choix d'un plan d'exécution* : une fois que des algorithmes robustes sont identifiés comme candidats à l'exécution de chaque opérateur, un seul parmi ces algorithmes est ensuite sélectionné pour chaque opérateur afin de générer un plan robuste initial. Cet algorithme physique est celui qui minimise le risque d'une violation de la robustesse à l'exécution.

Le deuxième module s'intitule *correction de plans d'exécution*. Il se base sur des opérateurs de contrôle et de décision insérés dans le plan à l'optimisation. Ces opérateurs ont pour rôle de surveiller l'exécution d'un plan et déclencher une correction de celui-ci si une violation de la robustesse est détectée. Ce module se compose de deux éléments :

- *Insertion d'opérateurs de contrôle et de décision* : lorsqu'il existe un risque que

le plan établi pour commencer l'exécution n'assure pas une robustesse jusqu'à la terminaison, nous déterminons les points dans ce plan où l'incertitude est jugée importante et peut avoir un effet néfaste sur la robustesse. Des opérateurs de contrôle et de décision sont insérés dans le plan à ces points. Ils collectent des statistiques à l'exécution et déterminent si une correction du reste du plan est nécessaire.

- *Modification de plans d'exécution* : les statistiques collectées à l'exécution permettent de déterminer si le plan en cours reste robuste. Si ce n'est pas le cas, une modification est déclenchée pour le reste du plan afin de corriger la violation de robustesse constatée. Les décisions de modifications sont gérées par les opérateurs de contrôle et de décision. Ces opérateurs sont capables de réagir sans avoir besoin de ré-invoquer l'optimsieur. Ils se basent sur l'information déterminée à l'optimisation concernant les plans possibles et leurs intervalles de robustesse associés.

Ces éléments seront détaillés plus tard dans ce document. L'organisation de ce document est décrite dans la section suivante.

1.4 Organisation du manuscrit

Ce document est constitué de cinq chapitres. Après une description du contexte et de la problématique de recherche dans ce chapitre, nous présentons dans le chapitre 2 un état de l'art sur les principales méthodes d'optimisation qui ont abordé le problème d'optimisation du aux erreurs d'estimations. Nous proposons une classification de ces méthodes en deux approches, suivant la manière dont les estimations sont considérées au moment de l'optimisation, i.e., précises ou sujettes à des erreurs. Nous proposons également une classification des méthodes dans chaque approche. Nous effectuons ensuite une comparaison des méthodes dans chaque approche et terminons par une comparaison des deux approches. Dans cette comparaison, nous accordons une attention particulière à la capacité de chaque approche de gérer -d'une part- l'incertitude des estimations utilisées

à l'optimisation et d'autre part, la variation des conditions d'exécution par rapport aux conditions prévues lors de l'optimisation.

Le chapitre 3 décrit notre méthode d'optimisation hybride. Notre méthode comporte deux modules. Nous présentons d'abord le premier module qui porte sur la *génération de plans d'exécution robustes*. Nous introduisons un exemple qui souligne nos motivations pour ce travail. Puis nous détaillons le fonctionnement de ce module et des éléments qui le composent. Nous présentons ensuite le deuxième module de notre méthode, qui concerne la *correction de plans d'exécution*. Nous introduisons ce module -également- par un exemple qui souligne nos motivations. Puis nous mettons en évidence l'importance de ce module pour assurer une robustesse jusqu'à la fin des exécutions de plans de requêtes.

Le chapitre 4 est consacré à l'évaluation des performances de notre méthode. D'abord, nous décrivons notre plateforme d'expérimentations. Puis, nous évaluons les performances de notre méthodes en utilisant des requêtes de types mono et multi-jointures.

Enfin, nous établissons dans le chapitre 5 une synthèse de notre travail et décrivons des voies possibles pour la continuation et l'évolution de nos travaux de recherche.

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

La qualité d'un plan d'exécution généré par un optimiseur de requêtes est fortement dépendante de la précision des estimations produites par le modèle de coûts (Cf. Figure 1.1, Page 18). Malheureusement, ces estimations manquent souvent de précision [GSW12, LGM+15]. Ceci peut entrainer des sous-optimalités de plans d'exécution produits. Dans l'objectif de résoudre ce problème, des chercheurs ont suggéré d'améliorer la qualité des statistiques stockées dans la méta-base. Ainsi, les estimations obtenues à partir de ces statistiques deviennent plus précises. Les travaux menés dans cette perspective peuvent être classés en deux approches. La première approche inclut les méthodes qui se basent sur des techniques du domaine de la statistique pour améliorer la qualité des statistiques dans la méta-base. La deuxième approche contient les méthodes qui se basent sur l'historique des exécutions pour ajuster les statistiques dans la méta-base.

Les méthodes dans la première approche utilisent des techniques de la statistique pour construire et maintenir des statistiques plus précises dans la méta-base. Plusieurs méthodes dans cette approche sont basées sur les histogrammes. Ces derniers peuvent être construits à partir des ensembles de données synthétisées [PI97], ou en analysant les résultats d'une requête [BCG01]. Les premières méthodes proposées s'appuyant sur les histogrammes (e.g., [PHIS96, JKM+98]) supposent l'indépendance entre les valeurs d'attributs. Ainsi, elles ne sont pas capables de saisir les corrélations entre les données. Ceci peut entraîner des erreurs significatives dans les estimations des cardinalités.

Pour corriger ceci, des chercheurs ont proposé des méthodes d'optimisation dans lesquelles l'estimation des cardinalités est basée sur des techniques de modélisation multidimensionnelle. Par exemple, dans [PI97, LKC99, BCG01], des histogrammes multidimensionnels sont présentés. Ces histogrammes permettent une meilleure estimation des cardinalités pour les requêtes évoquant plusieurs attributs d'une même relation. Cependant, même en utilisant cette variante d'histogrammes, des corrélations entre des données peuvent ne pas être détectées. Ceci est du à l'hypothèse souvent utilisée selon laquelle les données dans chaque classe d'un histogramme sont uniformément réparties [WNS16].

Ce problème a été adressé dans [DGR01, GTK01, TDJ11, TDJ13]. Dans ces travaux, les auteurs ont proposé des modèles graphiques probabilistes qui évitent cette hypothèse. Ces modèles permettent de visualiser d'éventuelles corrélations entre attributs et de modéliser la répartition de données. Bien que ces travaux permettent d'augmenter la précision des estimations, obtenir des estimations précises reste complexe. En effet, les estimations des tailles des relations temporaires présentent souvent un degré élevé d'imprécision du à la nature multidimensionnelle des statistiques utilisées [BC05, LGM+15].

Une solution alternative proposée dans [WCHN13, HCW+13, WWHN14] pour améliorer la qualité des estimations, se base sur la technique de l'échantillonnage. Dans [WCHN13, HCW+13, WWHN14], les auteurs utilisent des échantillons de données afin d'obtenir de meilleures prévisions des temps d'exécution de requêtes. Comme ces méthodes considèrent un nombre fini d'échantillons, il est possible que ces échantillons ne soient pas représentatifs des paramètres en question. Ainsi, les valeurs -à l'exécution- de ces paramètres peuvent être erronées par rapport aux valeurs estimées.

Motivé par la capacité limitée des méthodes dans cette approche de fournir des estimations précises, des chercheurs ont suggéré d'utiliser l'historique d'exécutions pour améliorer la qualité des statistiques. En effet, recueillir des informations à partir d'une exécution en cours peut aider à ajuster les statistiques stockées dans la méta-base. Ce principe a été initié dans [CR94], où les auteurs ont introduit une méthode appelée « Adaptive Selectivity Estimation using Query Feedback ». Dans cette méthode, une exécution en cours est surveillée. Les sélectivités observées sont comparées à celles estimées lors de l'optimisation (Cf. Figure 2.1, Page 31). Cette comparaison permet de mettre à jour des statistiques pour une meilleure optimisation de la même requête ou d'une requête future similaire. Cette solution a également été adoptée dans [SLMK01] sous forme de « LEarning Optimizer » (LEO). LEO contrôle les cardinalités pendant l'exécution. Les cardinalités observées sont comparées à celles estimées. La différence sert de facteur d'ajustement des statistiques pour des requêtes futures similaires. L'idée clé dans [CR94] et [SLMK01] est de compter le nombre de tuples produits par chaque opérateur dans un plan d'exécution. Cette infor-

mation est enregistrée dans la méta-base. Cette dernière sera consultée par l'optimiseur de requêtes lors de l'optimisation d'une requête future.

Plus tard, l'utilisation d'historique d'exécutions a été adopté dans le contexte de traitement de données massives [AKB+12, BAK+12]. La méthode RoPE (Reoptimizer for Parallel Executions) proposée dans [AKB+12] consiste à collecter des statistiques pendant l'exécution de jobs. Un *job* est un programme Map-Reduce qui peut contenir un ou plusieurs opérateurs de jointure [AU10, WLMO11]. Dans un programme Map-Reduce, chaque exécution est réalisée à l'aide de deux instructions appelées *Map* et *Reduce* sur un ensemble de données. Les statistiques recueillies lors de l'exécution d'un job sont utilisées pour ré-optimiser (e.g., changer le degré de parallélisme, ré-ordonnancer les opérateurs) des invocations futures du même job ou d'un job similaire.

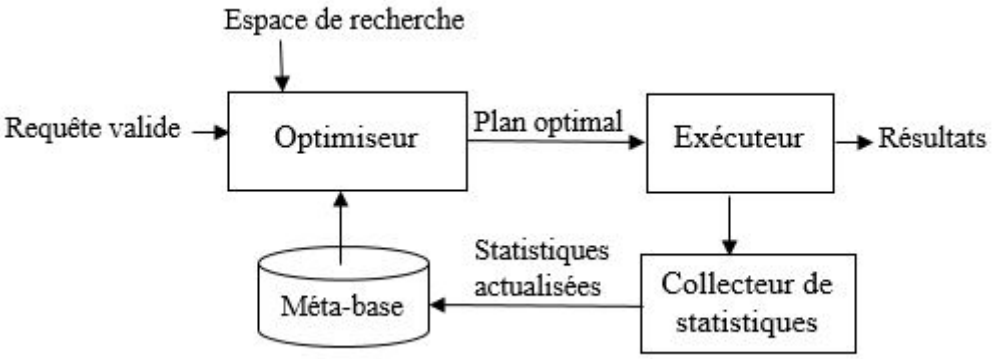


FIGURE 2.1 – Optimisation de requêtes basée sur l'historique

Dans ces méthodes, les informations collectées pendant l'exécution d'une requête sont limitées aux informations pouvant être obtenues en examinant les sorties des opérateurs d'un plan en cours d'exécution. Dans l'objectif de rendre ces informations beaucoup plus riches, Chaudhuri et al. [CNR08] ont proposé une méthode d'optimisation appelée « Pay-As-You-Go ». Dans cette méthode, une requête peut payer un sur-coût supplémentaire sur chaque exécution pour que la qualité des plans de futures invocations de la requête -ou de requêtes similaires- soit améliorée. Ce sur-coût toléré est spécifié par l'utilisateur. Le sur-coût est du à une étape supplémentaire introduite dans le processus d'optimisation de

requêtes, dite *modification de plans*. Durant cette étape, un plan d'exécution initialement produit par un optimiseur de requêtes est modifié (e.g., ré-ordonnancement d'opérateurs) afin d'obtenir des informations supplémentaires. Cette modification est effectuée tout en respectant la contrainte spécifiée par l'utilisateur sur le sur-coût toléré de cette étape. La décision de modification d'un plan dépend des informations supplémentaires sur les prédicats dont l'optimiseur a besoin.
Les méthodes d'optimisation basée sur l'historique décrites précédement peuvent être bénéfiques pour les requêtes récurrentes/similaires. Cependant, leurs performances peuvent être limitées pour les requêtes had-hocs puisque seulement les requêtes futures peuvent bénéficier de statistiques actualisées. Récemment, Wu et al. [WNS16], ont proposé une méthode d'optimisation qui permet d'améliorer les performances des requêtes ad-hocs. Dans [WNS16], les auteurs ont introduit une étape dans le processus d'optimisation de requêtes, appelée *validation de plan basée sur l'échantillonnage*. Durant cette étape, un plan d'exécution produit par un optimiseur de requêtes est examiné afin de détecter les points dans ce plan où une erreur dans les estimations des cardinalités est susceptible d'exister. Pour cela, le plan est testé sur un échantillon de données. Si aucune erreur n'est détectée, le plan est validé et l'exécution de ce plan peut commencer. Dans le cas contraire, ce plan ainsi que les estimations actualisées via l'échantillonnage sont retournés à l'optimiseur afin qu'il produise un nouveau plan. Ce nouveau plan sera à nouveau examiné. S'il n'est pas validé, il sera retourné à l'optimiseur. L'étape de validation d'un plan est répétée jusqu'à ce que le nouveau plan produit soit validé.

Les méthodes présentées ci-dessus permettent d'améliorer la qualité des statistiques stockées dans la méta-base. Cependant, obtenir des statistiques précises reste très difficile. En effet, une corrélation entre des attributs de deux relations n'est généralement pas détectée, ce qui induit une erreur d'estimation importante. De plus, les statistiques utilisées à l'optimisation peuvent évoluer pendant l'exécution, et devenir alors invalides [Chr84, PI97, GSW12].

Poussé par la complexité d'avoir des estimations précises lors de l'optimisation, plusieurs chercheurs ont concentré leurs travaux sur de nouvelles approches et méthodes d'optimisation capables de gèrer les erreurs d'estimations. Leur objectif commun est d'éviter une dégradation des performances due à des erreurs d'estimations. Nous avons étudié plusieurs méthodes proposées dans ce contexte et avons identifié deux approches principales. Une première approche, que nous appelons *optimisation basée sur des estimations en points singuliers*, consiste à étendre l'approche d'optimisation traditionnelle en entrelaçant les phases d'optimisation et d'exécution, éventuellement, plusieurs fois pendant l'exécution d'une requête. Les erreurs d'estimations peuvent être détectées en surveillant l'exécution. La sous-optimalité d'un plan peut être corrigée en ré-invoquant l'optimiseur à l'exécution pour ré-optimiser le reste du plan. Babu et al. [BB05] qualifient les méthodes proposées dans le cadre de cette approche de réactives car elles utilisent, d'abord, un optimiseur traditionnel pour générer un plan optimal. Puis pendant l'exécution, des statistiques sont collectées pour vérifier si une erreur d'estimation existe. Dans le cas d'erreur dans les estimations, une correction de la sous-optimalité constatée est déclenchée. Les performances de cette approche peuvent être limitées puisqu'à la phase d'optimisation, aucune information sur l'incertitude des estimations utilisées n'est fournie. L'optimiseur peut choisir un plan dont les performances dépendent d'estimations sujettes à des erreurs du fait qu'elles aient pu être obtenues par l'utilisation d'hypothèses invalides ou de statistiques obsolètes. En outre, ces méthodes ne sont pas capables de collecter des statistiques rapidement et avec précision lors de l'exécution [BB05]. La ré-optimisation peut être déclenchée avec des estimations à nouveau erronées. Ceci est susceptible de conduire à d'autres ré-optimisations d'un plan et par conséquent un sur-coût éventuellement important.

Dans l'objectif d'éviter un sur-coût relatif à plusieurs ré-optimisations potentielles d'un plan d'exécution, des chercheurs ont introduit une approche alternative, que nous appelons *optimisation basée sur des estimations en un ensemble de points*. Cette approche consiste à anticiper la réaction à une sous-optimalité éventuelle d'un plan d'exécution. Les méthodes proposées dans cette approche considèrent la possibilité d'erreurs dans les

estimations dès la phase initiale d'optimisation. L'incertitude concernant les estimations utilisées pour générer un plan est modélisée soit par des distributions de probabilités, soit par des intervalles d'estimations autour des valeurs possibles des paramètres en question. L'optimiseur a pour rôle de trouver le plan capable d'offrir des performances robustes pour les différentes valeurs possibles des paramètres, plutôt que le plan fournissant des performances optimales seulement si les estimations utilisées sont exactes.

Dans le reste de ce chapitre, nous présentons les principales méthodes d'optimisation proposées dans chacune de ces deux approches. Dans un premier temps, nous présentons une description de ces méthodes. Puis, nous les comparons suivant un ensemble de critères que nous détaillons. Pour conclure, nous présentons les points forts et les limites de chaque approche.

2.2 Approche d'optimisation basée sur des estimations en points singuliers

Les méthodes proposées dans cette approche font usage de valeurs singulières d'estimations. Pendant la phase initiale d'optimisation, l'optimiseur considère que les estimations sont précises. Cette approche peut être décrite comme suit (Cf. Figure 2.2, Page 35) : un optimiseur traditionnel est d'abord utilisé pour produire un plan optimal par rapport à un modèle de coûts. A l'exécution, des statistiques actualisées sont recueillies et comparées à celles utilisées par l'optimiseur. Si la différence est significative, alors l'exécution en cours est interrompue. Les statistiques actualisées sont utilisées pour ré-optimiser le reste du plan. A chaque invocation, l'optimiseur associe des valeurs spécifiques aux paramètres nécessaires aux calculs de coûts. Les plans d'exécution choisis sont alors optimaux pour des conditions d'exécution particulières.

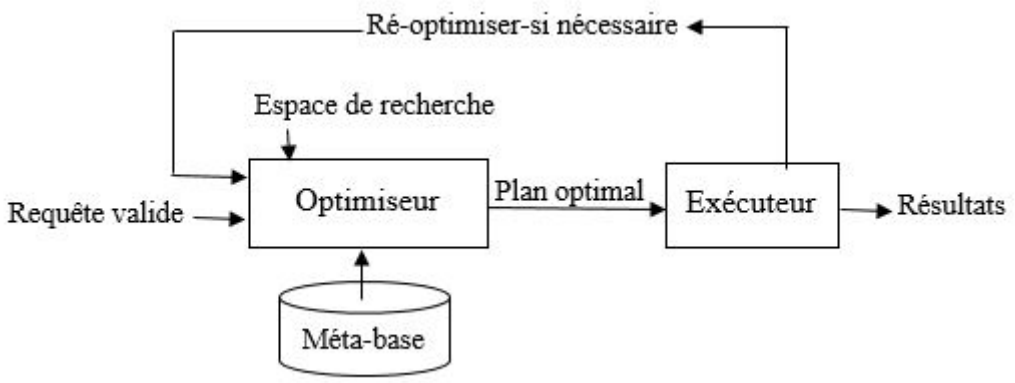


FIGURE 2.2 – Approche d'optimisation basée sur des estimations en points singuliers

Cette approche d'optimisation permet de détecter les erreurs d'estimations et corriger les sous-optimalités qui en résultent en modifiant le reste d'un plan durant l'exécution. Les phases d'optimisation et d'exécution peuvent être entrelacées, éventuellement, plusieurs fois lors de l'exécution d'une requête.
Les méthodes dans cette approche divergent sur la fréquence à laquelle elles peuvent modifier un plan d'exécution. Cette modification peut se produire soit entre les opérateurs soit lors de l'exécution d'un opérateur. Dans le reste de cette section, nous proposons de classifier les méthodes dans cette approche selon ce critère, i.e., granularité de modifications de plans d'exécution. Nous distinguons deux classes de méthodes : (1) des méthodes basées sur une correction de plans au niveau inter-opérateurs, et (2) des méthodes basées sur une correction de plans au niveau intra-opérateurs. Le reste de cette section est consacré à la description de ces deux classes de méthodes.

2.2.1 Méthodes basées sur une correction au niveau inter-opérateurs

Dans cette classe de méthodes, un optimiseur découpe un plan d'exécution d'une requête en plusieurs étapes, dites sous-plans. Un sous-plan peut contenir un ou plusieurs opérateurs. Les résultats intermédiaires produits par un sous-plan sont matérialisés puis utilisés comme entrée pour le sous-plan suivant. La collecte de statistiques peut être déclenchée après l'évaluation de certains sous-plans ou après un évènement particulier

comme l'arrivée de résultats partiels à partir des SGBDs locaux dans un environnement multi-bases (e.g., MIND [EDNO97]).

Pour décrire ce principe d'une manière plus précise, nous prenons comme exemple une requête de jointure entre trois relations R, S, et T. Supposons que le plan P_0 (Cf. Figure 2.3) est initialement sélectionné pour exécuter cette requête. Supposons qu'à l'exécution du premier sous-plan O_1, une erreur d'estimation est détectée. Comme illustré dans la figure 2.3, la réaction à cette erreur se produit après terminaison du sous-plan en cours $\mathbf{O_1}$. Le plan est ré-optimisé et l'exécution de la requête reprend avec un nouveau plan.

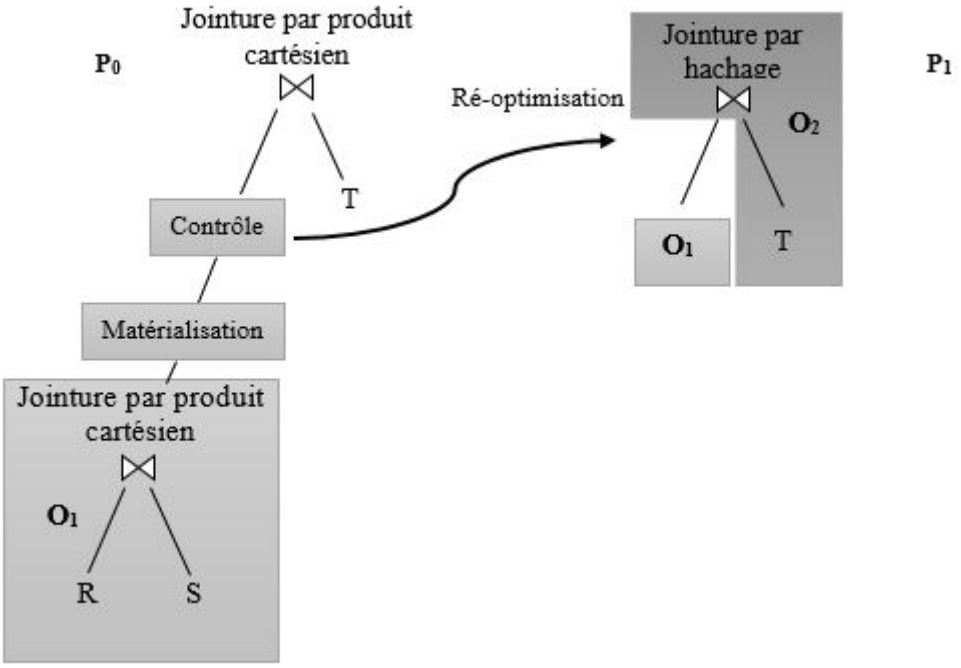


FIGURE 2.3 – Correction d'un plan d'exécution au niveau inter-opérateurs

La littérature fournit plusieurs méthodes s'inscrivant dans cette classe de méthodes. Chacune des méthodes proposées a été conçue dans l'objectif de corriger des sous-optimalités causées par un ou des événements spécifiques, et chacune propose une ou des corrections possibles. Par exemple, Ozcan et al. [EDNO97] se sont concentrés sur l'indisponibilité de statistiques nécessaires pour pouvoir produire un plan d'exécution optimal. Ils ont introduit, dans le cadre du système multi-bases MIND, des stratégies de ré-ordonnancement

dynamique d'opérateurs inter-sites (e.g., jointure). Ces stratégies exploitent les résultats partiels disponibles pour détecter les erreurs d'estimations et corriger un plan d'exécution. Cette correction consiste à ré-ordonnancer les opérateurs d'un plan pendant l'exécution. Dans une première étape, MIND décompose une requête en sous-requêtes. Celles-ci sont ensuite envoyées aux différents sites pour être exécutées en parallèle. A l'exécution, des statistiques sur les résultats intermédiaires retournés par les SGBDs locaux sont utilisées pour définir l'ordre d'exécution des opérateurs dépendant de ces résultats.
Dans le même esprit, Neumann et al. [NGl13] ont proposé une méthode (CIE) qui vise à résoudre le problème d'optimisation de requêtes du à l'imprécision des estimations des cardinalités. Pour cela, les auteurs se sont basés sur le principe d'exécution incrémentale. Tout d'abord, un plan d'exécution optimal est construit en utilisant un optimiseur traditionnel. Puis, ce plan est découpé en sous-plans : les fragments sensibles du plan sont identifiés, ce sont les fragments où des erreurs d'estimations des cardinalités pourraient conduire à des sous-optimalités du reste du plan. CIE évalue la propagation d'erreurs éventuelles à ces fragments dans le reste du plan. Lorsque cette propagation dépasse un certain seuil, elle est considérée fatale pour le reste du plan. Ensuite, ces fragments sont exécutés, les résultats sont matérialisés. La cardinalité requise est obtenue. Un nouveau plan optimal peut ainsi être trouvé pour le reste de la requête en déclenchant une ré-optimisation.

Matérialiser les résultats d'un sous-plan et les utiliser comme entrée pour le sous-plan suivant a également été proposé dans [KD98]. Dans ce travail, Kabra et DeWitt [KD98] ont introduit une méthode appelée « Mid-Query Re-optimization ». Cette méthode permet de détecter et de corriger la sous-optimalité d'un plan pendant l'exécution. Les auteurs ont adressé le problème de sous-optimalités de plans provenant d'un mauvais ordonnancement d'opérateurs, d'un choix inapproprié des algorithmes de jointure ou d'une mauvaise allocation de la mémoire. Ces anomalies seraient causées par des estimations de coûts erronées ou obsolètes, ou par un manque d'informations nécessaires relatives aux conditions d'exécution au moment de l'optimisation. Dans cette méthode [KD98], un optimiseur tra-

ditionnel est d'abord utilisé pour générer un plan optimal. Ce plan est ensuite annoté par les statistiques utilisées par l'optimiseur. Pendant l'exécution, des statistiques actualisées sont collectées. Celles-ci sont comparées à celles utilisées pour choisir le plan. Si elles y sont différentes d'une manière significative (i.e., la différence dépasse un certain seuil), l'optimiseur est invoqué pour ré-optimiser le reste du plan. Les statistiques sont collectées par des opérateurs de collecte de statistiques, insérés dans le plan à des points dits points de contrôle. Un opérateur de collecte de statistiques est un opérateur unaire qui sert à contrôler les tuples qui le traversent. La méthode d'insertion de ces opérateurs associe un niveau d'incertitude (faible, moyen, élevé) aux différentes estimations. Seules les estimations affectant la qualité d'un plan sont vérifiées à l'exécution.

Markl et al. [MRS+04] ont étendu la méthode dans [KD98] sous forme de «Progressive Query Optimization»(POP). Dans cette méthode, un plan optimal généré par un optimiseur de requêtes est également enrichi par des opérateurs de contrôle insérés dans le plan à des points clés. Afin de minimiser le besoin de ré-optimisation, POP associe à chaque opérateur de contrôle, un intervalle définissant les cardinalités pour lesquelles le plan choisit par l'optimiseur reste optimal. A l'exécution, des statistiques sont collectées par les opérateurs de contrôle. Ces derniers surveillent le nombre de tuples produits et vérifient la validité du plan en cours pour les cardinalités observées. Si les cardinalités observées sont dans l'intervalle de validité, l'exécution du plan se poursuit normalement. Dans le cas contraire, l'exécution de l'opérateur en cours se termine et une ré-optimisation est déclenchée pour le reste du plan.

Les travaux dans [KD98] et [MRS+04] sont proches de ceux de Ives et al. [IFF+99] puisque ces derniers ont proposé également de ré-optimiser le reste d'un plan d'exécution d'une requête lorsqu'une sous-optimalité est détectée. Dans la méthode d'optimisation proposée par Ives et al. [IFF+99], comme dans celle de Kabra et DeWitt [KD98], un plan d'exécution optimal est produit en utilisant un optimiseur traditionnel. Il est ensuite annoté par les différentes estimations utilisées. L'exactitude de ces estimations est vérifiée à l'exécution. Si ces estimations s'avèrent inexactes, une modification du reste du plan est

déclenchée. Contrairement aux méthodes dans [KD98, MRS+04] où la modification d'un plan d'exécution est décidée par l'optimiseur, cette méthode [IFF+99] utilise des opérateurs adaptatifs qui décident de la modification à apporter à un plan. Ces opérateurs se basent sur un ensemble de règles de type « Évènement-Condition-Action ». Ces règles sont utilisées pour déterminer le comportement d'un plan d'exécution en fonction des changements dans les conditions d'exécution. Les actions possibles sont : ré-allocation de la mémoire, ré-ordonnancement d'opérateurs, remplacement d'opérateurs et ré-optimisation du reste d'un plan d'exécution. Ces actions permettent de corriger les sous-optimalités de plans dues à une estimation erronée de la quantité de mémoire disponible et/ou du délais d'arrivée de données.

Amsaleg et al. [MFTU96] se sont également intéréssés aux erreurs d'estimations dues aux délais d'arrivées de données. Amsaleg et al. [MFTU96] ont proposé une méthode appelée « Query Scrambling ». Celle-ci a été développée dans l'objectif de traiter les blocages causés par des retards d'arrivées de données qui peuvent survenir lors de l'exécution de requêtes dans un environnement distribué. Un optimiseur traditionnel est initialement utilisé pour générer un plan d'exécution optimal. Puis, si l'exécution de ce plan est bloquée en raison de retards d'arrivée des tuples à partir de sources de données distantes, Query Scrambling tente de masquer ces retards soit par : (i) un ré-ordonnancement ; elle change l'ordre d'exécution des opérateurs pour éviter le temps d'inactivité, ou bien par (ii) un remplacement d'opérateurs ; dans ce cas, Query Scrambling introduit de nouveaux opérateurs à exécuter en cas de blocage d'autres opérateurs. La forme initiale du plan de la requête peut être modifiée par l'ajout ou/et la suppression d'opérateurs.

Comme Amsaleg et al. [MFTU96], Bruno et al. [BJZ13] ont adressé le problème d'erreurs d'estimations des données de nature distribuée. Dans [BJZ13], Bruno et al. ont proposé dans le cadre du système SCOPE, une stratégie d'optimisation appelée « Continuous Cloud Scale Query Optimization ». Cette stratégie permet d'adapter un plan d'exécution aux conditions actuelles dans un environnement d'exécution. Elle permet de réagir aux erreurs d'estimations liées à l'absence/obsolescence de statistiques sur les données en rai-

son de volumes massifs de données et de leur nature distribuée. Cette stratégie consiste à surveiller l'exécution d'une requête. Ainsi, des statistiques actualisées sont obtenues. Ces statistiques permettent de décider si une modification du reste du plan est nécessaire et si celle-ci est utile, en termes de coûts. La modification d'un plan en cours consiste à changer le degré du parallélisme.

Dans les méthodes décrites ci-dessus, le choix initial d'un plan d'exécution est fondé -exclusivement- sur les statistiques disponibles dans la méta-base. Dans l'objectif d'éviter les erreurs dans les décisions d'optimisation initiale, Karanasos et al. [KBK+14] ont proposé une méthode d'optimisation qui repose sur ce qu'ils ont appelé *des essais expérimentaux*. La méthode d'optimisation (DyOpt) dans [KBK+14] commence par identifier les opérateurs locaux d'une requête en entrée. Un opérateur est dit local s'il se réfère uniquement aux attributs d'une relation, tels que les opérateurs de sélection et de projection. Ensuite, des *essais expérimentaux* sont lancés pour exécuter ces opérateurs locaux, chacun sur un échantillon de données. En d'autres termes, une partie de la requête est exécutée. Les résultats produits permettent d'obtenir des statistiques actualisées et ainsi de meilleures estimations. L'optimiseur de requêtes s'appuie sur ces résultats pour choisir un plan d'exécution pour le reste de la requête. L'optimiseur choisit un ordonnancement d'opérateurs ainsi que les algorithmes de jointures appropriés. Ensuite des *jobs* sont générés pour exécuter le plan. Un *jobs* peut contenir un ou plusieurs opérateurs de jointure [AU10, WLMO11]. Lors de l'exécution, des statistiques sur les résultats intermédiaires sont recueillies. Si la différence entre les cardinalités observées et celles estimées dépasse un certain seuil, une ré-optimisation du reste du plan est déclenchée. La ré-optimisation peut s'effectuer après chaque *job*. Dans un contexte MapReduce, un job définit l'unité d'exécution. Ainsi, une modification inter-jobs est une modification inter-opérateurs.

Bien qu'elles diffèrent sur les événements auxquels elles réagissent et les réactions qu'elles offrent, les méthodes d'optimisation présentées dans cette sous-section possèdent un critère en commun ; elles permettent de modifier des plans d'exécution sous-optimaux après

la terminaison d'opérateurs. Une autre solution proposée dans la littérature consiste à modifier les plans d'exécution pendant l'exécution d'opérateurs. La sous-section suivante est réservée pour décrire les principales méthodes proposées dans ce contexte.

2.2.2 Méthodes basées sur une correction au niveau intra-opérateurs

Les méthodes dans cette classe affinent la granularité de la modification de plans d'exécution. Elles permettent de corriger un plan pendant l'exécution d'un opérateur. Les statistiques sont collectées après qu'un bloc de tuples soit traité. Un bloc peut être constitué d'un ou plusieurs tuples. Dans cette perspective, les méthodes suivantes ont été proposées : le routage de tuples [AH00], et le partitionnement dynamique de données [IHW04].

Le routage de tuples a été le principe du travail présenté par Avnur et al. [AH00]. Dans [AH00], les auteurs ont proposé un mécanisme d'évaluation de requêtes nommé «Eddy». Eddy réordonne en continu les opérateurs d'une requête afin de s'adapter dynamiquement aux variations des caractéristiques de données (coût d'un opérateur, sélectivités d'opérateurs, taux d'arrivé des tuples). Eddy peut être considéré comme un routeur de tuples positionné entre des relations et des opérateurs. Chaque opérateur doit avoir une ou deux files d'attentes d'entrée pour recevoir les tuples envoyés par Eddy et une file d'attente de sortie pour retourner les tuples résultats à Eddy. Les tuples reçus par Eddy sont acheminés vers les opérateurs dans des ordres différents. Eddy sélectionne automatiquement un ordre pour acheminer chaque tuple. Pour illustrer ceci, considérons la requête $Q = \sigma(R) \bowtie \sigma(S) \bowtie \sigma(T)$. Un opérateur Eddy, deux opérateurs de jointure et trois opérateurs de sélection sont nécessaires pour exécuter cette requête. Eddy exécute Q en routant les tuples reçus à partir des relations R, S et T, vers les opérateurs de sélection et de jointure. Lorsqu'un opérateur reçoit un tuple de Eddy, il y applique le prédicat correspondant et retourne les résultats à Eddy. Ce dernier transmet les résultats aux opérateurs restant pour la suite de l'exécution.

Les performances dans le mécanisme Eddy sont fortement dépendantes de la stratégie de

routage. Celle-ci se réfère à l'ensemble de règles utilisées pour choisir -pour un tuple- la destination valide. La stratégie de routage doit être efficace et intelligente afin de minimiser le temps de réponse. Par exemple, Eddy ne doit pas acheminer les tuples de R à l'opérateur de sélection σ(S). La principale limite dans Eddy, c'est que les algorithmes de routage sont des algorithmes gloutons. Ces algorithmes opèrent d'une manière itérative. A chaque itération, ils minimisent une fonction partielle des coûts en choisissant un optimal local. Ceci ne permet toujours pas d'obtenir une optimalité globale [BB05]. De plus, Eddy doit explorer plusieurs routes pour trouver le plan adéquat. Ceci peut induire un sur-coût important [BB05].

Dans la même perspective qu'Eddy [AH00], à savoir obtenir un résultat global à partir de résultats partiels, Ives et al. [IHW04] ont proposé une méthode d'optimisation basée sur le partitionnement dynamique de données. Cette méthode vise à corriger les sous-optimalités de plans causées par des erreurs d'estimations dues au manque d'informations sur les sources de données lors de l'optimisation. Comme dans Eddy [AH00], la correction d'un plan se fait pendant l'exécution d'un opérateur. Cependant, le principe de fonctionnement est différent. Dans [IHW04], un ensemble de plans d'exécution est associé à chaque requête. Ces plans seront exécutés en parallèle ou en séquence sur des partitions de données distinctes. Les décisions sur la façon de répartir les données dépendent de nombreux facteurs, par exemple, la sémantique de chaque opérateur, les statistiques disponibles sur les données. L'exécution d'un plan associé à une requête est contrôlée. Ce plan peut être remplacé par un nouveau plan s'il s'avère sous-optimal. Les tuples traités par chaque plan représentent un partitionnement de données. Quand un plan d'exécution est remplacé, un nouveau partitionnement de données est produit. L'union des tuples produits par les différents plans ne fournit qu'une partie du résultat global. Ainsi, pour calculer le résultat global d'une requête, il faut calculer les résultats des combinaisons des divers partitionnements de données. Pour illustrer ceci, nous fournissons un exemple d'une requête Q de jointure entre trois relations R, S et T. Supposons qu'en se basant sur des statistiques disponibles, un plan P0 est d'abord généré pour $Q = (R \bowtie S) \bowtie T$. Lors de l'exécution,

des statistiques actualisées sont recueillies. Supposons que l'optimiseur se rend compte que P0 est sous-optimal et qu'il serait préférable de remplacer P0 par un plan P1 pour $Q = R \bowtie (S \bowtie T)$. Plutôt que de redémarrer la requête en utilisant P1, l'optimiseur suspend le plan partiellement exécuté, à savoir P0 qui a déjà retourné des résultats. P0 représente la phase 0 de l'exécution. Les tuples traités dans cette phase sont dénotés R^0, S^0 et T^0. P1 représente la 1ère phase. Les tuples traités dans cette phase sont dénotés R^1, S^1 et T^1. Après exécution de P0 et P1, le résultat de la requête est calculé comme suit :

$$Q = (R^0 \bowtie S^0 \bowtie T^0) \bigcup (R^1 \bowtie S^1 \bowtie T^1) \bigcup (R^0 \bowtie S^0 \bowtie T^1) \bigcup (R^0 \bowtie S^1 \bowtie T^0) \bigcup (R^0 \bowtie S^1 \bowtie T^1) \bigcup (R^1 \bowtie S^0 \bowtie T^0) \bigcup (R^1 \bowtie S^1 \bowtie T^0) \bigcup (R^1 \bowtie S^0 \bowtie T^1) \quad (2.1)$$

Dans cette méthode, comme dans toutes les méthodes précédemment décrites dans l'approche d'optimisation basée sur des estimations en points singuliers, l'objectif est d'assurer des performances optimales dans des circonstances d'exécution spécifiques. Ces circonstances sont modélisées par des estimations singulières des paramètres nécessaires aux calculs des coûts. Pour atteindre leur objectif, les méthodes dans cette approche s'appuient sur un comportement réactif. Quand une sous-optimalité d'un plan d'exécution est détectée, l'optimiseur de requêtes est invoqué pour corriger cette sous-optimalité. Une correction décidée par l'optimiseur peut être fondée sur des estimations à nouveau erronées. Par conséquent, l'optimiseur peut être ré-invoqué plusieurs fois durant l'exécution d'une requête. Ceci peut induire un sur-coût important et une dégradation des performances.
Une approche alternative a été proposée pour éviter ce problème. Celle-ci vise la prévention plutôt que la correction de sous-optimalités de plans d'exécution. La possibilité d'erreurs dans les estimations utilisées par l'optimiseur est tenue en compte dès la phase initiale d'optimisation. Pour modéliser ces erreurs potentielles, cette approche alternative fait usage de plusieurs points plutôt qu'un point singulier pour estimer chaque paramètre présumé incertain. Cette approche est appelée optimisation basée sur des estimations en

un ensemble points. Elle est détaillée dans la section suivante.

2.3 Approche d'optimisation basée sur des estimations en un ensemble de points

Contrairement à la première approche qui ignore la possibilité d'erreurs d'estimations utilisées par l'optimiseur, cette approche considère les éventuelles erreurs dans les estimations dès le choix initial d'un plan d'exécution. L'incertitude concernant la valeur d'un paramètre est modélisée soit par une distribution de probabilités soit par un intervalle d'estimations contenant les valeurs possibles de ce paramètre.
Les méthodes dans cette approche se basent sur un comportement proactif dans le sens où elles anticipent la réaction à une sous-optimalité possible. La majorité des méthodes sont dotées d'un comportement totalement proactif. Seulement quelques méthodes ont un comportement à la fois proactif et réactif. La figure 2.4 ci-dessous illustre ceci. La ligne discontinue s'applique uniquement lorsque la méthode interagit avec l'environnement d'exécution, i.e., elle collecte des statistiques et peut déclencher une ré-optimisation.

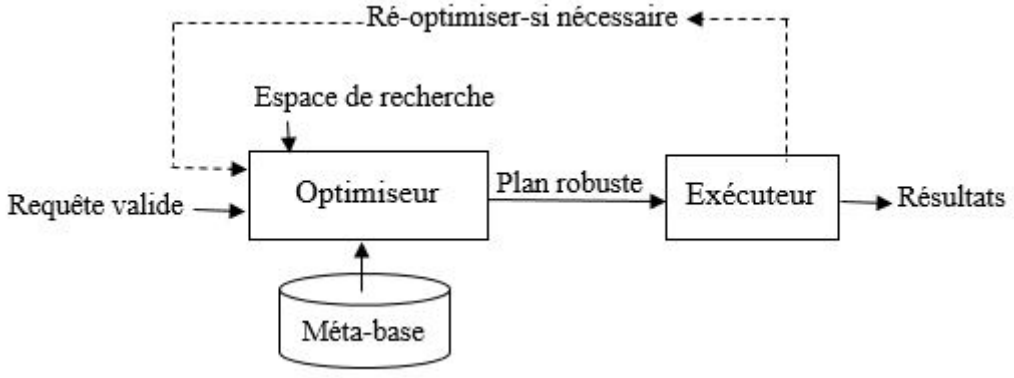


FIGURE 2.4 – Approche d'optimisation basée sur des estimations en un ensemble de points

Les méthodes proposées dans cette approche visent la robustesse plutôt que l'optimalité. Elles cherchent à engendrer des plans d'exécution fournissant des performances qui

restent stables lorsque les estimations utilisées lors de l'optimisation sont différentes des valeurs observées à l'exécution. Pendant la phase d'optimisation, les coûts de plans possibles pour une requête sont comparés sur l'ensemble des valeurs des paramètres présumés incertains. L'étape d'après consiste à choisir le plan adéquat pour commencer l'exécution. Après une étude de méthodes proposées dans le cadre de cette approche, nous avons distingué deux classes de méthodes : (1) des méthodes basées sur un seul plan d'exécution dans lesquelles -au moment de l'optimisation- un seul plan est généré pour une requête donnée (notons que lors d'une ré-optimisation, ce plan peut être modifié ou remplacé par un nouveau plan), et contrairement à celles-ci (2) des méthodes basées sur un ensemble de plans d'exécution dans lesquelles -au moment de l'optimisation- plusieurs plans sont générés pour une même requête.
Nous soulignons que cette classification est indépendante du comportement des méthodes à l'exécution, i.e., elles considèrent ou pas la ré-optimisation. Notre classification se concentre sur le comportement des méthodes pendant la phase initiale d'optimisation. Nous détaillons ces deux classes de méthodes dans le reste de cette section.

2.3.1 Méthodes basées sur un seul plan d'exécution

Une solution pour tenir compte de l'incertitude concernant la précision des estimations disponibles à l'optimisation, est de considérer une distribution de probabilités pour chaque paramètre présumé incertain. Une distribution de probabilités représente les valeurs possibles d'un paramètre, à l'exécution.
Dans ce contexte, Chu et al. [CHG02] ont proposé une méthode d'optimisation, appelée «Least Expected Cost Optimization»(LEC). LEC consiste à trouver -pour une requête- le plan d'exécution estimé le moins coûteux pour les différentes valeurs que les paramètres du coûts peuvent prendre lors de l'exécution. Pour une requête donnée, LEC suppose qu'il existe une mesure de la probabilité P sur un espace V des valeurs possibles des paramètres présumés incertains. Ces paramètres décrivent les données (e.g., cardinalités, distributions de valeurs d'un attribut), les composants de requêtes (e.g., sélectivités des prédicats) et

l'environnement d'exécution (e.g., quantité de mémoire disponible, charge CPU). LEC suppose également qu'il possède une fonction coût, notée C(p,v). C(p,v) prend en entrée deux paramètres : un plan p et un vecteur v représentant les valeurs des paramètres. C(p,v) représente le coût d'exécution du plan p tel que les paramètres incertains prennent comme valeurs le vecteur v $\in$ V. Ayant C(p,v) et P, le coût estimé de chaque plan p associé à une requête, est calculé comme suit :

$$LEC(p) = \sum_{v \in V} C(p, v) \times P(v) \tag{2.2}$$

L'optimiseur choisit ensuite le plan ayant le coût estimé minimal. LEC suppose que les distributions des valeurs des paramètres sont à priori disponibles. Ceci n'est pas toujours possible, particulièrement lorsque les corrélations de données sont complexes ou que les conditions d'exécution sont peu prévisibles. De plus, LEC suppose l'indépendance des distributions des probabilités des différents paramètres. Cette hypothèse est souvent violée. En effet, la disponibilité en mémoire et la charge de travail sont souvent liées. Quand la charge du travail augmente, la quantité de mémoire disponible diminue. Une application injustifiée de cette hypothèse est la principale source d'erreurs dans les estimations [IC91, PI97]. Il peut en résulter une sous-optimalité du plan d'exécution choisi. LEC ne considère pas la collecte des statistiques ni la ré-optimisation pendant l'exécution.
Le travail de Babcock et al. [BC05] est similaire à celui de Chu et al [CHG02] puisque Babcock et al. [BC05] ont proposé une méthode d'optimisation dans laquelle le processus d'estimation des cardinalités utilise également des distributions de probabilités. Cette méthode est appelée « Robust Query Optimizer ». En première étape, une distribution des probabilités sur les cardinalités ayant un impact significatif sur les performances de plans d'exécution associés à une requête, est obtenue. La distribution de probabilités est ensuite représentée par une fonction de densité de probabilités. Celle-ci permet de modéliser la dépendance entre les coûts des plans possibles pour une requête et les différentes valeurs de cardinalités. La deuxième étape consiste à extraire une valeur unique à partir de la distribution des probabilités. Le choix de cette valeur dépend de l'importance qu'accorde

l'utilisateur au *coûts estimés* par rapport à la *prévisibilité des coûts*. Le compromis souhaité entre ces paramètres est exprimé par un seuil de confiance, noté T. T est spécifié par l'utilisateur. T% définit la valeur percentile de la distribution de probabilités à considérer pour choisir un plan d'exécution. Un seuil égale à T indique qu'un plan est engendré tel que l'optimiseur est T% confiant que le coût réel de ce plan ne dépassera pas le coût estimé. T peut prendre une parmi trois valeurs : conservateur, modéré ou agressif. Ces valeurs correspondent respectivement à T égale à 50, 80, et 95.

Rendre l'utilisateur coopératif dans le choix d'un plan d'exécution pour une requête a également été le cas dans [TK15]. Dans ce travail, Trummer et al. [TK15] ont proposé une méthode d'optimisation de requêtes appelée « Probably Approximately Optimal Query Optimization » (PAO). L'objectif dans cette méthode est de trouver -pour une requête- un plan dont le coût est proche de l'optimal avec une certaine probabilité. Le principe de cette méthode peut être formulé comme suit : pour une requête q, trouver un plan d'exécution p tel que son coût puisse excéder le coût optimal d'au plus $\alpha \leq 1$ avec une probabilité minimale $\delta \in [0, 1]$. Cette condition est dite *condition d'optimalité approximative*. Le seuil α (exprimé en décimal) et la probabilité δ sont spécifiés par l'utilisateur. PAO opère d'une manière itérative. A la première itération, un ensemble initial d'échantillons de données est obtenu via un module d'échantillonnage. Puis, un optimiseur traditionnel est invoqué pour trouver le plan optimal pour une requête, pour les valeurs des sélectivités disponibles. Ces valeurs forment l'espace de sélectivités. PAO détermine ensuite la région dans l'espace de sélectivités où le plan généré par l'optimiseur satisfait la condition d'optimalité approximative. PAO calcule la probabilité que les sélectivités susceptibles d'être observées à l'exécution sont à l'intérieur de la région d'optimalité approximative déterminée. Si cette probabilité dépasse le seuil δ, l'exécution du plan commence. Sinon, le module d'échantillonnage est rappelé pour fournir un nouvel échantillon de données à considérer dans la prochaine itération. Les échantillons sont choisis en fonction des caractéristiques de l'itération actuelle. PAO peut itérer plusieurs fois, jusqu'à ce que la condition d'optimalité approximative soit respectée.

Une solution pour modéliser l'incertitude des estimations utilisées par l'optimiseur -autre que les distributions de probabilités- consiste à utiliser des intervalles d'estimations des valeurs des paramètres jugés incertains. L'utilisation d'intervalles d'estimations permet de considérer plus de valeurs susceptibles d'être observées à l'exécution, comparé aux distributions de probabilités. Utiliser des intervalles d'estimations serait plus judicieux lorsque l'incertitude est importante.

La littérature de recherche propose de nombreuses méthodes d'optimisation qui s'appuient sur des intervalles d'estimations pour générer des plans d'exécution adéquats. Un exemple de ces méthodes est le travail de Babu et al. [BBD05, BB05]. Babu et al. ont présenté dans [BBD05, BB05] une méthode d'optimisation de requêtes appelée « Proactive Reoptimization »(Rio). L'idée clé est de réagir d'une manière proactive aux sous-optimalités de plans dues à des erreurs d'estimations. Dans la phase d'optimisation initiale dans Rio, l'incertitude d'un paramètre est modélisée par un intervalle d'estimations pour ce paramètre. Puis, si le plan choisi par l'optimiseur aux bornes inférieure et supérieure de l'intervalle est le même que celui choisi au point correspondant à la valeur singulière estimée, alors ce plan est supposé robuste sur l'intervalle. L'exécution de ce plan est ensuite déclenchée. Pendant l'exécution, des statistiques sont collectées. Si la valeur observée du paramètre en question n'appartient pas à l'intervalle d'estimations pour ce paramètre, alors une ré-optimisation est déclenchée pour redéfinir l'intervalle d'estimations. Rio choisit, si nécessaire, un nouveau plan robuste. Lorsqu'il n'existe pas un plan robuste sur l'intégralité de l'intervalle d'estimations considéré, Rio cherche à produire un ensemble S de plans dits *interchangeables*. Ces plans vérifient les conditions suivantes :

- chaque plan de S possède un opérateur racine différent des opérateurs racines des autres plans dans S, i.e., l'algorithme physique de l'opérateur et l'ordre des relations opérandes sont différents,
- les opérateurs racines de tous les plans de S possèdent la même relation de base comme une des relations opérandes, et

- les opérateurs racines de tous les plans de S possèdent le même sous-plan comme l'autre relation opérande

L'ensemble S de plans interchangeables assure qu'à chaque point dans l'intervalle d'estimations utilisé, il existe au moins un plan robuste dans S. Notons qu'un plan robuste sur un intervalle en entier est un singleton de plans interchangeables. Dans le cas où Rio ne réussit pas à produire un ensemble de plans interchangeables, il se comporte comme la méthode proposée par Markl et al. [MRS+04] et décrite précédemment. Rio produit alors un plan estimé optimal pour les estimations en valeurs singulières des paramètres coûts. Alyoubi et al. [AHW15a, AHW15b] ont également abordé le problème de manque de fiabilité dans les estimations utilisées pour engendrer un plan d'exécution. Ils ont étudié -particulièrement- le problème d'ordonnancement des opérateurs de sélection et de jointure quand les sélectivités sont modélisées par des intervalles d'estimations plutôt que des valeurs spécifiques. Dans la méthode d'optimisation qu'ils proposent, l'optimiseur a pour rôle de choisir un plan d'exécution qui minimise le regret maximal. Le *regret* d'un plan d'exécution avec un ordre spécifique d'opérateurs est défini comme suit : *Soit un plan p, avec un ordonnancement o, le regret absolu γ(p, o) de p pour o est :*

$$\gamma(p, o) = Cost(p, o) - Cost(p_{opt(o)}, o) \tag{2.3}$$

Le *regret absolu* est défini par l'écart entre le coût d'un plan p avec un ordonnancement particulier o et le coût du plan optimal avec le même ordonnancement. Le regret d'un plan dans le pire des cas est nommé *regret maximal*. Ce dernier est défini comme $\max_{o \in O}(\gamma(\text{p, o}))$. Le plan ayant le regret optimal est celui qui a le plus petit regret maximal. Le regret optimal est défini comme suit : *Soient P un ensemble de plans d'exécution possibles et O l'ensemble des ordonnancements possibles, la minimisation du regret maximal est définie par :*

$$R(P, O) = min_{p \in P}(max_{o \in O}(\gamma(p, o))) \tag{2.4}$$

L'objectif est de trouver le plan qui minimise l'écart maximale entre son coût et le coût

du plan optimal pour les ordonnancements possibles.

Dans les méthodes décrites dans cette sous-section, un optimiseur génère un seul plan d'exécution pour une requête en entrée. Si la méthode considère la ré-optimisation, ce plan peut être modifié ou remplacé, mais à chaque invocation de l'optimiseur, un seul plan est produit. Lorsque l'incertitude de l'optimiseur concernant le choix d'un plan d'exécution est importante, une solution pour éviter de faire un mauvais choix est de produire plusieurs plans d'exécution pour une même requête. La littérature fournit de nombreuses méthodes d'optimisation dans ce contexte. Nous présentons ces méthodes dans la section suivante.

2.3.2 Méthodes basées sur plusieurs plans d'exécution

Les premières méthodes à engendrer un ensemble de plans d'exécution pour une requête, sont celles proposées dans le cadre d'une optimisation paramétrique [INSS92, HS02]. Dans une optimisation paramétrique, des plans d'exécution pour une requête sont identifiés lors de l'optimisation, tel que chacun est proche de l'optimal sur une partie/région de l'intervalle/espace d'estimations des sélectivités. Le plan adéquat est choisi au commencement de l'exécution, quand des informations manquantes deviennent disponibles. Dans ce contexte, Graefe et al. [CG94] ont proposé une méthode appelée «Dynamic Query Evaluation Plans». Dans cette méthode, des opérateurs dits *opérateurs de choix de plans*, sont utilisés pour choisir un plan d'exécution adéquat en se basant sur les informations collectées à l'exécution. La limite principale de ces méthodes réside dans le nombre important de plans à générer, stocker, charger, et comparer à l'exécution.
Dans l'objectif de réduire cette complexité, Haritsa et al. [DDH08, ABD+10] ont proposé une méthode d'optimisation appelée « Selectivity Estimate Error Resistance » (SEER). SEER se base sur la *réduction anorexique de diagrammes de plans*. Un diagramme de plans est une énumération codée-couleur de plans générés par un optimiseur pour une requête paramétrée, dans un espace de sélectivités S. Un diagramme de plans peut être

complexe et dense, avec un grand nombre de plans couvrant l'espace de sélectivités S. Ce diagramme est représenté sous forme d'une grille de points. Chaque point, noté q(x,y), correspond à une requête spécifique q avec les sélectivités x, y dans l'espace S. A chaque requête est associé un plan optimal p avec un coût *coût(p)* représentant le coût estimé de l'exécution de q par p. Certains plans sont plus sensibles aux erreurs d'estimations que d'autres. Un plan couvrant une petite région dans l'espace S est considéré plus sensible qu'un plan couvrant une plus grande région. SEER tente de remplacer les plans susceptibles de devenir sous-optimaux à l'exécution par des plans peu sensibles aux éventuelles erreurs d'estimations. Si l'utilisateur tolère une augmentation mineure des coûts de plans par rapport à leurs coûts originaux, le nombre de plans à générer et à comparer peut être réduit, sans pour autant dégrader les performances. Les diagrammes de plans complexes et denses peuvent alors être réduits à des diagrammes beaucoup plus simples, avec seulement quelques plans sans affecter la qualité de l'évaluation de requêtes. Ce principe est appelé réduction anorexique de diagrammes de plans.

Comme Harista et al. [DDH08, ABD$^+$10], Dutt et al. [DNH14, DH14, DH16] se sont intéressés aux erreurs d'estimations des sélectivités. Ils ont proposé une méthode d'optimisation qui aborde la difficulté d'obtenir des estimations précises des sélectivités lors de l'optimisation. Dans un premier temps, des intervalles/surfaces d'estimations sont calculés autour des prédicats d'une requête, jugés sujets à des erreurs d'estimations. Un optimiseur de requêtes énumère ensuite des plans d'exécution pour une requête en entrée tel que chaque plan est optimal sur une partie/région d'un intervalle/une surface d'estimations. Ces plans forment un ensemble paramétrique de plans optimaux (POSP). POSP couvre les intervalles/surfaces de sélectivités des prédicats. L'optimiseur compare ensuite les coûts de ces plans sur ces intervalles/surfaces d'estimations. A partir de la variation des coûts des plans, l'optimiseur détermine ce que les auteurs ont appelé *trajectoire du coût minimal*. Cette trajectoire est représentée sous forme d'une courbe (PIC) qui comporte les coûts minimals des plans POSP. PIC représente les performances idéales. PIC est ensuite discrétisée. En mathématique, discrétiser une courbe c'est regrouper les points

de cette courbe afin de la représenter par un ensemble fini de points plutôt que par une ligne. Dans [DNH14, DH14, DH16], la discrétisation du PIC est effectuée en projetant des étapes d'*IsoCoût* (IC) prédéfinies, IC1, IC2, ..., sur la courbe PIC. Une étape d'*IsoCoût* est une droite représentant un coût spécifique (comme une contrainte budgétaire). Les coûts de ces étapes sont progressifs. Par exemple, chaque valeur de IC double la IC précédente. L'intersection de chaque IC avec le PIC correspond à une valeur de sélectivité et permet d'identifier le meilleur plan dans POSP, pour cette sélectivité. L'ensemble des plans associés à ces IC est appelé un *bouquet de plans*. Un *bouquet de plans* est un sous-ensemble limité de plans optimaux de la requête tel qu'à chaque point/région de l'intervalle/surface d'estimations utilisé, il existe au moins un plan optimal dans ce bouquet. A l'exécution, le plan associé à l'étape IC la moins coûteuse est exécuté en premier. Si les coûts d'exécutions partielles dépassent la valeur IC, cela signifie que la sélectivité est au-delà de la plage pour laquelle le plan en cours est optimal. Dans ce cas, l'exécution du plan associé à la valeur IC suivante est déclenchée. Dans le cas où le plan en cours termine l'exécution avant d'atteindre la valeur IC, cela signifie que la sélectivité est dans la plage de sélectivités couverte par ce plan. Le fait que les coûts d'exécution de plans soient délimités par les valeurs IC, le principe de *bouquet de plans* permet de réduire les sous-optimalités dans le cas d'erreurs importantes dans les estimations.

Après avoir présenté les méthodes d'optimisation, nous proposons dans la section suivante une comparaison des méthodes dans chaque approche. Nous terminons la section suivante par une comparaison des deux approches.

2.4 Comparaison des méthodes d'optimisation

Dans cette section nous comparons les méthodes d'optimisation de requêtes décrites dans les sections précédentes. Pour cela, nous nous basons sur des critères que nous présentons ci-après. Nous terminons par une discussion des deux approches d'optimisation

identifiées en mettant en évidences les points forts et les limites de chacune.

2.4.1 Critères de comparaison

Les méthodes d'optimisation décrites dans ce chapitre divergent sur de nombreux aspects : elles n'ont pas toutes le même impact sur un plan d'exécution, elles ne réagissent pas toujours aux mêmes événements. Elles peuvent recueillir des informations à l'exécution à des fréquences différentes et s'adapter à des niveaux différents. En outre, elles sont applicables à des environnements différents. Ces aspects sont détaillés ci-après :

- *Modification de plan d'exécution* : des méthodes d'optimisation proposent de surveiller l'exécution d'un plan associé à une requête. Les statistiques recueillies sont utilisées pour modifier -si nécessaire- le plan en cours afin de l'adapter aux conditions d'exécution observées. Les principales modifications (champ MODIF dans le tableau 2.1) qui peuvent être apportées à un plan d'exécution sont :
  - **ré-ordonnancement d'opérateurs :** ceci s'applique quand la méthode adapte l'ordre d'exécution d'opérateurs aux conditions observées à l'exécution.
  - **remplacement d'opérateurs :** s'applique dans le cas où un opérateur physique peut être remplacé par un opérateur équivalent lors de l'exécution afin de produire un plan d'exécution plus efficace. Par exemple, une jointure par hachage peut être remplacée par une jointure par index, si un index sur l'attribut de jointure devient disponible à l'exécution.
  - **ré-optimisation :** quand la méthode est capable de produire un plan totalement différent pour le reste de la requête, en ré-ordonnançant/remplaçant des opérateurs.
  - **ré-allocation de ressources :** s'applique quand une méthode est capable d'adapter l'allocation de ressources (e.g., mémoire) aux exigences constatées à l'exécution.
  - **modification du degré de parallélisme :** dans ce cas, la modification d'un plan s'effectue par une adaptation du degré de parallélisme en fonction de la

disponibilité de ressources observée à l'exécution.

- *Événement* : les méthodes d'optimisation existantes diffèrent sur les événements auxquels elles réagissent (champ EVEN dans le tableau 2.1). Il existe principalement trois domaines d'intérêt :
  - **indisponibilité de statistiques :** des statistiques décrivant des données peuvent être indisponibles lors du choix d'un plan d'exécution en raison -par exemple- de la complexité de prédicats ou la provenance de données à partir de sources distantes.
  - **imprécision des estimations :** des estimations utilisées par l'optimiseur peuvent être erronées suite à l'utilisation d'hypothèses invalides, la variation de données à l'exécution par rapport à l'optimsiation, ou l'évolution de caractéristiques d'exécution (e.g., charge CPU, quantité de mémoire disponible) lors d'une exécution.
  - **retards d'arrivée de données :** une augmentation du trafic réseau dans un environnement distribué risque de provoquer du retard dans les délais prévus d'arrivée de données.
- *Moyen de collecte de statistiques* : pendant l'exécution d'un plan, des statistiques sont collectées pour vérifier la précision des estimations utilisées par l'optimiseur. La collecte de statistiques (champ MCC dans tableau 2.1) est assurée soit par :
  - **observation :** des statistiques (e.g., tailles des relations temporaires) sont collectées en contrôlant les tuples qui traversent des opérateurs spécifiques (e.g., *opérateurs de contrôle* [KD98]) insérés dans un plan d'exécution.
  - **exploration :** s'applique principalement dans les méthodes basées sur le routage de tuples. L'acheminement de tuples en entrée à travers différents opérateurs d'un plan dans des ordres différents, permet d'obtenir des statistiques sur les sélectivités des opérateurs.
- *Fréquence de collecte de statistiques* : Un critère permettant de distinguer les méthodes d'optimisation est la fréquence de collecte des statistiques (champ FC dans

le tableau 2.1). Celle-ci peut se produire au niveau :

— **inter-opérateur** (inter dans le tableau 2.1), ou

— **intra-opérateur** (intra dans le tableau 2.1)

Pour les techniques de la première catégorie, les statistiques sont collectées après une matérialisation de résultats intermédiaires entre différents opérateurs d'un plan. Pour celles de la deuxième catégorie, les statistiques sont collectées pendant l'exécution d'un opérateur.

- *Nature de la décision de modification* : les décisions de modification (champ DM dans le tableau 2.1) d'un plan d'exécution peut être affectée différemment. Elles peuvent être :
  — **centralisée** (C) : si l'optimiseur de requêtes, ou tout autre élément d'un SGBD, prend des mesures pendant l'exécution pour veiller à ce que le reste d'un plan soit évalué de la manière attendue.
  — **décentralisée** (DC) : s'applique principalement aux systèmes distribués où une vision globale de l'état des sites participants à l'évaluation d'une requête n'est pas facile à obtenir.
- *Environnement d'application* : les méthodes proposées se différencient sur l'environnement (champ ENV dans le tableau 2.1) pour lequel elles ont été conçues :
  — **uni-processeur** (U),
  — **parallèle** (P), ou
  — **distribué** (D)

  Une méthode conçue pour un environnement spécifique est capable de réagir, seulement, aux événements qui se produisent dans cet environnement. Une méthode conçue pour un environnement uni-processeur ne prend pas compte des problèmes spécifiques aux environnements distribué et parallèle. Par exemple, pour qu'un plan soit efficace dans un environnement distribué, des statistiques sur le paramètre réseau et les nœuds disponibles sont requises. Pour les environnements parallèles, le degré du parallélisme constitue un paramètre clé dans la qualité d'un plan.

2.4.2 Comparaison des méthodes d'optimisation basée sur des estimations en points singuliers

Nous présentons dans le tableau 2.1 dans la page suivante, une comparaison des méthodes d'optimisation basée sur des estimations en points singuliers. Nous nous appuyons sur les critères de comparaison décrits ci-dessus. Comme l'indique ce tableau, chaque méthode adresse le problème d'erreurs d'estimations causées par des événements spécifiques à un environnement particulier. Toutes les méthodes visent un même objectif : assurer des performances optimales d'évaluation de requêtes. Pour cela, chaque méthode propose une ou des modifications possibles d'un plan d'exécution pour corriger les sous-optimalités constatées. La majorité des méthodes contrôlent l'optimalité d'un plan par une observation des statistiques lors de l'exécution. Ces statistiques sont généralement recueillies avec une matérialisation de résultats intermédiaires. Nous constatons aussi dans ce tableau que le critère centralisé des décisions de modifications est dominant. Ceci peut handicaper la capacité des méthodes d'assurer des performances optimales. En effet, rappeler l'optimiseur plusieurs fois pendant l'exécution d'une requête peut engendrer un sur-coût important, d'autant plus qu'une décision de modification peut ne pas être bénéfique, i.e. modification basée sur des estimations à nouveau erronées.

La principale conclusion que nous pouvons tirer à partir de ce tableau, c'est que parmi les méthodes analysées, aucune n'est pleinement idéale pour l'ensemble des critères de comparaison. Nous trouvons que la méthode qui répond au mieux à nos critères est la méthode proposée par Kabra et DeWitt [KD98]. Cette méthode adresse plus d'événements comparé aux autres méthodes. Elle accorde une attention particulière à la nature des statistiques à collecter, aux points de collecte et à la condition de ré-optimisation. Cependant, sa limite principale est qu'elle est totalement réactive. Elle utilise la ré-optimisation pour corriger des sous-optimalités observées. A la phase d'optimisation ou lors d'une ré-optimisation, la possibilité d'erreurs dans les estimations utilisées est ignorée. Un plan généré est estimé optimal pour des valeurs particulières des paramètres nécessaires aux calculs des coûts. Ce principe de fonctionnement augmente le risque de sous-optimalités de plans et

de corrections requises. Plusieurs ré-optimisations d'un plan peuvent ralentir le temps de réponse, induisant ainsi des performances sous-optimales.

Critères / *Méthodes*	*MODIF*	*EVEN*	*MCC*	*FC*	*DM*	*ENV*
MIND [EDNO97]	ré-ordonnancement d'opérateurs	indisponibilité de statistiques	observation	inter	C	D
CIE [NGl13]	ré-optimisation	imprécision des estimations	observation	inter	C	U
Mid-Query Reoptimization [KD98]	ré-optimisation / ré-allocation de ressources (mémoire)	imprécision des estimations / indisponibilité de statistiques	observation	inter	C	P
POP [MRS+04]	ré-optimisation	imprécision des estimations / indisponibilité de statistiques	observation	inter	C	P
Tukwila [IFF+99]	ré-optimisation / ré-allocation de ressources (mémoire)	imprécision des estimations / indisponibilité de statistiques / retards d'arrivée de données	observation	inter/ intra	C/ DC	D
Query Scrambling [MFTU96]	ré-optimisation	retards d'arrivée de données	observation	inter	C	D
Continuous Query Optimization [BJZ13]	modification du degré de parallélisme	imprécision des estimations / indisponibilité de statistiques	observation	inter	C	D/P
DyOpt [KBK+14]	ré-optimisation	imprécision des estimations	observation	inter	C	P
Eddy [AH00]	ré-ordonnancement d'opérateurs	imprécision des estimations / indisponibilité de statistiques	exploration	intra	C	U
Data Partionning [IHW04]	ré-optimisation	imprécision des estimations / indisponibilité de statistiques	observation	intra	C	D

TABLE 2.1 – Comparaison des méthodes d'optimisation basée sur des estimations en points singuliers

2.4.3 Comparaison des méthodes d'optimisation basée sur des estimations en un ensemble de points

Dans le tableau 2.2 dans la page suivante, nous présentons une comparaison des méthodes d'optimisation basée sur des estimations en un ensemble de points, toujours suivant les critères décrits précédemment. Comme nous pouvons le constater à partir de ce tableau, toutes les méthodes tiennent compte de la possibilité d'erreurs d'estimations au moment de l'optimisation. Toutefois, elles ne considèrent pas toutes le contrôle d'exécutions ni la correction -si nécessaire- de plans d'exécution. Dans les méthodes qui considèrent la ré-optimisation, comme pour les méthodes de la première approche, nous constatons la dominance du critère centralisé des décisions de modifications de plans d'exécution.
Nous pouvons conclure à partir de ce tableau que la méthode qui répond au mieux à nos critères est celle proposée par Babu et al. [BBD05]. Elle est dotée d'un comportement à la fois préventif et correctif. Le choix d'un plan d'exécution s'effectue avec une prise en compte d'erreurs d'estimations potentielles. Le plan généré est susceptible de fournir des performances robustes sur un intervalle d'estimations. A l'exécution, si la valeur observée d'un paramètre jugé incertain n'est pas dans l'intervalle d'estimations utilisée, une ré-optimisation est déclenchée pour recalculer l'intervalle d'estimations pour ce paramètre et regénérer un plan adéquat. La limite principale que présente cette méthode [BBD05] c'est que le calcul d'un intervalle d'estimations est basé sur des statistiques dans la méta-base. Si les statistiques requises sont indisponibles ou encore obsolètes, un intervalle calculé risque d'être erroné. La valeur à l'exécution du paramètre en question risque de ne pas appartenir à cet intervalle. Cela peut engendrer plusieurs ré-optimisations pour recalculer l'intervalle et par conséquent, un sur-coût qui peut violer la robustesse visée. En outre, quand Rio [BBD05] échoue à trouver un plan robuste ou un ensemble de plans interchangeables sur l'intervalle d'estimations utilisé, Babu et al. [BBD05] proposent de sélectionner le plan estimé optimal en se basant sur des estimations singulières des paramètres coûts, comme dans [MRS+04]. L'incertitude des estimations n'est plus considérée. Le plan généré est estimé optimal pour des conditions d'exécution spécifiques. La robus-

tesse n'est plus en vigueur. Le risque de sous-optimalités de plans générés et de sur-coûts dus à des ré-optimisations éventuelles de plans augmente.

Critères / *Méthodes*	*MODIF*	*EVEN*	*MCC*	*FC*	*DM*	*ENV*
LEC [CHG02]	aucun effet	imprécision des estimations	aucun effet	aucun effet	C	U
Robust Query Optimizer [BC05]	aucun effet	imprécision des estimations	aucun effet	aucun effet	C	U
PAO [TK15]	ré-optimisation	imprécision des estimations	observation	inter/ intra	C	U
RIO [BBD05]	ré-optimisation	imprécision des estimations / indisponibilité de statistiques	observation	inter/ intra	C	U
Minmax Regret [AHW15a, AHW15b]	aucun effet	imprécision des estimations	aucun effet	aucun effet	C	U
Dynamic Query Evaluation plans [CG94]	ré-optimisation	indisponibilité de statistiques	observation	inter	C	U
SEER [ABD+10, DDH08]	aucun effet	indisponibilité de statistiques	aucun effet	aucun effet	C	U
Plan bouquet [DNH14, DH14, DH16]	ré-optimisation	indisponibilité de statistiques	observation	inter	C	U

Table 2.2 – Comparaison des méthodes d'optimisation basée sur des estimations en un ensemble de points

2.4.4 Comparaison des deux approches

Les méthodes proposées dans le cadre de la première approche ont la capacité de détecter et corriger les erreurs d'estimations et les sous-optimalités qui en résultent pendant l'exécution d'une requête. Avant d'appliquer une modification d'un plan d'exécution, cer-

taines étapes doivent être validées (e.g., collecte de statistiques, validation par rapport à un modèle de coûts) pour assurer que la modification est susceptible d'être bénéfique. Ceci entraine un sur-coût supplémentaire. En outre, l'optimiseur peut choisir un nouveau plan dont les performances dépendent de statistiques qui sont imprécises. En effet, la capacité de ces méthodes de collecter des statistiques rapidement et avec précision lors de l'exécution est limitée [BBD05]. La correction d'une sous-optimalité peut être déclenchée avec des estimations à nouveau erronées. Ceci peut entrainer d'autres ré-optimisations d'un plan et ainsi un sur-coût important.

La deuxième approche est basée sur la prévention plutôt que la correction. Cette approche inclut des méthodes qui considèrent la ré-optimisation pendant l'exécution et d'autres qui possèdent un comportement totalement proactif. Comme détaillé dans ce chapitre, la première approche se base sur une estimation spécifique pour chaque paramètre nécessaire aux calculs des coûts. Contrairement à ceci, la deuxième approche fait usage d'intervalles d'estimations ou de distributions de probabilités pour modéliser l'incertitude concernant les valeurs estimées des paramètres. Ces intervalles/distributions de probabilités sont ensuite utilisés pour faire un choix prudent de plans d'exécution.

Pour illustrer les conditions de pertinence de chaque approche par rapport à l'autre, nous présentons l'exemple suivant : considérons une requête Q' de jointure entre deux relations R et S, avec une clause sélective sur R :

$$\text{Q'} = \textit{select * from R, S where R.a = S.b and R.c} < \textit{v1 and R.d} > \textit{v2}$$

Cette requête nécessite une jointure entre σ(R) et S, où σ(R) est le résultat de "R.c < v1 and R.d > v2". Supposons que l'optimiseur estime -en utilisant un modèle de coûts- que le facteur de sélectivité de σ(R) est de 40% et qu'il existe deux plans d'exécution possibles pour Q', notés P1 et P2 (Cf. Figure 2.5, page 61). Supposons que la quantité de mémoire allouée pour exécuter la jointure entre S et σ(R) est supérieure à |S|. Si l'optimiseur estime que $|\sigma(R)| < |S|$, le plan P1 serait préférable et une table de hachage serait construite avec le résultat de σ(R). Cependant, si $|\sigma(R)|$ est sous-estimée et qu'elle est supérieure à la taille mémoire allouée, utiliser P1 conduit à exécuter une jointure par hachage par morceaux.

Ceci risque d'induire un sur-coût important et ainsi une dégradation des performances. Contrairement à cela, utiliser P2 permet de construire une table de hachage avec la relation de base S et garantir une exécution en un seul morceau. En d'autres termes, utiliser P1 serait un choix risqué alors qu'utiliser P2 serait un choix prudent. La variation des coûts des plans P1 et P2, illustrée dans la figure 2.5 permet de visualiser ceci.

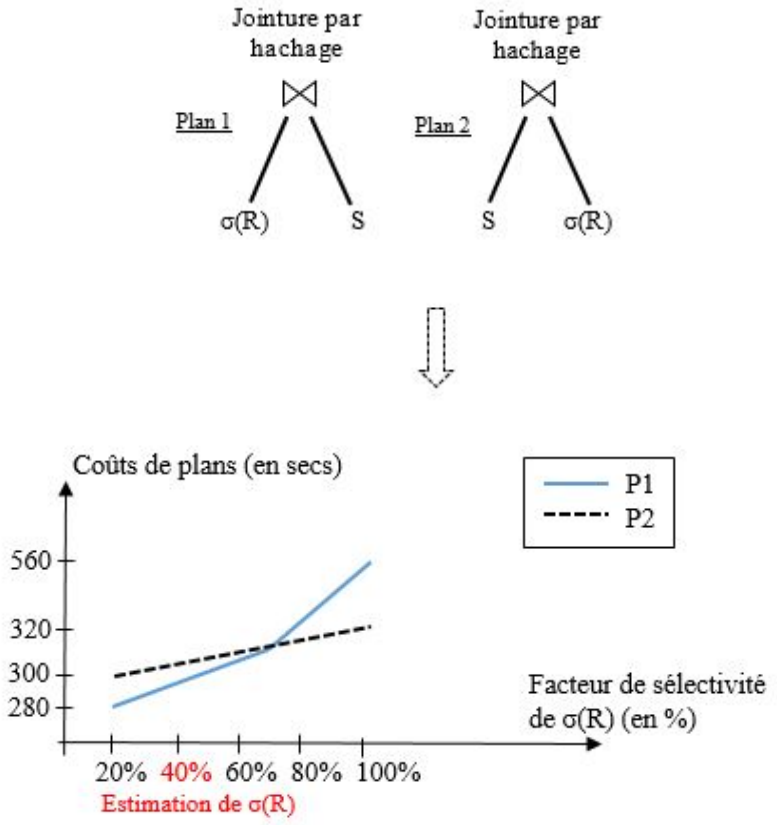


FIGURE 2.5 – Coûts des plans P1 et P2 en fonction de σ(R)

Nous pouvons constater à partir de cette figure que l'incertitude concernant le facteur de sélectivité de σ(R) possède un impact plus prononcé sur le coût de P1 que sur le coût de P2. Le coût de P2 est presque certainement compris entre 300 secondes et 320 secondes, alors que le coût de P1 pourrait être le plus faible (280 secondes) ou bien le plus élevé (560 secondes). Les méthodes qui utilisent l'approche d'optimisation basée sur des estimations en points singuliers choisiront le plan P1 car il génère des performances optimales pour

la valeur estimée de σ(R). Les méthodes utilisant l'approche d'optimisation basée sur des estimations en un ensemble de points choisiront P2, car il offre des performances robustes sur l'intervalle d'estimations du facteur de sélectivité de σ(R), i.e., [20% , 95%].
P1 est considéré comme un plan risqué. Il est optimale seulement si la valeur estimée est exacte. Il peut cependant engendrer des performances sous-optimales dans le cas contraire. Le plan P2 est considéré comme une alternative stable. Il offre des performances plus stables dans l'intervalle d'estimations de σ(R). Le choix entre P1 et P2 devrait dépendre des attentes de l'utilisateur. Si la minimisation des coûts d'exécution est la principale préoccupation, alors P1 pourrait être sélectionné. Pour les utilisateurs qui cherchent la stabilité, P2 serait préférable puisque son coût est moins sensible aux erreurs d'estimations. En règle générale, que P1 ou P2 soit le meilleur choix dépendrait de trois facteurs : (1) le type de requêtes, (2) les attentes des utilisateurs, et (3) l'environnement d'exécution. L'approche utilisant plusieurs points d'estimations est adapté à une utilisation dans un environnement où obtenir des estimations précises est très complexe. Lorsque les erreurs d'estimations sont peu probables, l'optimisation basée sur des estimations en points singuliers est susceptible de produire de meilleures performances. Le tableau 2.3 ci-dessous résume les avantages et les limites des deux approches.

	Optimisation basée sur des estimations en points singuliers	Optimisation basée sur des estimations en un ensemble de points
Avantages	• capacité de détecter et de corriger des sous-optimalités de plans pendant l'exécution	• stabilité de l'exécution
Limites	• choix d'un plan d'exécution d'une manière aveugle, l'incertitude des estimations est ignorée • ajout d'un sur-coût éventuellement important à l'exécution (e.g., collecte de statistiques, matérialisation de résultats intermédiaire, modification d'un plan)	• rare considération de la ré-optimisation • ajout d'un sur-coût à la phase d'optimisation (e.g., calcul d'intervalles d'estimations) • nécessité, à l'optimisation, d'informations supplémentaires (e.g., distribution de probabilités)

Table 2.3 – Avantages et limites des deux approches

2.5 Conclusion

Dans ce chapitre, nous avons présenté les principales méthodes d'optimisation proposées dans la littérature, dont l'objectif est de gérer les erreurs d'estimations et les sous-optimalités conséquentes. Nous avons distingué deux approches principales : (1) optimisation basée sur des estimations en points singuliers, et (2) optimisation basée sur des estimations en un ensemble de points, que nous avons étudiées et comparées. Les possibilités offertes par ces approches pour faire face à l'inexactitude et/ou l'indisponibilité des statistiques nécessaires pour engendrer un plan efficace ont été identifiées. En outre, les méthodes ont été comparées par rapport aux événements auxquels elles réagissent, la nature de modifications de plans de requêtes, la nature d'informations qu'elles collectent

à l'exécution ainsi que la fréquence à laquelle elles peuvent modifier un plan d'exécution. D'autres critères de comparaison constituent l'environnement d'exécution et la nature du composant responsable des modifications d'un plan.

L'étude de ces méthodes nous a permis de constater que chacune des deux approches identifiées répond partiellement au problème d'optimisation du aux erreurs d'estimations. Les méthodes dans le cadre de la première approche cherchent à réaliser au mieux un choix optimal d'un plan d'exécution. Elles se concentrent sur le contrôle de l'exécution d'un plan et sur la réaction à prendre pour corriger une sous-optimalité observée. Les estimations utilisées lors de l'étape initiale d'optimisation ou lors d'une ré-optimisation sont considérées précises. Ceci risque de handicaper la capacité de ces méthodes d'atteindre des performances optimales d'évaluation de requêtes.

La deuxième approche d'optimisation s'appuie sur un raisonnement préventif. Les méthodes dans le cadre de cette approche tiennent compte de la possibilité d'erreurs dans les estimations utilisées par l'optimiseur pour générer un plan d'exécution. Ces méthodes ont accepté le fait qu'il est très difficile de pouvoir produire un plan optimal puisqu'il est très difficile d'obtenir des estimations précises. Leur objectif est désormais d'engendrer des plans d'exécution robustes. Toutefois, engendrer un plan robuste sur un grand intervalle d'estimations reste complexe. De plus, la majorité de ces méthodes ignorent la variation des conditions d'exécution observées par rapport aux conditions d'exécution prévues à l'optimisation. Ainsi, un plan présumé robuste au moment de l'optimisation est supposé robuste jusqu'à la fin de l'exécution, ce qui n'est pas toujours correcte.

Compte tenu de ces constatations, nous proposons dans le cadre de cette thèse une méthode d'optimisation hybride qui étend les méthodes d'optimisation basée sur des estimations en un ensemble de points. Nous cherchons à augmenter la capacité d'un optimiseur d'engendrer des plans d'exécution robustes et essayons d'assurer cette robustesse jusqu'à la terminaison d'une exécution. Nous détaillons notre méthode dans le chapitre suivant.

Chapitre 3

HyOpt : Méthode d'optimisation hybride

Sommaire

3.1 Introduction

Dans le chapitre précédent, nous avons analysé les principales méthodes d'optimisation de requêtes existantes, en mettant en évidence les points forts et les limites des deux approches identifiées. La limite majeure de l'approche d'optimisation basée sur des estimations en points singuliers, est que la possibilité d'erreurs dans les estimations est ignorée lors du choix initial d'un plan d'exécution. Ceci rend les méthodes dans cette approche très opportunistes à des ré-optimisations d'un plan d'exécution. En outre, lors d'une ré-optimisation, l'optimiseur choisit un nouveau plan en supposant que les nouvelles estimations disponibles sont précises. Malheureusement, ce n'est pas toujours le cas. Des statistiques collectées pendant l'exécution d'un plan peuvent manquer de précision ou devenir obsolètes -notamment en cas de requêtes complexes- [BBD05, GSW12, LGM+15]. Les estimations produites à partir de ces statistiques peuvent être à nouveau erronées. Par conséquent, une nouvelle sous-optimalité peut être détectée et une autre ré-optimisation devient nécessaire, ce qui peut induire un sur-coût important.
En ce qui concerne l'approche d'optimisation basée sur des estimations en un ensemble de points, sa limite principale est due à une hypothèse souvent utilisée selon laquelle engendrer un plan robuste sur un ensemble de points est toujours possible. Cependant, plus l'incertitude concernant la précision des estimations utilisées augmente, plus l'ensemble de points modélisant cette incertitude est important. Dans ce cas, il devient complexe de trouver un plan qui est robuste pour tous ces points. En outre, le plan sélectionné à la phase d'optimisation comme celui offrant des performances robustes pour différentes conditions d'exécution, est généralement maintenu sans modification jusqu'à la fin de l'exécution. En effet, la majorité des méthodes dans cette approche supposent que les conditions d'exécution prévues au moment de l'optimisation correspondront aux conditions observées. Cette hypothèse reste souvent injustifiée. Des paramètres comme la quantité de mémoire disponible, les tailles de relations temporaires produites à des étapes intermédiaires lors de l'exécution d'un plan ne peuvent être connus avec précision au moment de l'optimisation. Ainsi, cette hypothèse peut entrainer de mauvaises performances [LGM+15].

Compte tenu des limites des approches d'optimisation étudiées, nous proposons dans ce chapitre une méthode d'optimisation hybride qui vise deux objectifs : la production de plans d'exécution robustes, notamment lorsque l'incertitude des estimations utilisées par l'optimiseur est importante, et la détection et la correction d'une violation de la robustesse pendant l'exécution d'un plan.

Notre méthode d'optimisation est composée de deux modules, appelés (1) *Génération de plans d'exécution robustes*, et (2) *Correction de plans d'exécution*. Le premier module a pour objectif de produire des plans d'exécution robustes qui gèrent l'incertitude des estimations utilisées à la phase d'optimisation. Il consiste en une stratégie d'optimisation [MMH16] qui s'appuie sur des intervalles d'estimations pour identifier des plans d'exécution robustes pour une requête en entrée. Lorsqu'ils existent plusieurs plans associés à cette requête, tel que chacun est robuste sur une partie d'un intervalle d'estimations, notre stratégie se base sur un raisonnement probabiliste pour sélectionner le plan qui minimise le risque d'une violation de la robustesse à l'exécution. Ce plan est utilisé pour commencer l'exécution. L'information concernant les autres plans et leurs sous-intervalles de robustesse associés est conservée pour être utilisée par le deuxième module.

Dans l'objectif d'essayer d'assurer la robustesse jusqu'à la terminaison de l'exécution, nous déterminons les points dans le plan utilisé pour commencer l'exécution, où l'incertitude est jugée importante et peut avoir un effet néfaste sur la robustesse. Des opérateurs dits de contrôle et de décision sont insérés dans le plan à ces points. Ces opérateurs représentent le pivot du module de correction de plans d'exécution. Ils ont pour rôle de surveiller l'exécution d'un plan et déclencher une correction du reste du plan si une violation de la robustesse est détectée. Les décisions de modifications sont gérées par ces opérateurs qui sont capables de réagir sans avoir besoin de ré-invoquer l'optimsieur. Ces opérateurs opèrent en se basant sur l'information déterminée au moment de l'optimisation concernant les autres plans et leurs sous-intervalles de robustesse associés, et sur les statistiques recueillies à l'exécution. L'idée clé de ce module est de préparer les décisions de corrections à la phase d'optimisation. Lorsqu'une correction est déclenchée à l'exécution,

la décision adéquate est sélectionnée sans avoir besoin de rappeler l'optimiseur de requêtes.

Le reste de ce chapitre est organisé comme suit : dans la section 3.2, nous décrivons notre stratégie d'optimisation pour la génération de plans d'exécution robustes, en détaillant les éléments qui la composent. Ensuite, nous présentons dans la section 3.3 notre méthode de correction de plans d'exécution sur la base d'opérateurs de contrôle et de décision. Enfin, nous concluons dans la section 3.4.

3.2 Génération de plans d'exécution robustes

La stratégie d'optimisation que nous proposons vise à produire des plans d'exécution robustes qui gèrent l'incertitude dans les estimations utilisées à la phase d'optimisation. Cette stratégie s'inscrit dans l'approche d'optimisation basée sur des estimations en un ensemble de points. Comme la majorité des méthodes dans cette approche, nous modélisons l'incertitude des estimations par des intervalles calculés autour de celles-ci. Ces intervalles permettent d'identifier le ou les plans robustes pour une requête. Après avoir étudié des méthodes proposées dans ce contexte, nous nous sommes aperçus que ces méthodes supposent qu'il est toujours possible de produire un plan robuste sur l'intégralité d'un intervalle d'estimations. Cependant, cette hypothèse n'est pas toujours valide. Pour illustrer ceci, nous présentons l'exemple 3.2 suivant :

Exemple 3.2 Considérons la requête Q_1 :

Select ***
from *client, commande*
where *client.ID = commande.clientID* ***and***
client.pays='France' ***and*** *client.ville='Paris';*

Supposons que la mémoire disponible pour l'exécution de cette requête est égale à 150 MB, que la taille de la relation *client*, notée |*client*| est égale à 300 MB et que |*commande*| = 100 MB.

En s'appuyant sur un modèle de coûts, un optimiseur de requêtes estime que $|\sigma(client)|$, qui représente le résultat de la sélection *client.pays='France' and client.ville='Paris'*, est sujette à une erreur d'estimation en raison d'une corrélation possible entre les attributs. Pour modéliser cela, un intervalle d'estimations, noté $I_{|\sigma(client)|}$, est calculé autour de $|\sigma(client)|$. Soit $I_{|\sigma(client)|}$ = [75, 250] MB. Cet intervalle est utilisé par la suite pour générer un plan d'exécution robuste. Supposons qu'il existe un index sur un attribut de la relation *commande*. En invoquant l'optimiseur, les plans Plan1, Plan2 et Plan3 (Cf. Figure 3.1) sont énumérés comme des plans d'exécution possibles pour la requête.

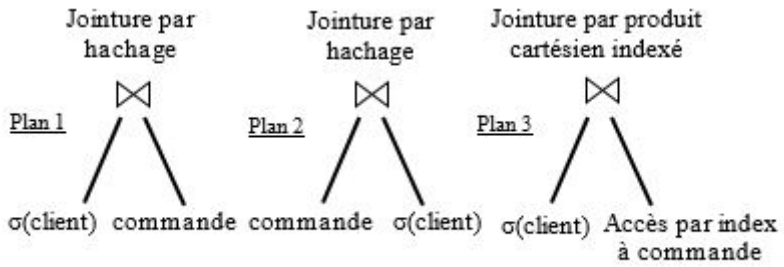


FIGURE 3.1 – Plans d'exécution possibles pour la requête Q_1

Dans les plans Plan1 et Plan2, un algorithme physique de jointure par hachage est utilisé. Le principe d'une jointure par hachage est de construire dans un premier temps une table de hachage avec la plus petite (en terme de taille) des relations opérandes. Puis, pour chaque tuple de l'autre relation, on détermine l'entrée dans la table de hachage et on effectue une jointure entre ce tuple et tous les tuples appartenant à cette entrée.
Dans le plan Plan3, un algorithme physique de jointure par produit cartésien indexé est utilisé. Le principe d'une jointure par produit cartésien indexé est de comparer chaque tuple d'une première relation dite relation "externe" avec les tuples d'une deuxième relation dite relation "interne", en utilisant un index sur l'attribut de jointure de la relation interne. La relation externe correspond à la relation opérande à partir de laquelle les tuples sont lus dans un premier temps par l'algorithme de jointure. La relation interne correspond à la relation opérande à partir de laquelle les tuples sont lus en utilisant les valeurs de l'attribut de jointure de la relation externe. Les tuples vérifiant la condition

de jointure forment le résultat. Plus de détails sur les algorithmes physiques de jointure peuvent être trouvés dans [GMWU99].

Revenons à notre exemple 3.2, supposons qu'un optimiseur de requêtes estime -en s'appuyant sur un modèle de coûts- les coûts des plans Plan1, Plan2 et Plan3 en fonction de la variation de $|\sigma(client)|$. L'objectif est de déterminer le plan qui offre des performances robustes sur l'intervalle $I_{|\sigma(client)|}$. Nous rappelons que suivant notre condition de robustesse (Chapitre 1, section 1.2), le coût de ce plan ne doit excéder le coût du plan optimal de plus de λ. λ est la sous-optimalité acceptée d'un plan, à une position dans l'intervalle $I_{|\sigma(client)|}$, par rapport au plan optimal à cette position. Nous supposons que λ est égale à 20%. La figure 3.2 ci-après montre la variation des coûts des plans Plan1, Plan2 et Plan3 sur l'intervalle d'estimations $I_{|\sigma(client)|}$.

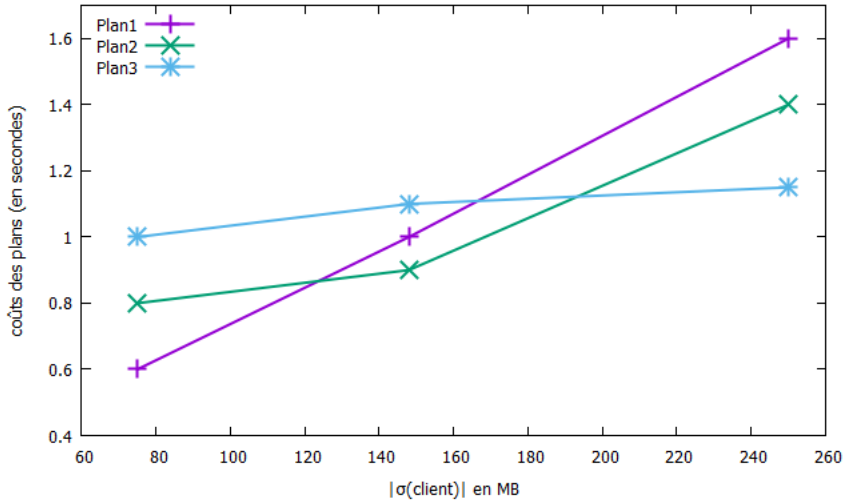


FIGURE 3.2 – Variation des coûts des plans en fonction de $|\sigma(\text{client})|$

Comme le montre cette figure, pour $\lambda = 20\%$, aucun de ces plans n'est robuste sur l'intervalle $I_{|\sigma(client)|}$. Chaque plan est robuste seulement sur une partie de l'intervalle. Une question importante concerne le choix -parmi ces plans- de celui par lequel commencer l'exécution. Au meilleur de notre connaissance, il n'existe pas dans la littérature de travaux proposant une solution à ce problème.

Dans l'objectif de résoudre ce problème, nous présentons dans cette section une stratégie

d'optimisation de requêtes qui permet de diviser -si nécessaire- un intervalle d'estimations en sous-intervalles et associer un plan robuste à chaque sous-intervalle. Un seul parmi ces plans doit être sélectionné pour démarrer l'exécution. Pour cela, nous calculons le risque que présente chaque plan, d'une violation de la robustesse à l'exécution. Le plan le moins risqué est choisi pour commencer l'exécution.

Dans le reste de cette section, nous détaillons notre stratégie de génération de plans robustes qui se compose de trois éléments : (1) **Sélection d'un ordonnancement robuste d'opérateurs**, (2) **Identification d'algorithmes physiques robustes**, et (3) **Choix d'un plan d'exécution**.

3.2.1 Sélection d'un ordonnancement robuste d'opérateurs

L'objectif de cet élément est d'ordonnancer les opérateurs d'une requête en tenant compte de l'incertitude sur les estimations des tailles de relations opérandes. Afin de décrire cet élément, nous considérons l'exemple suivant :

Exemple 3.2.1 Soit la requête Q :

Select ***
from *client, commande, article*
where *client.ID = commande.clientID* ***and***
commande.ID = article.commandeID ***and***
commande.prixtotal > [prixtotal] ***and***
article.prix < [prix] ;

Q est une requête de jointure entre les relations *client*, *commande*, et *article*. Ils existent des prédicats de sélection sur *commande.prixtotal* sous la forme $commande.prixtotal > [prixtotal]$, noté σ(commande), et sur *article.prix* sous la forme $article.prix < [prix]$, noté σ(article). Supposons que $|client| = 135$ MB, $|commande| = 220$ MB, $|article| = 380$ MB et que la mémoire disponible pour l'exécution de cette requête est de 150 MB.

Comme *client* est une relation de base, l'estimation de $|client|$ disponible à partir de

la méta-base est considérée précise par l'optimiseur. Nous supposons dans un premier temps qu'il existe dans la méta-base des informations précises concernant l'attribut *commande.prixtotal* (e.g., histogramme sur cet attribut). Ainsi, l'estimation de $|\sigma(commande)|$ est supposée précise. Soit $|\sigma(commande)|$=140MB. Nous supposons également qu'il n'existe pas dans la méta-base d'informations précises sur l'attribut *article.prix*. L'estimation de $|\sigma(article)|$ est alors considérée sujette à une erreur. Pour modéliser cela, nous supposons qu'un intervalle d'estimations, noté $I_{|\sigma(article)|}$, est calculé autour de $|\sigma(article)|$. Nous supposons que cet intervalle définit les limites inférieure et supérieure de la valeur de ce paramètre. Soient $I_{|\sigma(article)|}$ = [85, 175] MB.

Supposons qu'un optimiseur de requêtes énumère deux plans d'exécution possibles, P_{ac} et P_{cc} pour Q (Cf. Figure 3.3), tel que P_{ac} engendre des performances optimales pour les valeurs faibles de $|\sigma(\text{article})|$, tandis que P_{cc} engendre des performances optimales pour des valeurs importantes de $|\sigma(\text{article})|$.

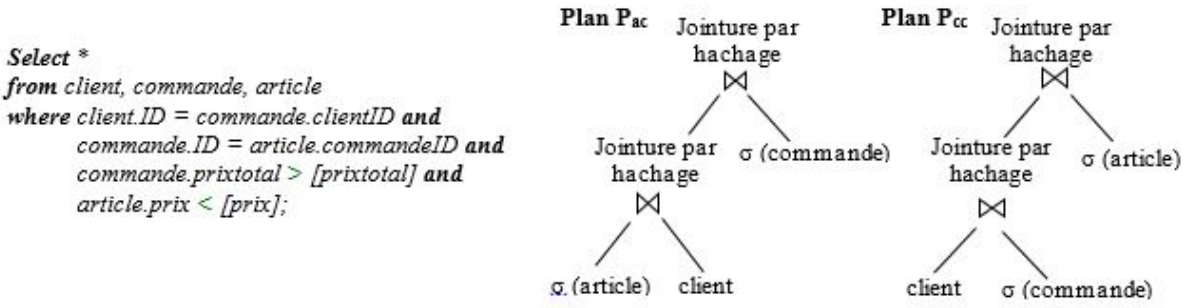


FIGURE 3.3 – Plans d'exécution possibles pour la requête Q

La figure 3.4 dans la page suivante montre la variation des coûts de ces plans en fonction de $|\sigma(article)|$.

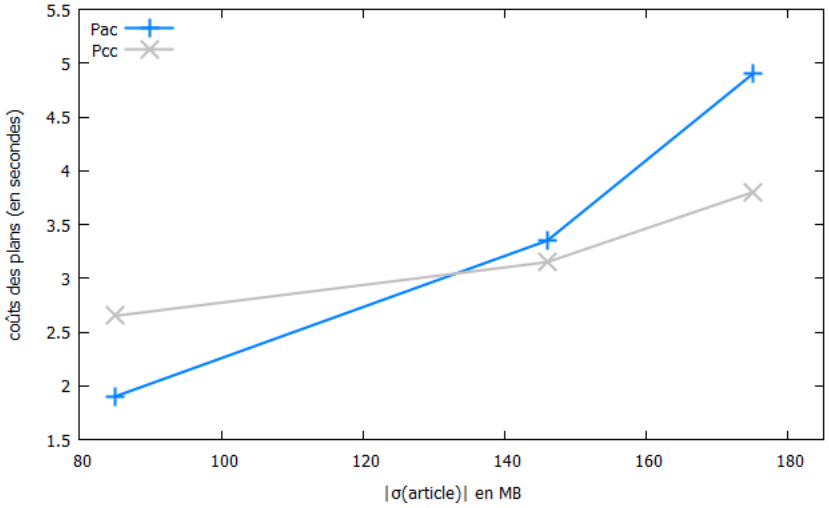


FIGURE 3.4 – Variation des coûts des plans en fonction de $|\sigma(\text{article})|$

La figure 3.4 met en évidence la difficulté d'engendrer un plan robuste sur un intervalle d'estimations quand celui-ci est large : en considérant uniquement la plage de valeurs [85, 163]MB, nous pouvons conclure que P_{ac} est un plan robuste puisque son coût n'excède pas 1.2 fois le coût optimal dans cet intervalle. Ainsi, P_{ac} peut être choisi pour exécuter la requête en toute sécurité. Cependant, en considérant l'intervalle [85, 175]MB, nous constatons que trouver un plan robuste sur cet intervalle est difficile puisque ni P_{ac} ni P_{cc} n'est robuste sur l'intégralité de cet intervalle.

Afin de produire un plan d'exécution robuste pour une requête, notre stratégie commence par spécifier un ordonanncement robuste des opérateurs de la requête. L'objectif est d'ordonnancer les opérateurs en tenant compte des incertitudes concernant les tailles des relations opérandes. Le terme taille est utilisé pour désigner la taille d'une relation en nombre d'octets ou en nombre de pages. Notons que le coût d'exécution d'un opérateur binaire (i.e. jointure) est une fonction des tailles des relations opérandes. La distinction entre les deux relations est cruciale car certaines formules de calcul du coût d'une jointure (e.g., jointure par produit cartésien) ne sont pas symétriques par rapport à l'ordre des relations "interne" et "externe" [SAC+79, Ioa96].

Nous rappelons que la relation externe correspond à la relation opérande à partir de la-

quelle les tuples sont lus dans un premier temps par l'algorithme de jointure. La relation interne correspond à la relation opérande à partir de laquelle les tuples sont lus en utilisant les valeurs de l'attribut de jointure de la relation externe. Pour produire un plan de requête ordonnancé, nous ordonnançons les opérateurs en mettant en premier les opérateurs de jointures entre les relations qui sont probablement les plus petites. En sortie de ce processus, nous avons un plan de requête sous forme d'un arbre d'opérateurs logiques. Trouver la relation la plus petite est simple lorsque les tailles de deux relations opérandes sont estimées en des points singuliers. Lorsque la taille d'une des relations (notée R_1) est modélisée par un intervalle d'estimations $\mathrm{I}_1 = [a, c]$ et qu'une estimation singulière est considérée pour la taille de la deuxième relation (notée R_2), nous calculons la probabilité que |R1| soit inférieure à |R2|, notée P(|R1| ≤ |R2|), comme suit :

$$P(|R_1| \leq |R_2|) = \begin{cases} 0 & \text{si } |R_2| < a \\ 1 & \text{si } |R_2| > c \\ \dfrac{|R_2| - a}{c - a} & \text{si } a \leq |R_2| \leq c \end{cases} \tag{3.1}$$

Les détails de ce calcul sont disponibles dans Annexe/A à la fin de ce document.

Jusqu'à présent, nous avons supposé dans notre exemple 3.2.1 que seulement $|\sigma(article)|$ est sujette à une erreur d'estimation et que des estimations précises de $|client|$ et $|\sigma(commande)|$ sont disponibles. Supposons maintenant qu'il n'existe pas dans la méta-base d'informations précises sur l'attribut *commande.prixtotal*. Pour prendre en considération l'incertitude concernant l'estimation de $|\sigma(commande)|$, nous supposons qu'un intervalle d'estimations, noté $\mathrm{I}_{|\sigma(commande)|}$, est calculé autour de $|\sigma(commande)|$. Soit $\mathrm{I}_{|\sigma(commande)|}$=[121, 150]MB. Dans ce cas, trouver la relation la plus petite nécessite -entre autres- de déterminer la probabilité que $|\sigma(article)|$ soit inférieure à $|\sigma(commande)|$.
Lorsque les tailles de deux relations à comparer sont représentées par des intervalles d'estimations, nous calculons pour une relation sa probabilité d'être la relation la plus petite en utilisant la règle suivante : *soient deux intervalles* $I_1 = [a, c]$ *et* $I_2 = [b, d]$ *d'estimations*

des tailles de deux relations R_1 et R_2. La probabilité que $|R1|$ soit inférieure à $|R2|$, notée $P(|R1| \leq |R2|)$, est donnée par :

$$P(|R_1| \leq |R_2|) = \begin{cases} \frac{b-a}{c-a} + \frac{d-c}{d-b} - \frac{b-a}{c-a} \times \frac{d-c}{d-b} + \frac{1}{2} & \text{si } b \in [a,c] \text{ et } d > c \\ \frac{1}{2} - \frac{a-b}{d-b} - \frac{c-d}{c-a} + \frac{a-b}{d-b} \times \frac{c-d}{c-a} & \text{si } a \in [b,d] \text{ et } c > d \\ \frac{1}{2} & \text{si } [a,c] = [b,d] \\ \frac{1}{2} + \frac{d-c}{d-b} - \frac{a-b}{d-b} & \text{si } [a,c] \subset [b,d] \\ \frac{1}{2} + \frac{b-a}{c-a} - \frac{c-d}{c-a} & \text{si } [b,d] \subset [a,c] \end{cases} \quad (3.2)$$

Les détails de ce calcul sont disponibles dans Annexe/B à la fin de ce document.

En appliquant ces formules avec en entrées les intervalles $I_{|\sigma(article)|} = [85, 175]$MB et $I_{|\sigma(commande)|} = [121, 150]$MB de notre exemple 3.2.1, l'ordonnancement choisi pour les opérateurs de la requête de notre exemple, est celui dans la figure 3.5 ci-dessous.

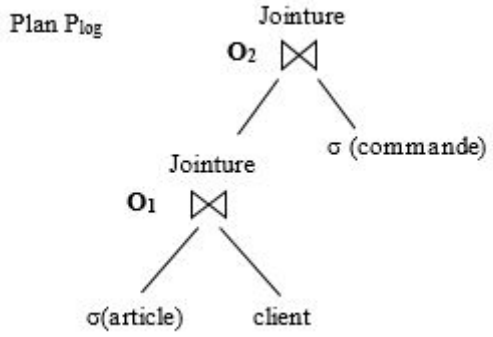


FIGURE 3.5 – Plan d'opérateurs logiques ordonnancés pour la requête Q

Une fois un ordonnancement est spécifié pour les opérateurs d'une requête, l'étape suivante consiste à associer un algorithme physique robuste à chaque opérateur logique. Nous détaillons cela dans la sous-section suivante.

3.2.2 Identification d'algorithmes physiques robustes

L'étape suivante dans notre stratégie de génération de plans robustes applique -à chaque opérateur dans le plan d'opérateurs logiques ordonnancés associé à une requête- le processus suivant : des algorithmes physiques sont énumérés et les coûts d'exécution de ces algorithmes sont comparés pour les valeurs/intervalles d'estimations des relations opérandes. L'objectif est d'identifier des algorithmes physiques robustes pour chaque opérateur. Lorsque la taille d'une relation opérande est estimée par un intervalle, nous calculons l'intervalle/sous-intervalle de robustesse de chaque algorithme énuméré.
Déterminer un (sous-)intervalle de robustesse peut être converti en un problème de résolution numérique. Reprenons notre exemple 3.2.1. Soit O_1 le premier opérateur de jointure dans le plan P_{log} (Cf. Figure 3.5, Page 76). C'est l'opérateur de jointure entre la relation temporaire résultante du prédicat de sélection sur la relation *article*, i.e., $\sigma(article)$, et la relation *client*. Soit O_{alt} un algorithme de jointure physique candidat à l'exécution de O_1. Déterminer la limite inférieure de l'intervalle/sous-intervalle de robustesse de O_{alt}, noté I_{alt}, sur $I_{|\sigma(article)|}$ revient à résoudre l'inéquation :

$$\begin{aligned} &Coût(O_{alt}, |\sigma(article)|, |client|) \leq \\ &(1 + \frac{\lambda}{100}) \times CoûtOptimal(S, |\sigma(article)|, |client|) \end{aligned} \tag{3.3}$$

$|\sigma(article)|$ est la variable de l'inéquation. La racine minimale de cette inéquation correspond à la limite inférieure de I_{alt}. $|client|$ est une valeur constante dans la fonction $Coût(O_{alt}, |\sigma(article)|, |client|)$. Cette fonction retourne le coût d'exécution de O_{alt} pour les estimations singulières de $|\sigma(article)|$ et $|client|$. $CoûtOptimal(S, |\sigma(article)|, |client|)$ est une fonction qui retourne le coût optimal dans un ensemble S d'algorithmes de jointure physiques candidats à l'exécution de O_1. Cette inéquation est résolue dans l'intervalle $I_{|\sigma(article)|}$. Pour cela, nous parcourons $I_{|\sigma(article)|}$ et vérifions pas par pas si la valeur retournée par $Coût(O_{alt}, |\sigma(article)|, |client|)$ est supérieure à la valeur retournée par $(1+\frac{\lambda}{100}) \times CoûtOptimal(S, |\sigma(article)|, |client|)$. Si c'est le cas, alors nous continuons à avancer dans

l'intervalle $I_{|\sigma(article)|}$. Nous nous arrêtons lorsque $Coût(O_{alt}, |\sigma(article)|, |client|)$ devient inférieure ou égale à $(1 + \frac{\lambda}{100}) \times CoûtOptimal(S, |\sigma(article)|, |client|)$. Le point d'arrêt définit la racine minimale de l'inéquation (3.3). Ce point détermine la limite inférieure de l'intervalle de robustesse de l'algorithme de jointure physique O_{alt} sur $I_{|\sigma(article)|}$.

Nous illustrons ceci dans Algorithme 1 présenté dans la page suivante. Algorithme 1 prend en entrée un algorithme de jointure physique noté O (O_{alt} dans notre exemple), un intervalle I_1 ($I_{|\sigma(article)|}$ dans notre exemple) d'estimations de la taille d'une relation temporaire T (la relation résultante de $\sigma(article)$ dans notre exemple), une estimation singulière de $|T|$ -notée $|T|_{est}$, et qui est initialisée à la limite inférieure de I_1, une estimation singulière de la taille d'une deuxième relation opérande R, notée $|R|_{est}$ tel que R est une relation de base (*client* dans notre exemple), un ensemble S d'algorithmes de jointure physiques candidats à l'exécution de l'opérateur logique en question (O_1 dans notre exemple), et un seuil λ de la sous-optimalité tolérée.

Algorithme 1 retourne la limite inférieure de l'intervalle de robustesse de O. Il parcourt l'intervalle d'estimations en entrée I_1 et compare pas par pas dans cet intervalle le coût de O au coût optimal dans l'ensemble S. Chaque pas détermine un point spécifique dans I_1. Ce point correspond à une estimation singulière de |T|, notée $|T|_{est}$. Afin d'éviter une grande complexité de calculs, nous calculons le pas en s'appuyant sur la méthode de la sécante [JMP13]. Nous détaillons ceci après avoir décrit les algorithmes de détermination des limites inférieure et supérieure d'un intervalle de robustesse.

Entrées : O, I_1, $|T|_{est}$, $|R|_{est}$, S, λ ;

Sorties : limite-inférieure ; *//de l'intervalle de robustesse de O*

méthodeDiverge := faux ;
tant que *Coût(O,* $|T|_{est}$*,* $|R|_{est}$*)* $> (1 + \frac{\lambda}{100})\times$ *CoûtOptimal(S,* $|T|_{est}$*,* $|R|_{est}$*)* **et** $|T|_{est}$ < *limite supérieure de* I_1 **et** *NON(méthodeDiverge)* **faire**
    $Diff_1$:= Coût(O, $|T|_{est}$, $|R|_{est}$) - $(1+\frac{\lambda}{100})\times$ CoûtOptimal(S, $|T|_{est}$, $|R|_{est}$);
    $|T|_{est} := |T|_{est} \times 1{,}2$;
    $Diff_2$:= Coût(O, $|T|_{est}$, $|R|_{est}$) - $(1+\frac{\lambda}{100})\times$ CoûtOptimal(S, $|T|_{est}$, $|R|_{est}$);
    si *(*$Diff_2 \geq 0$ **et** $Diff_2 \leq Diff_1$*)* **alors**
        $|T|_{est} := |T|_{est}\times (1 + \frac{Diff_2}{6\times(Diff_1 - Diff_2)})$;
    sinon
        méthodeDiverge := vrai ;
    fin
fin
limite-inférieure := $|T|_{est}$;

Algorithme 1 : Détermination de la limite inférieure d'un intervalle de robustesse

Notons que Algorithme 1 s'arrête dès que nous obtenons une inversion des coûts (i.e., $Diff_2 < 0$) ou s'il diverge. La divergence est définie par un écart de plus en plus grand entre le coût de l'algorithme physique en entrée et le coût optimal (i.e., $Diff_2$ devient supérieure à $Diff_1$ dans Algorithme 1).
Si la valeur de $|T|_{est}$ retournée par Algorithme 1 excède la limite supérieure de l'intervalle d'estimations en entrée I_1, alors l'algorithme de jointure physique O est présumé non robuste. Il est ainsi rejeté. Dans le cas contraire, nous déterminons d'une manière similaire à Algorithme 1, la limite supérieure de l'intervalle de robustesse de O. Reprenons notre exemple 3.2.1. O_{alt} est un algorithme physique candidat à l'exécution de l'opérateur de jointure O_1. Nous déterminons la limité supérieure de l'intervalle de robustesse de O_{alt},

noté I_{alt}, en résolvant l'inéquation :

$$Coût(O_{alt}, |\sigma(article)|, |client|) > \tag{3.4}$$
$$(1 + \frac{\lambda}{100}) \times CoûtOptimal(S, |\sigma(article)|, |client|)$$

Résoudre cette inéquation, c'est également parcourir l'intervalle d'estimations $I_{|\sigma(article)|}$, comparer pas par pas le coût de O_{alt} au coût optimal dans S et trouver la valeur minimale pour laquelle l'inégalité est vraie. Le parcours de $I_{|\sigma(article)|}$ commence à partir du point correspondant à la limite inférieure retournée par Algorithme 1. Algorithme 2 ci-après illustre la détermination de la limite supérieure d'un intervalle de robustesse. Les entrées à cet algorithme : O, I_1, $|R|_{est}$, S, et λ, sont les mêmes entrées décrites pour Algorithme 1. L'entrée $|T|_{est}$ est la valeur retournée par Algorithme 1.

Entrées : O, I_1, $|T|_{est}$, $|R|_{est}$, S, λ ;

Sorties : limite-supérieure //de l'intervalle de robustesse de O ;

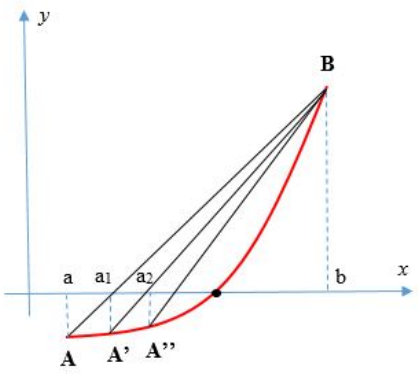
méthodeDiverge := faux ;
tant que *Coût(O,* $|T|_{est}$*,* $|R|_{est}$*)* $\leq (1 + \frac{\lambda}{100})\times$ *CoûtOptimal(S,* $|T|_{est}$*,* $|R|_{est}$*)* **et** $|T|_{est}$ < *limite supérieure de* I_1 **et** *NON(méthodeDiverge)* **faire**
    Diff$_1$:= $(1+\frac{\lambda}{100})\times$ CoûtOptimal(S, $|T|_{est}$, $|R|_{est}$) - Coût(O, $|T|_{est}$, $|R|_{est}$) ;
    $|T|_{est}$:= $|T|_{est}$ × 1,2;
    Diff$_2$:= $(1+\frac{\lambda}{100})\times$ CoûtOptimal(S, $|T|_{est}$, $|R|_{est}$) - Coût(O, $|T|_{est}$, $|R|_{est}$) ;
    si *(Diff*$_2$ $\leq$ *Diff*$_1$ **et** *Diff*$_2$ ≥ 0*)* **alors**
        $|T|_{est}$:= $|T|_{est}\times$ $(1 + \frac{Diff_2}{6\times(Diff_1 - Diff_2)})$;
    sinon
        méthodeDiverge = vrai ;
    fin
fin
limite-supérieure := $|T|_{est}$;

Algorithme 2 : Détermination de la limite supérieure d'un intervalle de robustesse

Dans les deux algorithmes présentés précédement, la comparaison des coûts se fait en avançant pas par pas dans l'intervalle d'estimations en entrée I_1. Afin d'éviter une grande complexité de calculs, ce pas est calculé en s'appuyant sur le principe de la méthode de la sécante [JMP13]. Nous nous sommes basés sur le principe de cette méthode du fait qu'elle permet d'obtenir des résultats relativement précis avec une complexité réduite [JMP13]. La méthode de la sécante est un algorithme de recherche de racines d'une fonction f. Elle permet de trouver des approximations précises des solutions x* sur un intervalle [a, b] de l'équation f(x)=0.

Principe de la sécante

Pour une fonction f continue sur un intervalle [a, b], et vérifiant f(a) < 0, f(b) > 0, on trace le segment [AB] où A = (a, f (a)) et B = (b, f (b)). Si le segment reste au-dessus du graphe de f alors la fonction s'annule sur l'intervalle [a_1, b] où (a_1, 0) est le point d'intersection de la droite (AB) avec l'axe des abscisses. La droite (AB) s'appelle la sécante. On réitère en partant maintenant de l'intervalle [a_1, b] pour obtenir une valeur a_2.



La méthode de la sécante est basée sur la relation de récurrence :

$$a_{n+1} = a_n - \frac{a_n - a_{n-1}}{f(a_n) - f(a_{n-1})} \times f(a_n) \tag{3.5}$$

L'initialisation de cette méthode nécessite deux points a_0 et a_1. Il n'est pas nécessaire que a_0 et a_1 encadrent une racine de f. Plusieurs itérations de la relation de récurrence peuvent être effectuées afin de trouver la/les racine(s) d'une fonction f. L'arrêt des itérations peut être conditionné -par exemple- par un nombre maximal d'itérations, ou par un intervalle délimitant la/les racine(s).

Nous avons étendu le principe de la méthode de la sécante à notre calcul. Nous avons utilisé cette relation de récurrence dans nos algorithmes pour itérer le pas dans l'intervalle d'estimations en entrée I_1. A la première itération (n=1) dans Algorithme 1 et Algorithme 2, nous supposons que :

- a_1 correspond à la valeur actuelle de $|T|_{est}$.
- a_0 correspond à l'ancienne valeur de $|T|_{est}$ qui est égale à $\frac{|T|_{est}}{1.2}$ puisqu'à chaque fois on multiplie $|T|_{est}$ par 1.2.
- $f(a_1)$ correspond à la différence actuelle des coûts, i.e., $Diff_2$.
- $f(a_0)$ correspond à l'ancienne différence des coûts, i.e., $Diff_1$.

- a_2 correspond à $|T|_{est} \times (1 + \frac{Diff_2}{6 \times (Diff_1 - Diff_2)})$, et est calculé comme suit : en supposant que n = 1 dans la relation de récurrence nous obtenons :

$$a_2 = a_1 - \frac{a_1 - a_0}{f(a_1) - f(a_0)} \times f(a_1)$$

En remplaçant a_1 par $|T|_{est}$, $f(a_1)$ par $Diff_2$, a_0 par $\frac{|T|_{est}}{1.2}$, et $f(a_0)$ par $Diff_1$, nous obtenons :

$$a_2 = |T|_{est} - \frac{|T|_{est} - \frac{|T|_{est}}{1.2}}{Diff_2 - Diff_1} \times Diff_2$$

En factorisant par $|T|_{est}$, nous obtenons :

$$a_2 = |T|_{est} \times (1 - \frac{\frac{0.2}{1.2}}{Diff_2 - Diff_1} \times Diff_2)$$

$$a_2 = |T|_{est} \times (1 + \frac{\frac{1}{6}}{Diff_1 - Diff_2} \times Diff_2)$$

$$a_2 = |T|_{est} \times (1 + \frac{Diff_2}{6 \times (Diff_1 - Diff_2)})$$

Dans le cas général (n $\geq$ 1), nous calculons le pas en supposons que :

- a_n correspond à $|T|_{est}$.
- a_{n-1} correspond à $\frac{|T|_{est}}{1.2}$.
- $f(a_n)$ correspond à $Diff_2$.
- $f(a_{n-1})$ correspond à $Diff_1$.
- a_{n+1} correspond à $|T|_{est} \times (1 + \frac{Diff_2}{6 \times (Diff_1 - Diff_2)})$, et qui remplacera $|T|_{est}$ dans l'itération suivante.

Algorithme 1 et Algorithme 2 opèrent sous la condition que la taille d'une seule des relations opérandes d'un opérateur en entrée, est modélisée par un intervalle d'estimations. Cependant, les deux relations opérandes d'une jointure peuvent être deux relations temporaires, dont les tailles sont modélisées -chacune- par un intervalle d'estimations. L'intersection de ces intervalles forme ce que nous appelons une surface d'estimations.

Considérons toujours notre exemple 3.2.1. Soit O_2 l'opérateur de jointure dans le plan P_{log} (Cf. Figure 3.5, Page 76), entre la relation temporaire résultante de la jointure entre $\sigma(article)$ et *client*, et la relation temporaire résultante du prédicat de sélection sur *commande* (i.e., $\sigma(commande)$). Supposons qu'un intervalle d'estimations, noté $I_{|\sigma(article)\bowtie client|}$, est calculé autour de $|\sigma(article) \bowtie client|$. $I_{|\sigma(commande)|}$ est l'intervalle d'estimations autour de $|\sigma(commande)|$. L'intersection de ces deux intervalles forme ce que nous appelons une surface d'estimations comme le montre la figure 3.6 ci-dessous.

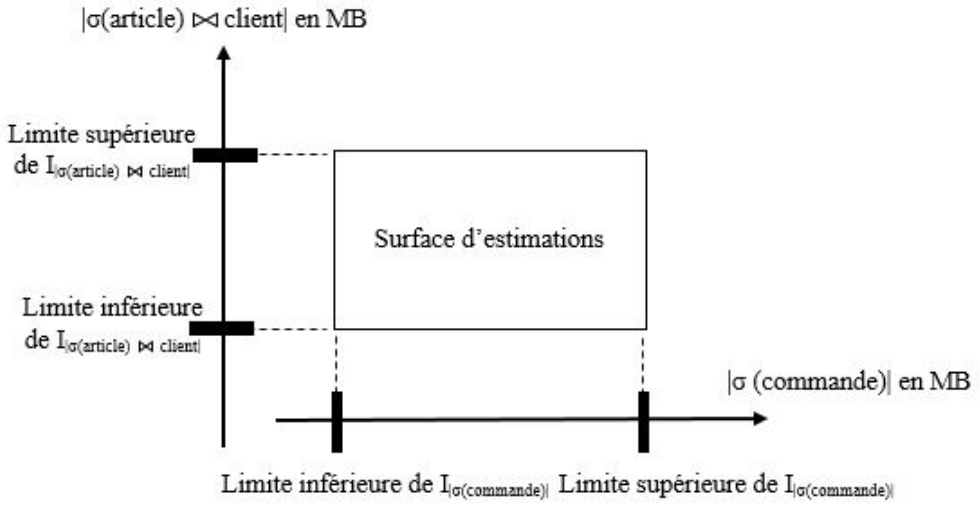


FIGURE 3.6 – Surface d'estimations formée par $I_{|\sigma(article)\bowtie client|}$ et $I_{|\sigma(commande)|}$

Cette surface d'estimations permet d'identifier des algorithmes physiques robustes pour l'exécution de O_2. Ainsi, au lieu de déterminer un intervalle de robustesse, l'objectif est désormais de déterminer une surface de robustesse pour chaque algorithme de jointure physique candidat à l'exécution de O_2.

Une surface de robustesse est définie par deux coordonnées C_1 et C_2. Chaque coordonnée est un intervalle de robustesse. Une surface de robustesse n'est qu'une région dans une surface d'estimations initiale. Ainsi, pour illustrer la définition d'une surface de robustesse, nous pouvons prendre comme exemple la surface d'estimations dans la figure 3.6 ci-dessus. Les coordonnées de cette surface sont : $C_1 = [$*limite inférieure de* $I_{|\sigma(commande)|}$,

limite supérieure de $I_{|\sigma(commande)|}$] et C_2 = [*limite inférieure de* $I_{|\sigma(article)\bowtie client|}$, *limite supérieure de* $I_{|\sigma(article)\bowtie client|}$]. Le calcul des limites inférieures des coordonnées d'une surface de robustesse est illustré dans Algorithme 3 dans la page suivante. Cet algorithme prend en entrée un algorithme physique noté O, un intervalle I_1 d'estimations de la taille d'une première relation opérande, un intervalle I_2 d'estimations de la taille d'une deuxième relation opérande, un ensemble S d'algorithmes physiques possibles associés à l'opérateur de jointure logique en question, et finalement un seuil λ de la sous-optimalité acceptée. Algorithme 3 retourne les limites inférieures des coordonnées de la surface de robustesse associée à O. Pour cela, il parcourt les intervalles I_1 et I_2 et compare pas par pas, le coût de O au coût optimal dans S. I_1 et I_2 n'étant pas forcément égaux, le parcourt de l'un de ces intervalles peut se terminer avant que le parcourt de l'autre intervalle le soit. Dans ce cas, nous fixons le pas dans l'intervalle totalement parcouru et continuons à avancer dans l'autre intervalle en faisant appel à Algorithme 1, comme illustré dans Algorithme 3 dans la page suivante (lignes 16=>23).

Entrées : O, I_1, I_2, S, λ

Sorties : limite-inférieure-C1, limite-inférieure-C2 ; *//de la surface de robustesse de O*;

méthodeDiverge := faux ;
$|T1|_{est}$:= limite inférieure de I_1 ;
$|T2|_{est}$:= limite inférieure de I_2 ;
tant que *Coût(O,* $|T1|_{est}$, $|T2|_{est}$*)* $> (1 + \frac{\lambda}{100})\times$ *CoûtOptimal(S,* $|T1|_{est}$, $|T2|_{est}$*)* **et** *NON(méthodeDiverge)* **et** $|T1|_{est}$ < *limite supérieure de* I_1 **et** $|T2|_{est}$ < *limite supérieure de* I_2 **faire**
    Diff_1 := Coût(O, $|T1|_{est}$, $|T2|_{est}$)-$(1+\frac{\lambda}{100})\times$ CoûtOptimal(S, $|T1|_{est}$, $|T2|_{est}$) ;
    $|T1|_{est} := |T1|_{est} \times 1{,}2$;
    $|T2|_{est} := |T2|_{est} \times 1{,}2$;
    Diff_2 := Coût(O, $|T1|_{est}$, $|T2|_{est}$)-$(1+\frac{\lambda}{100})\times$ CoûtOptimal(S, $|T1|_{est}$, $|T2|_{est}$) ;
    si *(*$Diff_2 \leq Diff_1$ **et** $Diff_2 \geq 0$*)* **alors**
        $|T1|_{est} := |T1|_{est} \times (1 + \frac{Diff_2}{6\times(Diff_1 - Diff_2)})$;
        $|T2|_{est} := |T2|_{est} \times (1 + \frac{Diff_2}{6\times(Diff_1 - Diff_2)})$;
    sinon
        méthodeDiverge = vrai ;
    fin
fin
si $|T1|_{est}$ > *limite supérieure de* I_1 **alors**
    $|T1|_{est}$:= limite supérieure de I_1 ;
    $|T2|_{est}$:= Algorithme 1 (O, I_2, $|T2|_{est}$, $|T1|_{est}$, S, λ) ;
fin
si $|T2|_{est}$ > *limite supérieure de* I_2 **alors**
    $|T2|_{est}$:= limite supérieure de I_2 ;
    $|T1|_{est}$:= Algorithme 1 (O, I_1, $|T1|_{est}$, $|T2|_{est}$, S, λ) ;
fin
limite-inférieure-C1 := $|T1|_{est}$;
limite-inférieure-C2 := $|T2|_{est}$;

Algorithme 3 : Calcul des limites inférieures d'une surface de robustesse

Si les valeurs $|\mathrm{T1}|_{est}$ et $|\mathrm{T2}|_{est}$ retournées par Algorithme 3 excèdent les limitent supérieures des intervalles d'estimations en entrée I_1 et I_2, alors O est présumé non robuste. Il est ainsi rejeté. Dans le cas contraire, nous déterminons d'une manière similaire à Algorithme 3, les limites supérieures de la surface de robustesse de l'algorithme physique O. Pour cela, nous parcourons la surface d'estimations en partant des points $|T1|_{est}$ et $|T2|_{est}$ retournées par Algorithme 3, et cherchons les coordonnées minimales pour lesquelles l'inégalité $Co\hat{u}t(O, |T1|_{est}, |T2|_{est}) > (1 + \frac{\lambda}{100}) \times Co\hat{u}tOptimal(S, |T1|_{est}, |T2|_{est})$ est vraie. Plutôt que de faire appel à Algorithme 1 (lignes 18 et 22 dans Algorithme 3), nous utilisons Algorithme 2. Outre ces changements, le mécanisme de calcul reste identique à celui de Algorithme 3.

A ce stade dans notre méthode d'optimisation, des algorithmes physiques robustes sont identifiés comme candidats à l'exécution de chaque opérateur dans un plan de requête. Un seul parmi ces algorithmes robustes doit être ensuite sélectionné pour chaque opérateur afin de générer un plan robuste initial. Une question se pose : comment établir un plan robuste initial ? La sous-section suivante est consacrée à répondre à cette question.

3.2.3 Choix d'un plan d'exécution

Pour générer un plan robuste initial, nous nous basons sur une approche probabiliste. Nous calculons pour chaque algorithme physique robuste associé à un opérateur logique, le risque qu'il présente d'une violation de la robustesse à l'exécution. Puisqu'un opérateur de jointure possède deux relations opérandes, nous distinguons deux cas possibles pour le calcul de ce risque :

i. la taille d'une seule des relations opérandes est représentée par un intervalle d'estimations
ii. la taille de chacune des relations opérandes est représentée par un intervalle d'estimations

Dans le premier cas (i), le risque d'une violation de la robustesse est défini comme suit : *Soit un algorithme physique O_{alt}, avec un intervalle de robustesse I_{alt} sur un intervalle d'estimations initial I. Le risque $\Re(O_{alt}, I)$ de O_{alt} sur I est donné par :*

$$\Re(O_{alt}, I) = 1 - \frac{amplitude\ de\ I_{alt}}{amplitude\ de\ I} \tag{3.6}$$

Le risque est défini par le complémentaire du quotient entre l'amplitude de l'intervalle I_{alt} de robustesse de O_{alt} et l'amplitude de l'intervalle I d'estimations de la taille de la relation opérande en question.

L'algorithme physique ayant le risque optimal, noté R(O_{alt}, I) est celui qui fournit le risque minimal d'une violation de la robustesse à l'exécution. Il est celui dont l'intervalle de robustesse couvre le plus de valeurs dans l'intervalle d'estimations initial. Le risque minimal est défini comme suit : *Soient S un ensemble d'algorithmes physiques candidats à l'exécution d'un opérateur logique dans un plan de requête, et I l'intervalle d'estimations de la taille d'une relation opérande. La minimisation du risque est définie par :*

$$R(S, I) = min_{O_{alt} \in S}(\Re_{O_{alt} \in S}, I) \tag{3.7}$$

L'objectif est de déterminer l'algorithme physique qui réduit le risque d'une violation de la robustesse à l'exécution, sur l'intervalle d'estimations initial.

Dans ce premier cas (i), seulement la taille d'une des relations opérandes d'une jointure est modélisée par un intervalle d'estimations. Dans le cas (ii) où la taille de chacune des relations opérandes est modélisée par un intervalle d'estimations, le risque d'une violation de la robustesse est alors défini comme suit : *Soit un algorithme physique O_{alt}, avec une surface de robustesse S_{alt}, et une surface d'estimations initiale S_{est}. Le risque $\Re(O_{alt}, S_{est})$ de O_{alt} sur S_{est} est donné par :*

$$\Re(O_{alt}, S_{est}) = 1 - \frac{surface\ de\ S_{alt}}{surface\ de\ S_{est}} \tag{3.8}$$

Dans ce cas, le risque est défini par le complémentaire du quotient entre la surface de robustesse S_{alt} associée à O_{alt} et la surface d'estimations initiale. L'objectif est également d'identifier l'algorithme physique qui minimise le risque d'une violation de la robustesse à l'exécution.

Lorsque le risque associé à un algorithme physique est non null, le choix de cet algorithme peut résulter en une violation de la robustesse à l'exécution. Même si ce risque est réduit, il est existant et peut avoir un effet néfaste sur la qualité de l'évaluation de requêtes. Pour éviter cela, nous proposons comme solution d'enrichir le plan initialement choisi par des opérateurs dits de contrôle et de décision. Ces opérateurs surveillent l'exécution du plan. Ils apportent -si besoin- les corrections nécessaires pour assurer une robustesse jusqu'à la fin de l'exécution, sans faire appel à l'optimiseur de requêtes.
Nous détaillons ceci dans la section suivante qui décrit le deuxième module de notre méthode d'optimisation hybride.

3.3 Correction de plans d'exécution

Le module précédent décrit une stratégie d'optimisation qui se focalise exclusivement sur le choix -à l'optimisation- d'un plan estimé robuste pour exécuter une requête. Bien que ce plan soit établi avec un risque réduit d'une violation de la robustesse à l'exécution, le risque est existant et peut avoir un effet néfaste sur la robustesse.
Pour illustrer ceci, considérons l'exemple de la requête dans la section 3.2. Comme nous pouvons l'observer dans la figure 3.2 (page 71), l'intervalle de robustesse du plan Plan1 est [75, 200]MB, celui du Plan2 est [90, 235]MB, et celui de Plan3 est [160, 250]MB. Plan2 présente une probabilité plus élevée d'éviter une violation de la robustesse puisque son intervalle de robustesse couvre le plus de valeurs susceptibles d'être observées à l'exécution. Plan2 peut être choisi pour commencer l'exécution. Si à l'exécution du premier opérateur (i.e., $\sigma(client)$), la valeur constatée de $|\sigma(article)|$ est égale à 240MB, continuer

d'exécuter Plan2 résulterait en une violation de la robustesse. Une solution à ceci serait d'interagir avec les conditions observées à l'exécution. En effet, collecter des informations lors de l'exécution d'un plan et utiliser celles-ci pour assurer l'adéquation de ce plan, peut améliorer les performances d'une manière significative [MRS+04]. Dans cette perspective, nous proposons dans le cadre de cette thèse, une méthode capable de détecter et de corriger une violation de la robustesse d'un plan pendant l'exécution. Cette méthode utilise des opérateurs de contrôle et de décision, insérés à des points spécifiques dans le plan d'exécution robuste choisi pour exécuter une requête. Ces opérateurs réagissent aux informations collectées à l'exécution afin d'adapter -si nécessaire- le plan en cours d'exécution et garantir ainsi une robustesse jusqu'à la terminaison.

Dans le reste de cette section, nous détaillons notre méthode de correction de plans d'exécution. Nous commençons par décrire comment des opérateurs de contrôle et de décision peuvent être insérés dans un plan d'exécution. Puis nous présentons comment ces opérateurs permettent de corriger une violation de la robustesse sans avoir besoin de rappeler l'optimiseur de requêtes.

3.3.1 Insertion d'opérateurs de contrôle et de décision

A ce stade dans notre méthode d'optimisation, un plan d'exécution robuste pour une requête est produit, en spécifiant un ordonnancement des opérateurs et en associant à chaque opérateur un algorithme physique probablement robuste. La robustesse est évaluée dans les intervalles calculés autours des estimations des tailles des relations opérandes. Un algorithme physique choisi pour un opérateur logique de la requête peut être robuste pour toutes les valeurs d'un intervalle d'estimations ou bien seulement pour quelques valeurs.
Lorsqu'un algorithme physique est robuste pour quelques valeurs d'un intervalle, il risque d'en résulter une violation de la robustesse si les valeurs observées à l'exécution ne correspondent pas à celles pour lesquelles cet algorithme est estimé robuste. Une solution

à ce problème serait de surveiller l'exécution du plan afin de détecter une violation de la robustesse. Dans les méthodes d'optimisation existantes qui abordent cette question, des opérateurs de collecte de statistiques sont insérés dans le plan d'exécution lors de l'optimisation (e.g., [KD98, MRS+04]). Ces opérateurs contrôlent l'exécution de ce plan. Si une sous-optimalité du plan est détectée, l'optimiseur de requêtes est ré-invoqué pour corriger cette sous-optimalité. La limite que présente ces méthodes c'est qu'un optimiseur de requêtes peut être invoqué plusieurs fois pendant l'exécution d'une requête. Ceci est essentiellement du à la complexité d'obtenir des estimations précises -à l'optimisation et lors d'une ré-optimisation. Ainsi, à chaque invocation, l'optimiseur peut faire un choix sous-optimal d'un plan d'exécution, ce qui peut engendrer de mauvaises performances.
Dans ce document, nous proposons une solution alternative pour la correction d'une violation de la robustesse à l'exécution. Nous proposons d'enrichir le plan robuste généré par le premier module de notre méthode d'optimisation, par des opérateurs appelés «opérateurs de contrôle et de décision». Ces opérateurs ont pour double rôle de surveiller l'exécution du plan et prendre -si nécessaire- la décision de modification du reste du plan pour corriger une violation de la robustesse constatée. Ces opérateurs réagissent sans avoir besoin de rappeler l'optimiseur. Ils sont capables de réagir d'une manière autonome en se basant sur leurs propres données. Une partie des données est déterminée au moment de l'optimisation (c-à-d. les intervalles de robustesse associés aux algorithmes physiques énumérés pour les opérateurs de la requête). Les autres données sont recueillies lors de l'exécution (c-à-d. des statistiques actualisées).
Insérer des opérateurs de contrôle et de décision à de nombreux points dans un plan d'exécution et vérifier la robustesse de ce plan à chaque point peut induire un surcoût important. Pour éviter ceci, notre méthode d'insertion d'opérateurs de contrôle et de décision détermine les points dans le plan où le risque d'une violation de la robustesse est considérable. Notre méthode commence par attribuer un niveau d'incertitude, qui peut être ou bien « faible » ou bien « élevé », aux différents algorithmes physiques dans le plan initialement établi pour exécuter une requête. Un niveau d'incertitude élevé indique qu'il

y a un risque important que l'algorithme choisi conduise à une violation de la robustesse. Le niveau d'incertitude est attribué en utilisant un paramètre utilisateur, noté δ et appelé seuil de risque. Un seuil de risque définit la valeur percentile maximale du risque d'une violation de la robustesse que l'utilisateur est prêt à accepter. Prenons comme exemple un algorithme physique O_{alt} choisi pour l'exécution d'un opérateur logique O du fait qu'il possède la probabilité la plus élevée d'éviter une violation de la robustesse. Supposons que cette probabilité est égale à 0.8. Le risque d'une violation de la robustesse lors de l'exécution de O_{alt} est alors égale à 0.2. Si le seuil de risque est de 30%, le choix de O_{alt} est considéré suffisamment fiable, le niveau d'incertitude attribué à O_{alt} est faible. L'utilisation d'un seuil de risque de 10% se traduirait par un niveau d'incertitude élevé pour O_{alt}.

Après avoir décrit le principe du seuil de risque, nous abordons maintenant la question de l'insertion des opérateurs de contrôle et de décision dans un plan d'exécution. Pour illustrer ceci, nous reprenons notre exemple de requête dans la section 3.2. Supposons toujours que l'estimation de $|\sigma(client)|$ est considérée sujette à une erreur. Les plans Plan1, Plan2 et Plan3 (Cf. Figure 3.1, Page 70) sont des plans d'exécution possibles pour la requête en question. Comme nous pouvons l'observer dans la figure 3.2 (Page 71), l'intervalle de robustesse du plan Plan1 est [75, 200]MB, celui du Plan2 est [90, 235]MB, et celui de Plan3 est [160, 250]MB. Le plan qui minimise le risque est Plan2 ($\Re(Plan1, I_{|\sigma(article)|})$= 0.36, $\Re(Plan2, I_{|\sigma(article)|})$= 0.23, $\Re(Plan3, I_{|\sigma(article)|})$=0.55). Plan2 est initialement choisi pour exécuter la requête. Supposons que le seuil de risque est fixé par l'utilisateur à 20%. De ce fait, l'incertitude sur la robustesse du plan choisi est élevée. Un opérateur de contrôle et de décision est alors inséré dans le plan comme le montre la figure 3.7 dans la page suivante. L'incertitude provient de l'opérateur de sélection qui s'applique à la relation *client*. Juste après l'opérateur de sélection, un opérateur de contrôle et de décision est inséré dans le plan d'exécution.

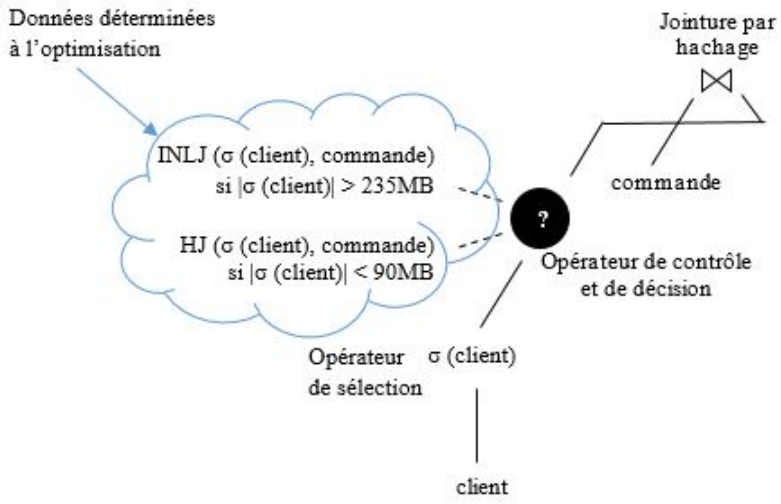


FIGURE 3.7 – Insertion d'un opérateur de contrôle et de décision

Lorsque les tuples sont produits par l'opérateur de sélection, ils peuvent être examinés par l'opérateur de contrôle. Ainsi, les statistiques requises peuvent être recueillies sans interrompre l'exécution de la requête. La cardinalité du résultat de l'opérateur de sélection peut être calculée en maintenant un compte courant du nombre de tuples qui transitent au-delà de l'opérateur de contrôle.

Cette approche de collecte de statistiques permet d'obtenir des statistiques actualisées sur les cardinalités avec une précision raisonnable. Cependant, elle présente une limite majeure, qui concerne le cas d'une exécution d'opérateurs en multiprogrammation. Si la collecte de statistiques se fait au milieu d'une telle exécution, aucun de ces opérateurs ne peut bénéficier de ces statistiques. En effet, les opérateurs s'exécutent concurremment avec la collecte de statistiques. Par conséquent, les statistiques ne seront prêtes que lorsque tous ces opérateurs auront achevé une partie importante de leur exécution. Une alternative à cela serait d'utiliser l'approche de matérialisation des résultats intermédiaires. Il faut noter qu'en cas d'exécution d'opérateurs dépendants, cette exécution sera interrompue aux points où les statistiques doivent être collectées. Ceci peut ralentir le temps d'exécution. Pour éviter ce problème, nous considérons les opérateurs devant s'exécuter en multiprogrammation comme un opérateur atomique. Ainsi, les statistiques sont collectées

après terminaison de ces opérateurs.
Dans notre méthode, nous déterminons les points dans un plan d'exécution où une collecte de statistiques est susceptible d'être bénéfique pour une exécution robuste de la requête. Comme décrit précédemment, cette responsabilité est déléguée à l'utilisateur qui spécifie à l'aide du paramètre seuil de risque le compromis qu'il souhaite entre la correction du plan pendant son exécution et le temps final de l'évaluation de la requête.
Une fois les statistiques collectées de cette manière lors de l'exécution de la requête, elles peuvent être exploitées pour obtenir de nouvelles estimations pour les tailles de résultats intermédiaires et les coûts d'exécution des opérateurs pour le reste de la requête. Dans la sous-section suivante, nous décrivons comment ces estimations améliorées peuvent être utilisées pour corriger une violation de la robustesse pendant l'exécution d'un plan.

3.3.2 Modification de plans d'exécution

Dans cette sous-section, nous décrivons comment des statistiques actualisées peuvent être utilisées pour détecter une violation de la robustesse et modifier un plan d'exécution. Lorsque des statistiques sont recueillies par un opérateur de contrôle et de décision inséré dans un plan d'exécution, cet opérateur vérifie si l'opérateur physique qui sera exécuté juste après dans le plan, reste robuste pour ces statistiques. Si ce n'est pas le cas, l'opérateur de contrôle et de décision est capable de prendre la décision de correction du plan d'une manière autonome. Pour cela, il s'appuie sur l'information déterminée au moment de l'optimisation concernant les plans robustes et leurs intervalles de robustesse associés. Dans ce document, nous supposons qu'une fois un opérateur ou un ensemble d'opérateurs physiques d'un plan commencent à s'exécuter, ils ne peuvent pas être modifiés. En d'autres termes, les statistiques améliorées ne peuvent être utilisées que pour les opérateurs physiques qui n'ont pas commencé à s'exécuter. En effet, nous pensons que rejeter la quantité du travail qui a déjà été effectué et recommencer à nouveau peut être bénéfique seulement si le travail à effectuer additionné au travail qui a été fait, est inférieur au travail qui aurait été effectué si l'exécution aurait continuer. Ça pourrait être le cas. Cependant,

nous pensons que ceci est risqué par rapport à la robustesse que nous cherchons à obtenir. Nous illustrons maintenant la modification d'un plan d'exécution à l'aide de la figure 3.8 ci-dessous. Cette figure montre un plan d'exécution P_{ac} associé à la requête de notre exemple 3.2.1. Supposons que le niveau d'incertitude associé au choix du premier sous-plan SP1 est faible. Ainsi, aucun opérateur de contrôle et de décision n'est inséré pour vérifier la robustesse de ce sous-plan à l'exécution. Supposons également que le niveau d'incertitude associé au choix du deuxième sous-plans SP2 est élevé en raison d'une forte incertitude concernant la taille de la relation temporaire résultante du sous-plan SP1. Un opérateur de contrôle et de décision est alors inséré dans le plan d'exécution de la requête juste avant le sous-plan SP2.

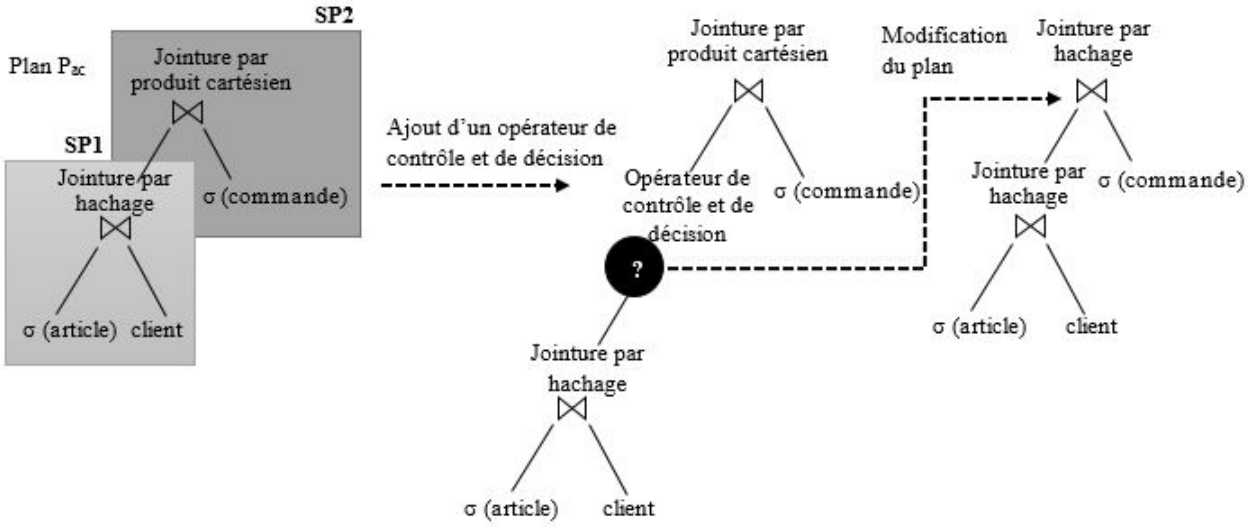


FIGURE 3.8 – Modification d'un plan d'exécution

Dans notre exemple, supposons que l'exécution de SP1 est terminée. La taille de la relation résultante du sous-plan SP1 peut être observée. Maintenant que l'incertitude est soulevée, l'opérateur de contrôle et de décision est capable de décider si une modification du reste du plan est nécessaire. Supposons également que SP2 n'a pas encore commencer l'exécution. Ainsi, SP2 peut être modifié. L'opérateur de contrôle et de décision produit

un nouveau plan pour SP2. La décision de modification est basée sur l'information déterminée à l'optimisation concernant les opérateurs physiques robustes associés à chaque opérateur logique de la requête et leurs intervalles de robustesse associés, et sur les statistiques collectées à l'exécution. La taille d'une relation temporaire observée à l'exécution peut appartenir à plusieurs intervalles de robustesse associés aux différents plans robustes énumérés. Dans ce cas, une question qui se pose concerne le choix du plan pour le reste de la requête. Pour résoudre ce problème, nous trions les plans d'exécution robustes déterminés à l'optimisation dans un ordre décroissant en fonction de leurs probabilités d'éviter une violation de la robustesse lors de l'exécution. Quand une correction est déclenchée, le premier plan d'exécution dont l'intervalle de robustesse inclut la valeur observée, est choisi pour le reste de la requête.

3.4 Conclusion

Dans ce chapitre, nous avons présenté une méthode d'optimisation hybride qui vise deux objectifs : la production de plans d'exécution robustes, et la correction d'une violation de la robustesse à l'exécution. Cette méthode comporte deux modules. Le premier module est une stratégie d'optimisation qui s'appuie sur des intervalles d'estimations calculés autour des tailles de relations opérandes, pour produire des plans d'exécution robustes. Dans un premier temps, les coûts de plans candidats à l'exécution d'une requête donnée sont comparés dans ces intervalles. Les plans offrant des performances robustes sont identifiés. Le plan qui minimise le risque d'une violation de la robustesse à l'exécution est ensuite choisi pour commencer l'exécution. Bien que ce plan réduise le risque, ce risque est existant et une violation de la robustesse peut avoir lieu. Dans l'objectif d'essayer d'assurer la robustesse jusqu'à la terminaison, nous avons proposé d'enrichir un plan d'exécution généré pour une requête, par des opérateurs de contrôle et de décision. L'insertion de ces opérateurs dans un plan est basé sur un seuil de risque défini par l'utilisateur. Les opérateurs de contrôle et de décision collectent des statistiques à l'exécution

et vérifient la robustesse du plan en cours. Si la robustesse est violée, ces opérateurs sont capables de prendre des décisions de correction du reste du plan sans avoir besoin de rappeler l'optimiseur. Ils se basent sur l'information déterminée à l'optimisation concernant les intervalles de robustesse d'algorithmes physiques pour chaque opérateur de la requête, et prennent la décision de correction en fonction des statistiques recueillies à l'exécution. Après avoir présenté notre méthode d'optimisation, nous évaluons ses performances dans le chapitre suivant.

Chapitre 4

Évaluation des performances

Sommaire

4.1 Introduction

Après avoir décrit notre méthode d'optimisation dans le chapitre précédent, nous présentons dans ce chapitre notre plateforme d'expérimentations et fournissons les résultats de l'évaluation des performances de notre méthode.
Pour mener nos expérimentations, nous avons décidé de développer notre propre simulateur. La simulation informatique [ZKP00] est une méthode d'expérimentation dans laquelle un système réel est remplacé par un modèle décrivant le système réel avec une précision suffisante. Le but d'une simulation est de visualiser le comportement d'un système réel dans des conditions fixées par des paramètres de simulation.
Nous rappelons que notre méthode est constituée de deux modules : un module de génération de plans d'exécution robustes, et un module de correction -si nécessaire- de plans pendant l'exécution. Nous comparons les performances obtenues par notre méthode, notée HyOpt, avec celles obtenues en utilisant les méthodes ReOpt [KD98] et Rio [BBD05]. ReOpt s'inscrit dans l'approche d'optimisation basée sur des estimations en points singuliers tandis que Rio s'inscrit dans l'approche d'optimisation basée sur des estimations en un ensemble de points. Le choix de ces méthodes comme références est motivé par la comparaison des méthodes dans chaque approche, que nous avons faite dans le chapitre état de l'art. Nous étudions le comportement de chacune de ces méthodes en variant l'erreur dans les estimations des tailles de relations opérandes dans les plans de requêtes fournies comme données de test. Nous varions aussi le niveau d'incertitude concernant les estimations utilisées à la phase d'optimisation. Nous considérons trois niveau d'incertitude : *faible*, *moyen*, et *élevé*. Le niveau d'incertitude est une entrée au calcul d'intervalles d'estimations autour des paramètres jugés incertains. Plus l'incertitude est élevée plus les intervalles sont larges. Ces intervalles sont utilisés par Rio et HyOpt pour engendrer des plans robustes. Quant à ReOpt, elle se base sur des valeurs singulières d'estimations pour trouver des plans estimés optimaux.
Avant de présenter les résultats de nos expérimentations, nous décrivons dans la section suivante notre modèle de simulation.

4.2 Description du simulateur

Pour effectuer nos expérimentations, nous avons utilisé un simulateur qui comporte un générateur de requêtes, notre méthode d'optimisation hybride, et une méta-base.
L'ensemble des requêtes générées est décrit dans chacune des expérimentations. La méthode simulée contient deux composants principaux : (1) un optimiseur de requêtes, et (2) un exécuteur de requêtes. La méta-base inclut des informations système décrivant l'environnement de l'évaluation des requêtes ainsi que des informations sur les données. L'environnement d'exécution dans notre simulateur est caractérisé par un processeur, un disque et une mémoire cache. La performance d'un processeur est définie en nombre d'instructions exécutées par seconde. La performance du système d'entrée/sortie disque est mesurée en nombre de mégaoctets lus et écrits par seconde. La mémoire pouvant être allouée à un opérateur dans un plan d'exécution est exprimée en mégaoctets. Les données manipulées sont des relations. Une relation est caractérisée par un ensemble d'attributs et le nombre de tuples qu'elle inclut. Chaque attribut est caractérisé par une taille exprimée en octets et un nombre de valeurs distinctes qu'il peut prendre.
Notre simulateur offre la possibilité de configurer plusieurs paramètres. Un exemple de configuration des principaux paramètres de simulation est donné dans Tableau 4.1 ci-dessous. Nous avons essayé de choisir les conditions de simulation les plus réalistes.

Paramètre	**Valeur**
Taille mémoire allouée à un opérateur	400 Mo
Performance CPU	100 000 MIPS
E/S disque	100 Mo/s
Latence moyenne	20 ms
Taille d'une page	4 Ko
Taille d'un tuple	1-3 Ko
Taille d'une relation	100 – 1000 Mo
Nombre d'attributs dans une relation	10
Taille d'un attribut	10-500 octets

TABLE 4.1 – Configuration de données d'expérimentation

Le reste de cette section est organisée comme suit : nous décrivons dans la sous-section 4.2.1 la simulation du module de génération de plans robustes. La sous-section 4.2.2 est dédiée à la description de la simulation du module de correction éventuelle de plans pendant l'exécution. Finalement, la sous-section 4.2.3 présente une description de la mise en œuvre logicielle de notre simulateur.

4.2.1 Simulation du module de génération de plans robustes

Ce module a pour objectif de générer pour chaque requête en entrée un ou des plans d'exécution robustes sur un ou des intervalles d'estimations utilisés. Notre simulateur utilise un générateur de requêtes pour fournir les requêtes d'entrée. Le générateur de requêtes peut générer plusieurs requêtes sous forme d'arbres d'opérateurs logiques. Notre simulateur permet de configurer le nombre d'opérateurs dans une requête ainsi que le nombre de requêtes à générer. Les requêtes générées peuvent être réutilisées dans plusieurs tests pour assurer la reproductibilité des résultats.
Le générateur de requêtes reçoit en entrée un ensemble de relations stockées dans la métabase, une valeur entière spécifiant le nombre de relations à joindre, et un tableau binaire qui définit les relations pouvant être jointes. Un exemple de ce tableau est donnée ci-dessous (Cf. Tableau 4.2). Dans ce tableau, la valeur 1 signifie que les deux relations se trouvant dans la même ligne et la même colonne que 1 peuvent être jointes. La valeur 0 signifie que les deux relations se trouvant dans la même ligne et la même colonne que 0 ne peuvent être jointes.

	R_1	R_2	R_3	...
R_1	0	1	0	...
R_2	1	0	1	...
R_3	0	1	0	...
...	...	...	...	...

TABLE 4.2 – Description de liens entre relations

Le générateur de requêtes choisit d'une manière aléatoire les relations à joindre pour

chaque requête à produire. Les requêtes sont produites sous forme d'arbres d'opérateurs logiques. Ces derniers sont ensuite utilisés comme entrée à notre stratégie de génération de plans robustes, et qui représente la partie centrale de la simulation. Cette stratégie détermine pour chaque requête un ou des plans d'exécution robustes face à l'incertitude des estimations disponibles à partir de la méta-base. Les données d'entrée pour cette stratégie sont l'ensemble d'arbres d'opérateurs produit par le générateur de requêtes, des méta-données décrivant les relations et les caractéristiques système (qui sont configurables à partir de fichiers d'entrée que nous discutons dans la sous-section 4.2.3), et un seuil de la sous-optimalité tolérée par l'utilisateur et qui détermine la condition de robustesse définit dans le chapitre 1 à la section 1.2.

Pour chaque arbre d'opérateurs en entrée, notre stratégie commence par sélectionner - d'une manière aléatoire- les relations opérandes qui seront considérées comme relations de base et celles qui seront considérées comme des relations temporaires. Les tailles des relations de base sont spécifiées par des estimations singulières à partir de la méta-base. Quant aux relations temporaires, un intervalle d'estimations est calculé autour de la taille estimée de chacune de ces relations. Nous supposons qu'un intervalle d'estimations pour un paramètre, inclut toutes les valeurs possibles de ce paramètre. En effet, nous supposons comme dans [SB99] qu'il est possible de déterminer les limites inférieure et supérieure de la valeur d'un paramètre. Cette méthode [SB99] étant complexe, nous avons décidé d'utiliser la technique proposée dans [BBD05] et décrite précédemment dans le chapitre 1 à la section 1.2, pour calculer des intervalles d'estimations. Nous avons fait en sorte que les valeurs d'un paramètre pouvant être observées à l'exécution soient à l'intérieur de l'intervalle d'estimations de ce paramètre.

Notre stratégie de génération de plans robuste transforme ensuite chaque arbre d'opérateurs logiques de façon à avoir un ordonnancement robuste des opérateurs. Les relations qui sont probablement les plus petites en termes de tailles seront les premières à être jointes, comme expliqué dans le chapitre précédent dans la sous-section 3.2.1. Les arbres d'opérateurs sont transformés de façon à tenir compte de l'incertitude des tailles de rela-

tions opérandes. L'étape d'après consiste à appliquer le principe suivant à chaque arbre d'opérateurs : des algorithmes physiques sont énumérés pour chaque opérateur logique et les coûts d'exécution de ces algorithmes sont comparés pour les valeurs/intervalles d'estimations des relations opérandes. L'objectif est d'identifier les algorithmes physiques robustes associés à chaque opérateur. Puis nous calculons les risques d'une violation de la robustesse à l'exécution, que présentent les algorithmes physiques robustes associés à chaque opérateur. Un algorithme physique qui minimise ce risque est sélectionné pour chaque opérateur. Un plan initial est alors établi pour commencer l'exécution de la requête. L'ensemble de plans d'exécution de requêtes fourni par ce module est ensuite transmis au module de correction de plans d'exécution pour préparer l'exécution. La simulation de ce module est décrite dans la sous-section suivante.

4.2.2 Simulation du module de correction de plans d'exécution

Ce module a pour objectif de corriger une violation de la robustesse constatée à l'exécution. Les données d'entrée pour ce module sont les plans d'exécution robustes établis pour commencer l'exécution des requêtes générées, et un seuil de risque fixé par l'utilisateur. Dans un premier temps, chaque plan en entrée est parcouru -opérateur par opérateur- et le risque que présente chaque opérateur physique de résulter en une violation de la robustesse est comparé au seuil de risque toléré. Si ce risque est inférieur au seuil de risque, le parcours du plan continue normalement. Cependant, si ce risque dépasse le seuil de risque, un opérateur dit de contrôle et de décision est inséré dans l'arbre d'opérateurs physiques représentant le plan.

Dans notre simulateur, une requête est représentée sous forme d'un arbre d'opérateurs logiques (e.g., lecture, jointure). Lorsqu'on spécifie un ordre pour les relations opérandes ainsi qu'un algorithme physique (e.g., jointure par hachage) à un opérateur logique dans cet arbre, nous l'appelons désormais un opérateur physique. Un plan d'exécution est alors un arbre d'opérateurs physiques ordonnancés.

Un opérateur de contrôle et de décision est inséré de façon à qu'il soit exécuté avant l'opérateur physique en question. Un opérateur de contrôle et de décision dans notre simulation est caractérisé par une paire d'opérations (lecture, décision). L'opération de lecture consiste à lire les tuples reçus à partir d'un opérateur physique, et comparer la cardinalité observée à celle pour laquelle l'opérateur physique qui suit est présumé robuste. Si la cardinalité observée est dans l'intervalle de robustesse de l'opérateur physique qui suit, aucune modification n'est apportée et l'exécution continue. Dans le cas contraire, l'information déterminée au moment de l'optimisation concernant les intervalles de robustesse des algorithmes physiques énumérés pour l'opérateur en question, est consultée. L'objectif est d'apporter la correction adéquate au plan d'exécution. La correction d'un plan porte sur le reste des opérateurs physiques à exécuter.
En sortie de cette simulation, nous avons les temps de réponse des requêtes et d'autres métriques (e.g., variances des temps de réponse) permettant d'évaluer la qualité de l'évaluation des requêtes.
La simulation de l'exécution des plans s'effectue dans un échelle de temps propre au simulateur. Ainsi, les résultats de la simulation ne dépendent pas de la performance de la machine utilisée. Celle-ci présente un impact seulement sur la durée de la simulation.

4.2.3 Mise en œuvre

Pour implémenter notre simulateur, nous avons décidé d'utiliser la plateforme Java. Pour le développement, nous avons utilisé l'environnement de développement (IDE) Eclipse. Notre simulateur consiste en une application console qui reçoit les données d'entrée à partir de fichiers de type CSV. Un fichier CSV est un fichier tableur, contenant des données sur chaque ligne séparées par un caractère de séparation. Voici un exemple de fichier utilisé comme entrée à notre application :

Relation_ID	Nbr_Tuples
1	100
2	145
3	1000
4	75
...	...

TABLE 4.3 – Fichier CSV contenant les propriétés des relations stockées

Attrib_ID	Relation_ID	Taille (en octets)	Cardinalité
1	1	100	0,56
2	1	145	0,24
3	1	103	0,92
4	1	75	0,42
5	1	10	0,89
...	...	...	...

TABLE 4.4 – Fichier CSV contenant les attributs des relations

Le tableau 4.3 décrit le nombre de tuples par relation. Le tableau 4.4 liste et décrit les attributs des relations dans la méta-base. La méta-base est formée par l'ensemble de fichiers CSV utilisé comme entrée à notre application. Ces fichiers comportent les propriétés des sources de données (i.e., relations) et des ressources de calcul (i.e., CPU, mémoire, bande passante).

Dans l'évaluation des performances, nous avons étudié les requêtes de type mono et multi-jointures. Nous avons simulé principalement l'opération de jointure, dont le coût d'exécution est une fonction des coûts d'utilisation CPU et des E/S disque :

$$\text{Coût}_{jointure} = \text{Coût}_{CPU} + \text{Coût}_{E/S}$$

avec

$$\text{Coût}_{CPU} = \text{N}_{comp} \times \text{T}_{comp} \qquad \text{et} \qquad \text{Coût}_{E/S} = \frac{N_{E/S} \times S_{tuple}}{D_{E/S}} + \text{latence}$$

où :

— N_{comp} : nombre de comparaisons entre les tuples des relations à joindre.
— T_{comp} : temps nécessaire pour effectuer une comparaison entre deux tuples.
— $N_{E/S}$: le nombre de tuples transféré par le sous-système d'entrée/sortie disque.
— S_{tuple} : la taille d'un tuple en octets.
— $D_{E/S}$: la bande passante disponible du sous-système d'entrée/sortie, qui est mesurée en octets par seconde.
— latence : temps moyen d'accès à une page sur le disque.

Nous calculons la taille, notée S_{rel}, d'une relation *rel* résultante d'une jointure entre deux relations rel_1 et rel_2 ($rel_1.x = rel_2.y$) comme suit :

$$S_{rel} = N_{result} \times S_{tuple}$$

où

$$N_{result} = \frac{N_{rel1} \times N_{rel2}}{max(d_1, d_2)}$$

tel que :

— N_{result} : le nombre de tuples dans la relation *rel*.
— N_{rel1} : le nombre de tuples dans la relation rel_1.
— N_{rel2} : le nombre de tuples dans la relation rel_2.
— d_1 : le nombre de valeurs distinctes de l'attribut de jointure (x) dans rel_1.
— d_2 : le nombre de valeurs distinctes de l'attribut de jointure (y) dans rel_2.

Après avoir décrit notre modèle de simulation, nous présentons dans les sections suivantes les résultats de l'évaluation des performances de notre méthode d'optimisation hybride. Nous commençons tout d'abord par l'évaluation des requêtes de type mono-jointures.

4.3 Évaluation des performances de requêtes mono-jointures

Dans cette section, nous nous intéressons à l'évaluation des performances de notre méthode d'optimisation pour des requêtes de type mono-jointures. Nous étudions d'abord le cas où la taille de seulement une des deux relations opérandes d'une jointure est sujette à une erreur d'estimation. Ensuite, nous étendons notre analyse au cas où les tailles des deux relations opérandes d'une jointure sont considérées sujettes à des erreurs d'estimations. Tout au long des expérimentations, nous considérons que le seuil λ de la sous-optimalité acceptée pour un plan par rapport au plan optimal est égal à 20%.

4.3.1 Une relation sujette à une erreur d'estimation

Pour les expérimentations dans ce cadre, nous utilisons un ensemble S composé de 100 requêtes. Chaque requête contient une jointure et vérifie les deux conditions ci-dessous :

- une des deux relations est une relation de base, notée R1, et
- l'autre relation est une relation temporaire, notée T, résultante d'une opération de sélection sur une relation de base R2

Notre optimiseur accède aux informations disponibles dans la méta-base (i.e., fichiers d'entrée de type CSV) sur les tailles des relations de base. Quant aux relations temporaires, les informations disponibles sur leurs tailles sont considérées non fiables. Ceci peut être dû à une mauvaise répartition inhérente aux données ou à d'éventuelles corrélations entre les attributs de sélection sur les relations de base. La non fiabilité de l'estimation disponible de la taille d'une relation temporaire est modélisée par un intervalle d'estimations. Pour mener nos expérimentations, nous avons utilisé la technique proposée dans [BBD05] pour calculer un tel intervalle. Nous considérons trois niveaux d'incertitude : faible, moyen, et élevé. Les intervalles calculés sont ensuite utilisés par Rio et HyOpt pour engendrer des plans d'exécution robustes pour les requêtes en entrée. ReOpt utilise des estimations en valeurs singulières pour choisir, pour chaque requête, un plan estimé optimal.

Dans le reste de cette sous-section, nous comparons les performances de ReOpt, Rio, et HyOpt sur la base de deux paramètres : le temps d'optimisation qui réfère à la durée du processus d'optimisation de requêtes, et le temps de réponse qui est la somme des temps d'optimisation et d'exécution.

4.3.1.1 Impact d'erreur d'estimation sur le temps d'optimisation

Le temps d'optimisation de HyOpt réfère au temps nécessaire pour le module de génération de plans d'exécution robustes. Le temps d'optimisation de Rio est le temps nécessaire pour générer -pour une requête- un plan robuste, un ensemble de plans interchangables, ou un plan optimal comme dans [MRS+04]. Finalement, le temps d'optimisation de ReOpt est le temps requis pour engendrer un plan optimal pour une requête donnée.

La comparaison des temps d'optimisation des méthodes permet de visualiser la complexité de chaque méthode à la phase d'optimisation. La figure 4.1 ci-dessous illustre les temps d'optimisation médians de l'ensemble S des requêtes mono-jointures, pour HyOpt, Rio, et ReOpt. Nous avons décidé d'utiliser la médiane plutôt que la moyenne car des valeurs extrêmes des temps d'optimisation peuvent augmenter ou diminuer fortement la moyenne, donnant ainsi une vision qui n'est pas représentative de la grande majorité des valeurs.

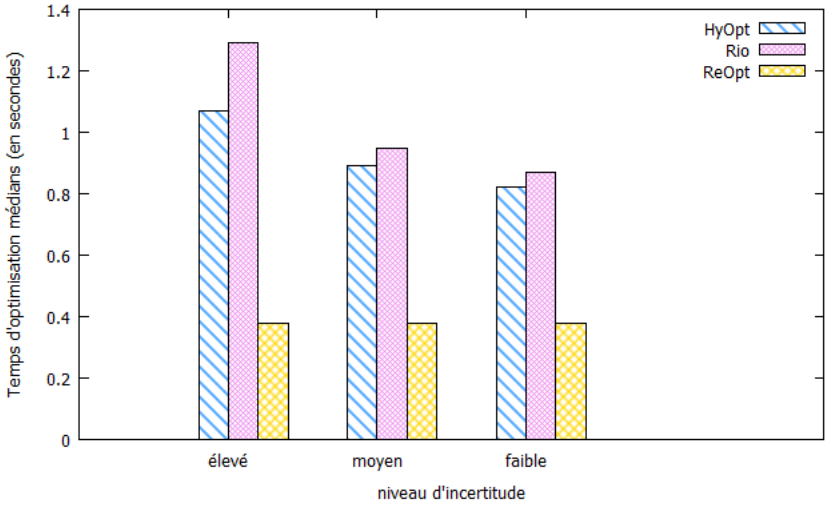


FIGURE 4.1 – Temps d'optimisation médians des méthodes (1er cas)

Comme nous pouvons le constater dans la figure 4.1, HyOpt fournit des temps d'optimisation beaucoup plus élevés comparé à ReOpt. Ceci s'explique par le travail supplémentaire que nécessite HyOpt pendant la phase d'optimisation (e.g., calcul d'intervalles/surfaces de robustesse) par rapport au processus d'optimisation de requêtes dans ReOpt. L'écart entre les temps d'optimisation par HyOpt et Rio est cependant moins important. HyOpt et Rio présentent des temps d'optimisation proches du fait de la similarité de leurs comportements lors de l'optimisation d'une requête. En effet, Rio nécessite également un travail supplémentaire à l'optimisation (e.g., calcul d'intervalles d'estimations, vérification de la condition de robustesse, recherche de plans interchangeables) afin de générer des plans adéquats.

Nous observons aussi dans cette figure que plus l'incertitude augmente plus les temps d'optimisation engendrés par HyOpt et Rio augmentent. Pour HyOpt, ceci est causé par un sur-coût relatif aux parcours d'intervalles plus grands pour identifier des plans robustes sur ces intervalles ou sur des sous-intervalles de ces intervalles. Pour Rio, quelque soit la largeur d'un intervalle utilisé, la condition de robustesse d'un plan est vérifiée à trois points dans cet intervalle, i.e., les bornes inférieure et supérieure de l'intervalle et l'estimation en une valeur singulière du paramètre en question. L'augmentation des temps d'optimisation par Rio est due à la complexité de trouver un plan robuste sur l'intégralité d'un intervalle quand ce dernier est grand, et le travail supplémentaire ainsi déclenché (e.g., recherche de plans interchangeables). La génération de plans d'exécution par ReOpt est indépendante du niveau d'incertitude. ReOpt utilise des estimations en valeurs singulières pour choisir des plans estimés optimaux. Pour cette raison, les temps d'optimisation engendrés par ReOpt sont les mêmes quelque soit le niveau d'incertitude.

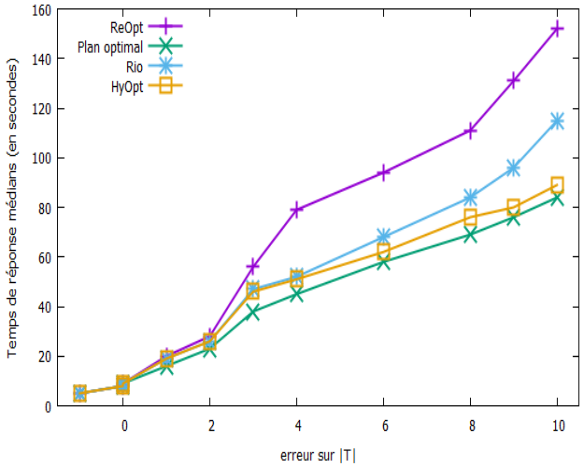
Le temps d'optimisation est inclus dans le temps de réponse. Nous vérifions l'impact des coûts d'optimisation sur les performances finales des méthodes. Nous présentons ci-après les résultats de l'évaluation des temps de réponse de requêtes pour HyOpt, Rio, et ReOpt.

4.3.1.2 Impact d'erreur d'estimation sur le temps de réponse

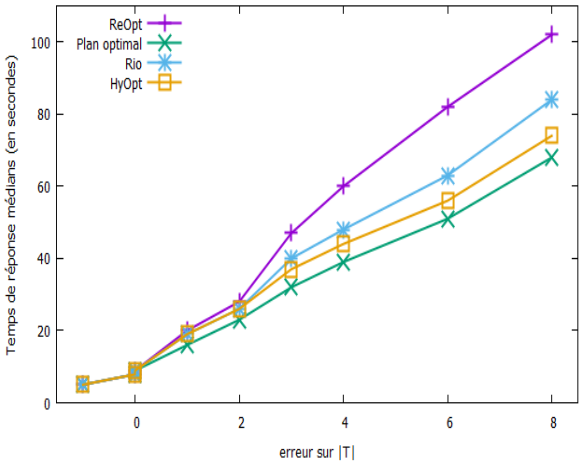
Nous cherchons à travers cette expérimentation à évaluer la qualité des plans générés par HyOpt, Rio et ReOpt. Nous nous intéressons à la capacité de ces méthodes de fournir des plans qui minimisent le besoin de corrections lors de l'exécution. Pour cela, nous simulons un comportement statique de ces méthodes. Nous ne simulons pas dans cette expérimentation l'interaction entre l'exécution des plans générés et l'environnent d'exécution (e.g., insertion d'opérateurs de contrôle et de décision, collecte de statistiques). L'apport du module de correction de plans d'exécution sera discuté dans la section 4.4.
Le temps de réponse d'une requête dépend de la qualité du plan d'exécution associé à cette requête. Nous évaluons l'impact d'erreur d'estimation sur les temps de réponse de requêtes pour chaque méthode. Nous rappelons que le temps de réponse est constitué du temps d'optimisation et du temps d'exécution. La figure 4.2 ci-après montre la variation des temps de réponse pour l'ensemble des requêtes dans S, pour ReOpt, Rio et HyOpt. Cette variation est déterminée en fonction de l'erreur sur l'estimation de la taille de la relation temporaire de chaque requête. Cette figure montre également le temps de réponse optimal. Le temps de réponse par une méthode à un point de l'axe des abscisses -représentant la valeur de l'erreur- est calculé comme la médiane des temps de réponse par cette méthode pour les 100 requêtes dans S. L'erreur d'estimation représentée sur l'axe des abscisses est calculée comme suit [BBD05] :

$$erreur = \frac{|T|_{observé}}{|T|_{estimé}} - 1 \tag{4.1}$$

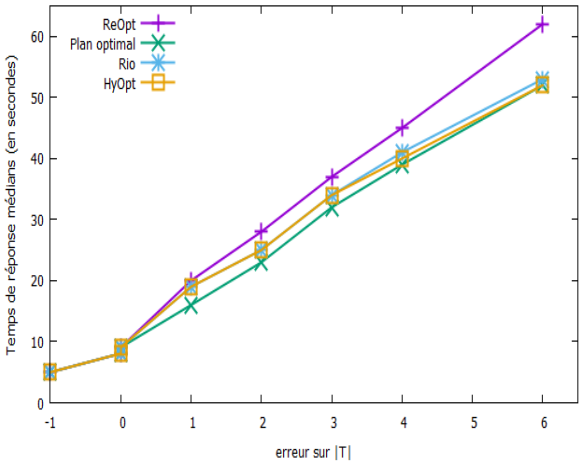
$|T|_{observé}$ définit la valeur à l'exécution de la taille de la relation temporaire T. $|T|_{estimé}$ définit la taille estimée de T. Une erreur positive indique une sous-estimation tandis qu'une erreur négative indique une sur-estimation de $|T|_{observé}$ par rapport à $|T|_{estimé}$.



(a) incertitude élevée



(b) incertitude moyenne



(c) incertitude faible

FIGURE 4.2 – Variation des temps de réponse médians en fonction de l'erreur sur |T|

Nous pouvons conclure à partir de la figure 4.2 que le sur-coût d'optimisation que présente HyOpt par rapport à ReOpt est compensé à l'exécution. Nous observons aussi dans cette figure que les performances de ReOpt restent acceptables lorsque l'erreur sur |T| est faible. Cependant, quand l'erreur devient importante (> 2) on observe une augmentation significative des temps de réponse des requêtes pour ReOpt. En effet, ReOpt choisit un plan d'exécution dont le temps d'exécution est estimé optimal pour la valeur estimée de la taille de la relation temporaire, notée T pour chaque requête. Les plans choisis par ReOpt sont alors très sensibles aux erreurs d'estimations. Rio et HyOpt choisissent, chacune, des

plans d'exécution capables d'engendrer des performances robustes pour différentes valeurs de |T|. La figure 4.2 montre que Rio présente des temps de réponse inférieurs à ceux de ReOpt. Cependant, ces temps sont supérieurs à ceux par HyOpt. L'écart entre les temps de réponse fournis par Rio comparé à HyOpt s'explique par le fait que plus l'incertitude concernant l'estimation de |T| est élevée plus l'intervalle d'estimations est grand. Ainsi, il devient difficile de trouver un plan robuste sur l'intégralité de l'intervalle. Dans ce cas, Rio cherche à produire des plans interchangeables. Si Rio échoue, il se comporte désormais comme la méthode dans [MRS+04]. Rio se base sur la valeur estimée de |T| pour choisir un plan d'exécution estimé optimal. Ce plan se révèle souvent être sous-optimal en cas d'erreurs d'estimations. Contrairement à Rio, HyOpt conserve un comportement basé sur des estimations en un ensemble de points. HyOpt parcourt les intervalles d'estimations pour trouver des plans robustes. Utiliser un intervalle d'estimations plutôt qu'une valeur singulière pour l'estimation de |T| permet à HyOpt de produire plus souvent des plans dont les temps d'exécution sont plus stables comparé à Rio. Nous remarquons également dans la figure 4.2 que plus l'incertitude est faible plus les temps de réponse médians par Rio et HyOpt sont proches. En effet, plus un intervalle d'estimations est réduit plus il devient facile pour Rio et HyOpt de trouver un plan robuste sur cet intervalle. Rio a ainsi un comportement similaire à HyOpt et des performances proches.

Après avoir évalué le cas où la taille d'une relation est sujette à une erreur d'estimation, nous présentons dans la sous-section suivante l'étude du cas où les tailles des deux relations opérandes d'une requête mono-jointure sont sujettes à des erreurs d'estimations.

4.3.2 Deux relations sujettes à des erreurs d'estimations

Pour cette évaluation, nous utilisons un ensemble S composé de 100 requêtes, tel que chaque requête comporte une jointure et vérifie les conditions suivantes :

- une des deux relations est une relation temporaire, notée T1, résultante d'une opération de sélection sur une relation de base R1, et

- l'autre relation est une relation temporaire, notée T2, résultante d'une opération de sélection sur une relation de base R2

Comme Rio exige qu'au moins une des relations opérandes soit une relation de base, nous comparons HyOpt seulement à ReOpt. Nous évaluations ReOpt et HyOpt en se basant sur les mêmes données d'expérimentations et les mêmes métriques que précédemment.

4.3.2.1 Impact d'erreur d'estimation sur le temps d'optimisation

La figure 4.3 ci-dessous montre les temps d'optimisation médians engendrés par les méthodes HyOpt et ReOpt.

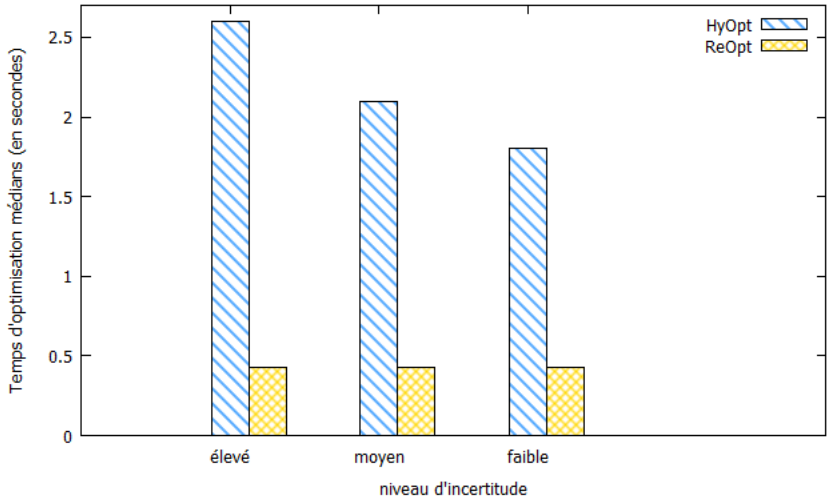
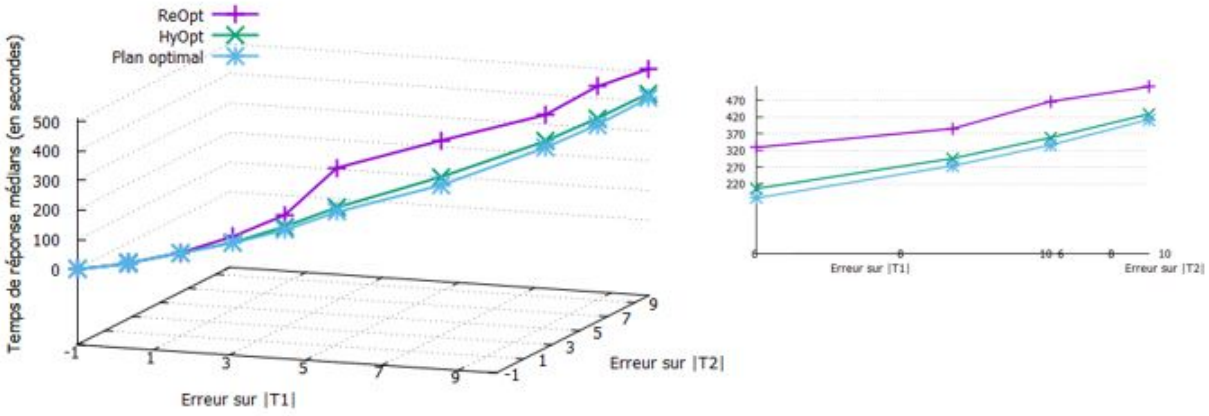


FIGURE 4.3 – Temps d'optimisation médians des méthodes (2ème cas)

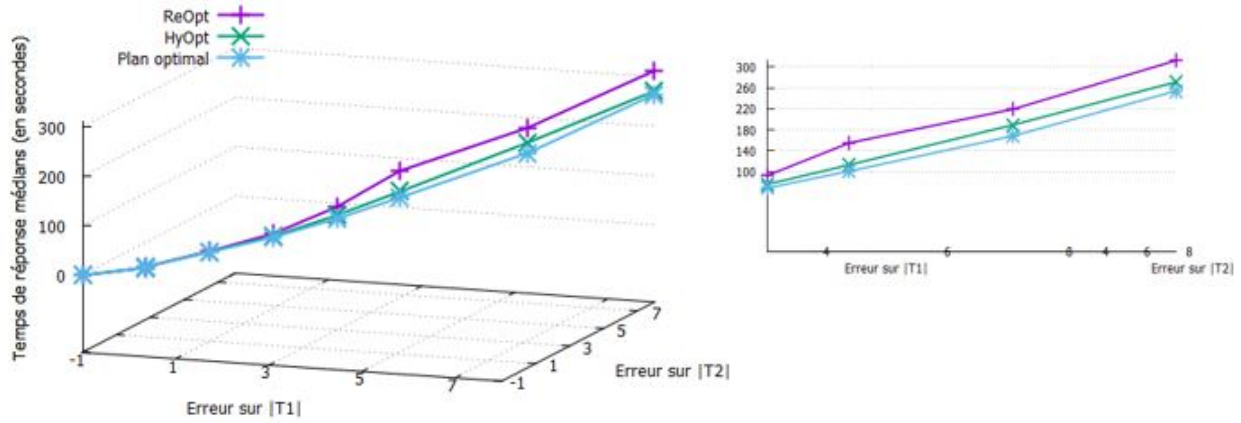
Comme nous pouvons le constater dans cette figure, ReOpt présente toujours de meilleurs coûts d'optimisation comparé à HyOpt. Les temps d'optimisation engendrés par HyOpt sont plus élevés comparé au cas précédent (à la sous-section 4.3.1.1). Ceci est du au sur-coût induit par le calcul de surfaces plutôt que d'intervalles de robustesse pour les algorithmes physiques candidats à l'exécution de chaque requête mono-jointure. L'augmentation des temps d'optimisation pour HyOpt avec l'augmentation de l'incertitude est causée -comme le cas précédent- par un sur-coût relatif au parcours d'intervalles plus grands pour identifier des plans robustes sur des surfaces formées par ces intervalles.

4.3.2.2 Impact d'erreur d'estimation sur le temps de réponse

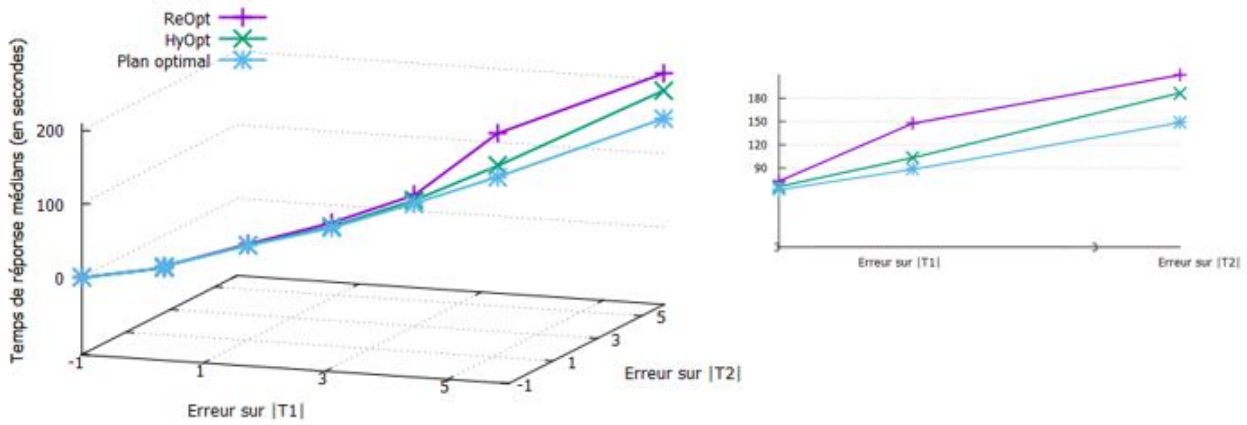
La figure 4.4 ci-dessous montre la variation des temps de réponse médians pour les méthodes HyOpt et ReOpt en fonction des erreurs sur les estimations des tailles des relations opérandes T1 et T2. Comme dans la sous-section 4.3.1, nous simulons un comportement statique de ces méthodes. L'objectif est d'étudier la qualité des plans générés et leurs capacités de minimiser le besoin de corrections.



(a) incertitude élevée



(b) incertitude moyenne



(c) incertitude faible

FIGURE 4.4 – Variation des temps de réponse en fonction des erreurs sur |T1| et |T2|

La figure 4.4 confirme les résultats obtenus précédemment. HyOpt fournit des temps de réponse plus stables en présence d'erreurs d'estimations, comparé à ReOpt.

Dans ce qui précède, nous avons évalué les performances de notre méthode d'optimisation hybride en étudiant sa complexité à l'optimisation et la qualité des plans d'exécution générés, par rapport aux méthodes Rio et ReOpt. Pour cela, nous avons supposé un comportement statique de ces méthodes et avons utilisé des requêtes de type mono-jointures. Ce type de requêtes nous donne un moyen de visualiser l'apport en stabilité d'exécution qu'offre HyOpt face à des erreurs d'estimations. Pour appuyer ces constatations, nous présentons dans la section suivante l'évaluation des performances de notre méthode en utilisant des requêtes multi-jointures. L'évaluation des performances de ce type de requêtes invoque le module de corrections de plans d'exécution.

4.4 Évaluation des performances de requêtes multi-jointures

Les expérimentations présentées dans cette section utilisent des requêtes de type multi-jointures et supposent un comportement réactif des méthodes faces à des erreurs d'estimations constatées lors de l'exécution d'un plan. Nous étudions dans un premier temps l'impact de la valeur du seuil de risque défini par l'utilisateur, sur les temps d'exécution des plans de requêtes. Puis, nous étudions l'impact d'erreurs d'estimations et du niveau d'incertitude sur les temps de réponse. Nous terminons cette section par une étude de la variance des temps de réponse offerts par HyOpt, Rio et ReOpt.

4.4.1 Impact du seuil de risque sur le temps d'exécution

Cette expérimentation consiste à étudier l'impact du seuil de risque, noté δ et défini par l'utilisateur, sur les temps d'exécution de plans par HyOpt. Pour cela, nous considérons différentes valeurs de δ, et déterminons pour chaque valeur la variation des temps d'exécution médians pour les plans d'exécution en entrée. La variation des temps d'exécution est calculée en fonction de l'erreur sur les tailles de relations opérandes. Ces relations sont celles sélectionnées par notre stratégie de génération de plans robustes d'une manière aléatoire, comme expliqué dans la sous-section 4.2.1. Nous appliquons la même valeur de l'erreur à chaque fois. L'erreur varie entre -1 et 7 comme illustré dans la figure 4.5 ci-dessous. Les données d'entrée pour cette expérimentation sont les plans générés par le module de génération de plans robustes, puis enrichis par des opérateurs de contrôle et de décision, et les diverses valeurs de δ.

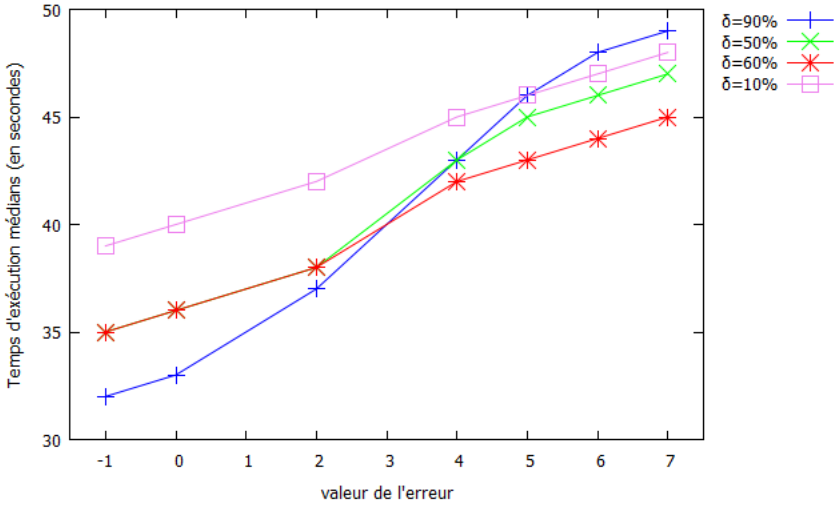


FIGURE 4.5 – Impact du seuil de risque sur les temps d'exécution médians

Comme nous pouvons le constater dans la figure 4.5, des valeurs extrêmes du seuil de risque δ induisent un coût élevé : lorsque δ est important (i.e., 90%), peu d'opérateurs de contrôle et de décision sont insérés dans un plan d'exécution afin de vérifier la robustesse de ce dernier pendant l'exécution. Des violations de la robustesse peuvent ainsi ne pas être détectées ni corrigées. Lorsque δ est faible (i.e., δ=10%), de nombreux opérateurs de contrôle et de décision sont insérés dans un plan d'exécution. La collecte de statistiques et la comparaison des valeurs observées avec les valeurs estimées sont à l'origine de ce sur-coût. Ceci explique les coûts d'exécution élevés pour ces valeurs de δ.
La figure 4.5 nous donne un moyen de visualiser que des valeurs modérées de δ permettent d'obtenir de bonnes performances, comparé aux valeurs extrêmement élevées ou faibles. Cette figure montre que les meilleurs coûts d'exécution se produisent lorsque δ=60%. Pour cette raison, nous considérons dans le reste des expérimentations, que δ est égale à 60%.

4.4.2 Impact d'erreur d'estimation sur le temps de réponse

Nous cherchons à travers cette expérimentation à comparer les temps de réponse engendrés par HyOpt, Rio et ReOpt pour des requêtes multi-jointures. L'objectif est d'évaluer les performances finales de notre méthode HyOpt en présence de violations de la robus-

tesse à l'exécution, toujours par rapport aux méthodes de références ReOpt et Rio.
Le temps de réponse d'une requête par HyOpt inclut le temps d'optimisation durant lequel un ou des plans robustes sont produits puis enrichis par des opérateurs de contrôle et de décision, et le temps d'exécution durant lequel une correction du reste du plan en cours peut avoir lieu. Le temps de réponse d'une requête par Rio comprend le temps d'optimisation durant lequel un plan robuste, ou un ensemble de plans interchangeable ou encore un plan optimal est généré, et le temps d'exécution du plan produit. Finalement, le temps de réponse d'une requête par ReOpt comprend le temps d'optimisation durant lequel un plan optimal est choisi puis enrichi par des opérateurs de collecte de statistiques, et le temps d'exécution durant lequel une ré-optimisation du reste du plan peut être déclenchée.
Dans les plans d'exécution associés aux requêtes multi-jointures utilisées, les relations opérandes d'un opérateur de jointure peuvent être : deux relations de base, deux relations temporaires, ou une relation de base et une relation temporaire. Pour pouvoir comparer HyOpt à Rio, il faut qu'au moins une des relations opérandes de chaque opérateur dans un plan d'exécution soit une relation de base. En effet, Rio se base sur cette hypothèse. Ainsi, dans cette expérimentation nous faisons en sorte que chaque opérateur dans un plan d'exécution possède en entrée une relation de base. Nous varions l'erreur sur les estimations des tailles des relations temporaires dans chaque plan d'exécution et calculons les temps de réponse. La figure 4.6 ci-dessous montre la variation des temps de réponse engendrés par HyOpt et Rio pour l'ensemble des requêtes en entrée. Cette figure montre également le temps de réponse optimal.

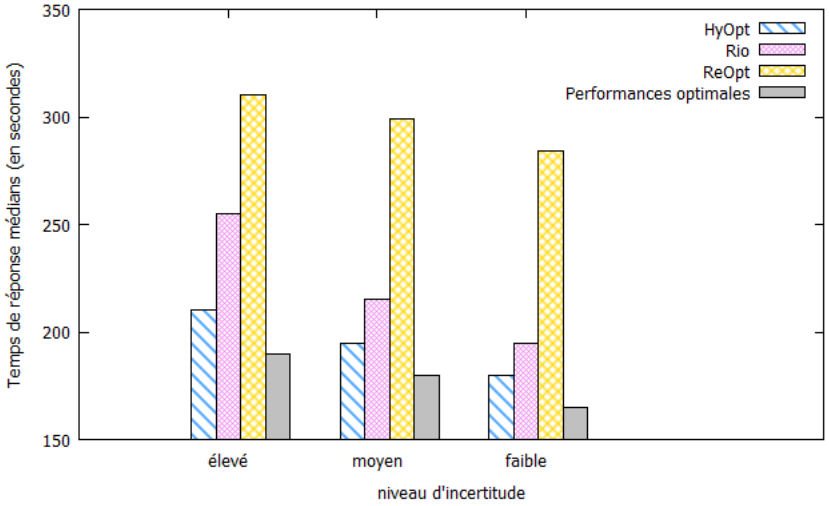


FIGURE 4.6 – Variation des temps de réponse médians des méthodes (1er cas)

La figure 4.6 appuie nos constatations à partir des expérimentations précédentes. Nous observons dans cette figure que ReOpt fournit des performances sous-optimales aux différents niveaux d'incertitude. Le sur-coût que présente ReOpt est du aux nombreuses ré-optimisations requises pour corriger les plans d'exécution. Le fait que ReOpt choisisse des plans d'exécution en ignorant la possibilité d'erreurs dans les estimations utilisées, augmente le risque de sous-optimalités des plans et ainsi de ré-optimisations à l'exécution. Bien que le comportement de ReOpt est indépendant du niveau d'incertitude, nous constatons que les performances de ReOpt ne sont pas les mêmes aux trois niveaux d'incertitude. En effet, un intervalle d'estimations est calculé en fonction de la valeur estimé d'un paramètre, jugée sujette à une erreur. Un niveau faible induit un intervalle restreint et des valeurs -à l'exécution- pas très éloignées de la valeur estimée. Cependant, un niveau plus important (moyen ou élevé) implique un intervalle plus grand et de ce fait des valeurs -potentiellement- plus grandes à l'exécution. Les tailles des relations sont des paramètres dans les fonctions de calculs des coûts. Ainsi plus les tailles sont importantes plus les temps d'exécution augmentent.

Comme HyOpt et Rio cherchent des plans robustes sur des intervalles d'estimations des tailles des relations temporaires, ces méthodes présentent des performances plus stables

face à des erreurs d'estimations. Nous visualisons dans cette figure que l'écart entre HyOpt et Rio augmente quand le niveau d'incertitude augmente. Cela confirme nos conclusions à partir des expérimentations précédentes.

Dans l'expérimentation ci-dessus, nous avons comparé HyOpt à Rio. Pour cela, nous avons supposé qu'un moins une des relations opérandes de chaque opérateur dans un plan d'exécution, est une relation de base. Nous supposons maintenant que les relations opérandes d'un opérateur dans un plan d'exécution peuvent être deux relations temporaires. Nous présentons dans la figure 4.7 ci-après la variation des temps de réponse des méthodes HyOpt et ReOpt.

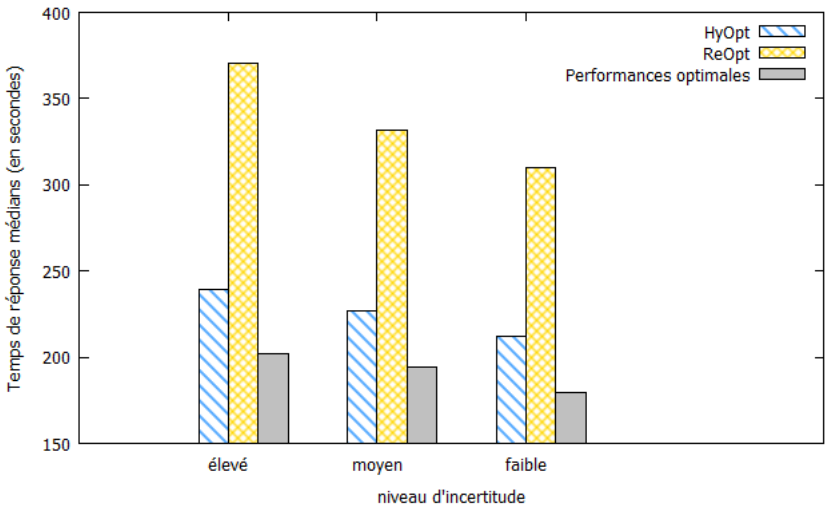
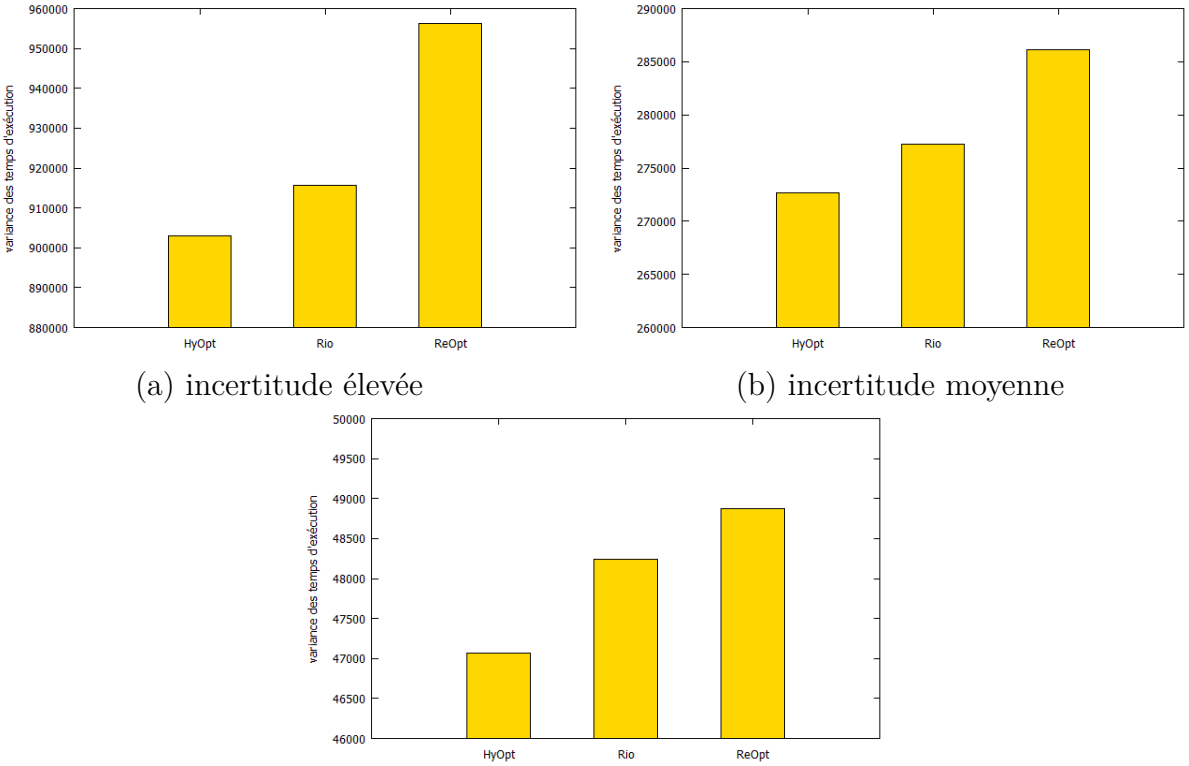


FIGURE 4.7 – Variation des temps de réponse médians des méthodes (2ème cas)

Comme le montre cette figure, des erreurs d'estimations sur les tailles de deux relations opérandes ont un effet encore plus dégradant sur les performances engendrées par ReOpt. HyOpt fournit des performances plus stables comparé à ReOpt puisque des intervalles ou des surfaces de robustesse sont considérées pour les plans d'exécution, plutôt que des estimations en valeurs singulières.

4.4.3 Impact d'erreur d'estimation sur la consistance

La consistance d'une méthode réfère à sa capacité de faire face à des erreurs d'estimations et/ou à des changements dans les conditions d'exécution par rapport aux conditions prévues lors de l'optimisation. Une méthode a une consistance élevée si ses performances ne se dégradent pas d'une manière significative en présence d'erreurs d'estimations. Afin de mesurer la consistance des méthodes HyOpt, Rio, et ReOpt, nous calculons la variance des performances de chaque méthode. La consistance est inversement proportionnelle à la variance. Pour cette expérimentation, nous utilisons les mêmes données de test (e.g., plans d'exécution) que précédemment. Les résultats sont illustrés dans la figure 4.8 ci-dessous.



(a) incertitude élevée

(b) incertitude moyenne

(c) incertitude faible

FIGURE 4.8 – Variance des méthodes

Comme nous pouvons le constater sur la figure 4.8, les variances de HyOpt et Rio sont proches lorsque l'incertitude est faible. Cependant, plus l'incertitude est élevée plus l'écart entre les variances augmente. Une variance élevée signifie une dispersion importante des

temps d'exécution. Ainsi, la figure 4.8 montre que HyOpt offre des performances plus stables comparé à Rio et ReOpt. Ceci confirme les résultats que nous avons obtenus précédemment.

4.5 Conclusion

Ce chapitre a été consacré à l'évaluation des performances de notre méthode d'optimisation hybride. Nous avons présenté dans un premier temps le simulateur développé, puis les résultats de l'évaluation des performances de notre méthode pour des requêtes mono-jointures et multi-jointures. Notre simulateur permet de configurer les paramètres de simulation et les données d'entrée, et de visualiser le comportement de notre méthode. Les résultats obtenues mettent en évidence une augmentation des temps d'optimisation qu'engendre notre méthode par rapport à une optimisation basée sur des estimations en points singuliers. Cependant, notre méthode présente des temps de réponse moins importants comparé à la méthode de référence ReOpt [KD98] qui représente l'approche d'optimisation basée sur des estimations en points singuliers. Ceci est en raison de la stabilité des temps d'exécution que fournit notre méthode et qui compense le sur-coût d'optimisation. Le sur-coût d'optimisation qu'engendre notre méthode est au profit d'une exécution stable et performante des plans d'exécution. Ceci présente un avantage majeure lorsque l'incertitude concernant les estimations utilisées à l'optimisation est importante et que les caractéristiques de l'exécution sont peu prévisibles.
Les résultats permettent également de constater que les performances engendrées par notre méthode sont proches de celles offertes par la méthode Rio [BBD05] qui référence l'approche d'optimisation basée sur un ensemble de points. Comme notre méthode, plusieurs méthodes dans cette approche utilisent des intervalles d'estimations pour produire des plans d'exécution robustes. Lorsque les intervalles utilisés sont restreints, Rio et HyOpt fournissent des performances proches. Cependant, quand ces intervalles sont grands, notre méthode présente de meilleures performances. Considérer plusieurs plans robustes sur dif-

férents sous-intervalles d'un intervalle d'estimations initial puis choisir le plan le plus probable d'être robuste, augmenter les chances d'obtenir des performances stables. De plus, le fait que notre méthode se base sur des décisions déterminées au moment de l'optimisation concernant les corrections d'une violation de la robustesse à l'exécution, permet de réduire le sur-coût de correction en évitant le coût d'une ré-optimisation.

Après avoir évalué les performances de notre méthode d'optimisation, nous terminons ce document par une synthèse de nos travaux de recherche et proposons des voies possibles pour des travaux futurs, dans le chapitre suivant.

Chapitre 5

Synthèse et perspectives

Sommaire

5.1 Synthèse

Dans ce document, nous nous sommes intéressés au problème d'optimisation de requêtes du aux erreurs d'estimations. Ce problème a été largement étudié dans la littérature en vue de son impact sur la qualité des plans d'exécution.

Nous avons commencé par introduire, dans le chapitre 1, notre problématique de recherche en mettant en évidence nos motivations. Dans le chapitre 2, nous avons étudié les principales méthodes d'optimisation existantes, dont l'objectif est de gérer les erreurs d'estimations et les sous-optimalités conséquentes. Nous avons distingué deux approches que nous avons appelées : (1) *optimisation basée sur des estimations en points singuliers*, et (2) *optimisation basée sur des estimations en un ensemble de points*. Nous avons étudié et comparé les méthodes dans chaque approche, puis nous avons analysé les solutions offertes par ces approches pour faire face à l'imprécision des estimations utilisées par l'optimiseur. La première approche utilise des valeurs singulières d'estimations pour choisir un plan d'exécution estimé optimal. A l'exécution, des statistiques actualisées sont recueillies. Si une erreur d'estimation est détectée, une ré-optimisation est déclenchée pour le reste du plan. Les estimations utilisées à l'étape d'optimisation initiale ou lors d'une ré-optimisation sont considérées précises. Cette approche peut ainsi induire plusieurs ré-optimisations d'un plan, engendrant de mauvaises performances. Pour éviter cela, la deuxième approche a été proposée. Cette approche considère la possibilité d'erreurs d'estimations dès la phase d'optimisation. Ceci est modélisé par l'utilisation d'un ensemble de points d'estimations pour chaque paramètre présumé incertain. Les méthodes dans cette approche cherchent à générer des plans robustes dans le sens où ils sont capables de fournir des performances acceptables et stables pour plusieurs conditions d'exécution. Toutefois, engendrer un plan robuste quand l'ensemble de points d'estimations utilisé est grand, reste complexe. De plus, la majorité de ces méthodes ignorent la variation des conditions d'exécution par rapport aux conditions prévues à l'optimisation. Ainsi, un plan généré est présumé robuste jusqu'à la fin de l'exécution.

Compte tenu de ces constatations, nous avons proposé une méthode d'optimisation hy-

bride qui vise à produire des plans d'exécution robustes-notamment lorsque l'incertitude des estimations utilisées est importante, et à corriger une violation de la robustesse à l'exécution.

Le chapitre 3 a été consacré à décrire notre méthode d'optimisation. Celle-ci se compose de deux modules. Le premier module est une stratégie d'optimisation qui se focalise sur le choix, à l'optimisation, d'un plan d'exécution robuste pour une requête en entrée. Cette stratégie s'appuie sur des intervalles ou des surfaces d'estimations pour modéliser l'incertitude sur les estimations disponibles au moment de l'optimisation. Les coûts de plans candidats à l'exécution de la requête sont comparés dans ces intervalles/surfaces. Les plans offrant des performances robustes sont identifiés. Le plan qui minimise le risque d'une violation de la robustesse à l'exécution est choisi pour commencer l'exécution. Bien que ce risque soit réduit, il est existant et peut avoir un effet néfaste sur la robustesse. Une solution à ce problème est apportée par le deuxième module. Dans ce module, le plan initialement choisi est enrichi par des opérateurs de contrôle et de décision. Ces opérateurs sont insérés dans le plan aux endroits où le risque d'une violation de la robustesse est jugé important par l'utilisateur. En effet, l'utilisateur spécifie à travers un paramètre d'entrée appelé seuil du risque, le risque d'une violation de la robustesse qu'il est prêt à accepter. Les opérateurs de contrôle et de décision collectent des statistiques à l'exécution. Ils vérifient si le plan en cours reste robuste. Si ce n'est pas le cas, ils déclenchent une correction du reste du plan. Ces opérateurs sont capables de prendre des décisions de modifications sans avoir besoin de ré-invoquer l'optimiseur. Ils se basent sur leurs propres données. Une partie des données concerne les intervalles de robustesse d'algorithmes physiques associés à chaque opérateur, et est déterminée au moment de l'optimisation. Les autres données sont des statistiques recueillies à l'exécution.

Le chapitre 4 a été consacré à présenter les résultats de l'évaluation des performances de notre méthode d'optimisation hybride. Pour évaluer les performances de notre méthode, nous avons développé un simulateur. Ce dernier comporte un générateur de requêtes, notre méthode d'optimisation ainsi que les méthodes Rio [BBD05] et ReOpt [KD98],

et une méta-base. Après une description de notre plateforme d'expérimentations, nous avons étudié dans ce chapitre le comportement de notre méthode par rapport aux deux méthodes de références choisies : Rio [BBD05] et ReOpt [KD98]. Ce choix a été motivé par l'étude de ces méthodes dans notre état de l'art dans le chapitre 2. Les résultats de l'évaluation des performances ont mis en évidence un sur-coût d'optimisation engendré par notre méthode, mais qui est compensé à l'exécution. Ce sur-coût est au profit d'une exécution stable et avec des performances acceptables. Nos expérimentations ont montré que notre méthode fournit des temps de réponse plus stables comparé aux autres méthodes, notamment lorsque l'incertitude est importante. L'évaluation des performances a également montré que notre méthode offre des performances comparables aux méthodes existantes quand l'incertitude est faible, mais qu'elle permet d'augmenter jusqu'à 60%, la capacité d'un optimiseur de requêtes de générer des plans robustes lorsque l'incertitude sur les estimations utilisées est importante.

5.2 Perspectives

Nous terminons ce document par présenter des voies possibles pour la continuation de nos travaux de recherche.

Les premières perspectives touchent les points qui n'ont pas pu être approfondis dans cette thèse. Un point important est le ré-ordonnancement des opérateurs d'une requête à l'exécution. Dans ce document, nous avons proposé une stratégie d'ordonnancement des opérateurs d'une requête qui prend en compte l'imprécision des tailles de relations opérandes. Cet ordonnancement est basé sur un raisonnement probabiliste pour trier les relations en fonction de leurs tailles. Il est ensuite maintenu jusqu'à la terminaison de l'exécution. Bien que cet ordonnancement soit sélectionné d'une manière préventive, les performances de notre méthode pourraient être améliorées en tirant profits d'informations collectées à l'exécution. Nous pourrions identifier plusieurs ordonnancements possibles d'opérateurs, tel que chaque ordonnancement est adéquat pour des conditions d'exécu-

tion. Puis, les informations collectées à l'exécution seraient utilisées pour décider si un ré-ordonnancement des opérateurs est susceptible d'améliorer les performances. Si c'est le cas, alors l'ordonnancement adéquat parmi ceux identifiés est choisi, sans besoin de ré-optimisation.

Un autre point de recherche porte sur le comportement des opérateurs de contrôle et de décision, responsables de corriger une violation de la robustesse à l'exécution. Chaque opérateur inséré dans un plan d'exécution prend une décision de correction d'une manière autonome, sans en informer les autres opérateurs. Il serait opportun d'étudier l'apport d'une stratégie de coopération entre les différents opérateurs. Un exemple de coopération concerne la propagation des corrections d'erreurs d'estimations. Qu'un opérateur de contrôle et de décision transmet des informations actualisées à un autre opérateur, pourrait aider ce dernier à restreindre l'intervalle d'estimations sur lequel il porte. Par conséquent, des algorithmes physiques estimés robustes sur les parties rejetés de l'intervalle d'estimations pourraient être aussi rejetés. Ceci pourra améliorer le processus de décision.

En ce qui concerne les perspectives à plus long terme, nous pourrions étendre la notion de robustesse à d'autres contextes. La méthode d'optimisation hybride proposée dans ce document s'applique dans un environnement mono-processeur. Il serait intéressant d'adapter cette méthode à un autre environnement. Par exemple, l'émergence d'applications de bases de données dans les domaines qui font appel à de très grands volumes de données rend la parallélisation des plans de requêtes nécessaire pour obtenir de hautes performances. Il est important de choisir un degré de parallélisme pour chaque opération. Le choix du degré de parallélisme est fonction de valeurs de paramètres comme les tailles des relations à manipulées. Lorsque des statistiques sur les valeurs de ces paramètres sont imprécises ou indisponibles, un degré de parallélisme choisi par un optimiseur peut entrainer de mauvaises performances. Une sur-estimation ou une sous-estimation du degré de parallélisme peut induire un coût important. Par exemple, si l'optimiseur estime que la taille d'une relation à manipuler est faible, il peut choisir un degré faible de parallélisme. Si lors de

l'exécution, la taille de la relation en question s'avère importante, la répartition de l'exécution du plan entre les processeurs peut induire une surcharge de travail pour certains processeurs et par conséquent, de mauvaises performances. Les méthodes existantes dans la littérature (e.g., [AKB+12, BAK+12]) proposent d'adapter le degré de parallélisme aux conditions observées à l'exécution. Lorsque l'environnement d'exécution est hautement évolutif, ses caractéristiques peuvent changer à tout moment, induisant plusieurs adaptations du degré de parallélisme. Cela peut engendrer un sur-coût important. Il serait intéressant d'étudier la possibilité de définir un degré de parallélisme robuste, qui permet d'obtenir des performances stables en évitant un sur-coût relatif à plusieurs adaptations possibles du degré de parallélisme.

Appendices

Annexe

Dans cet annexe, nous détaillons les calculs des formules (3.1) et (3.2) présentées dans le chapitre 3, sous-section 3.2.1, pages 75 et 76.

A.

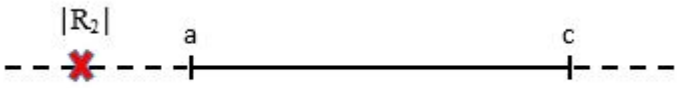
La formule (3.1) présentée dans le chapitre 3 - sous-section 3.2.1 - page 75, calcule la probabilité que la taille d'une relation R_1 (notée $|R_1|$) soit inférieure à la taille d'une relation R_2 (notée $|R_2|$), avec les contraintes suivantes :

— la taille de R_1 est modélisée par un intervalle d'estimations [a, c].

— la taille de R_2 est donnée par une valeur spécifique.

Nous traitons $|R_1|$ et $|R_2|$ comme deux variables aléatoires. Nous supposons que $|R_1|$ suit une loi de probabilité uniforme continue sur l'intervalle I_1= [a, c]. Nous calculons la probabilité $P(|R_1| \leq |R_2|)$ dans les différents cas possibles :

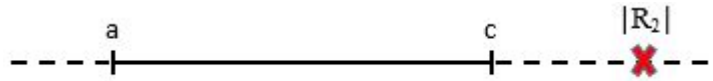
1. **Premier cas : $|R_2| < a$:**



Dans ce cas, $|R_2|$ est toujours inférieure à $|R_1|$, quelque soit la valeur de $|R_1|$ dans [a, c].

Ainsi $P(|R_1| \leq |R_2|) = 0$

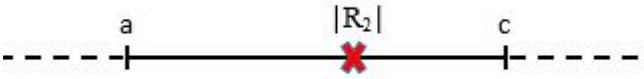
2. **Deuxième cas : $|\mathbf{R}_2| > \mathbf{c}$:**



Dans ce cas, $|R_1|$ est toujours inférieure à $|R_2|$, quelque soit la valeur de $|R_1|$ dans [a, c].

Ainsi $P(|R_1| \leq |R_2|) = 1$

3. **Troisième cas : $\mathbf{a} \leq |\mathbf{R}_2| \leq \mathbf{c}$:**



En théorie des probabilités, la probabilité qu'une variable aléatoire X soit dans un intervalle [x, y] peut être calculée en utilisant la fonction de densité de probabilité associée à cette variable. Une densité de probabilité, notée *f*, est une fonction qui permet de représenter une loi de probabilité sous forme d'intégrales. On dit qu'une fonction f est une densité de probabilité d'une variable aléatoire X si :

$$\forall\ x,\ P(X \leq x) = \int_{-\infty}^{x} f(t)\mathrm{d}t$$

La probabilité $P(x \leq X \leq y)$ se calcule alors comme suit :

$$P(x \leq X \leq y) = \int_{x}^{y} f(t)\mathrm{d}t$$

La loi uniforme sur un intervalle [a, b] est la loi continue dont la densité de proba-

bilité f est la fonction constante sur l'intervalle [a, b] :

$$f(t) = \begin{cases} \dfrac{1}{b-a} & \text{si } a \leq t \leq b \\ 0 & \text{sinon} \end{cases}$$

Revenons à notre calcul. Supposons qu'il existe une densité de probabilité associée à $|\mathrm{R}_1|$ sur I_1, notée f_{R1} et définie par :

$$f_{R1}(t) = \begin{cases} \dfrac{1}{c-a} & \text{si } a \leq t \leq c \\ 0 & \text{sinon} \end{cases}$$

Nous obtenons, $\mathrm{P}(|\mathrm{R}_1| \leq |\mathrm{R}_2|) = \int_a^{|R_2|} f_{R1}(t)\mathrm{d}t = \int_a^{|R_2|} \frac{1}{c-a}\mathrm{d}t = \frac{|R_2|-a}{c-a}$

Ainsi $\mathrm{P}(|\mathrm{R}_1| \leq |\mathrm{R}_2|) = \frac{|R_2|-a}{c-a}$

B.

La formule (3.2) présentée dans le chapitre 3 - sous-section 3.2.1 - page 76, calcule la probabilité que la taille d'une relation R_1 (notée $|\mathrm{R}_1|$) soit inférieure à la taille d'une relation R_2 (notée $|\mathrm{R}_2|$), avec les contraintes suivantes :

— la taille de R_1 est modélisée par un intervalle d'estimations [a, c].

— la taille de R_2 est modélisée par un intervalle d'estimations [b, d].

Nous traitons $|\mathrm{R}_1|$ et $|\mathrm{R}_2|$ comme deux variables aléatoires suivant une loi de probabilité uniforme continue sur les intervalles respectives I_1= [a, c] et I_2= [b, d]. Supposons qu'il existe une densité de probabilité associée à $|\mathrm{R}_1|$ sur I_1, notée f_{R1}, et qui est définie par :

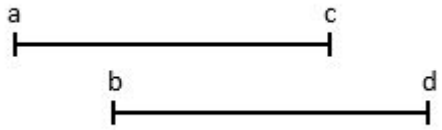
$$f_{R1}(t) = \begin{cases} \dfrac{1}{c-a} & \text{si } a \leq t \leq c \\ 0 & \text{sinon} \end{cases}$$

Supposons qu'il existe une densité de probabilité associée à $|R_2|$ sur I_2, notée f_{R2}, et qui est définie par :

$$f_{R2}(t) = \begin{cases} \dfrac{1}{d-b} & \text{si } b \leq t \leq d \\ 0 & \text{sinon} \end{cases}$$

Nous calculons la probabilité $P(|R_1| \leq |R_2|)$ dans les différents cas possibles :

1. **Premier cas : b ∈ [a, c] et d>c :**



Dans ce cas : $P(|R_1| \leq |R_2|) = P((a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) \cup (a \leq |R_1| \leq c \cap c \leq |R_2| \leq d) \cup (b \leq |R_1| \leq |R_2| \leq c))$

Pour mener ce calcul, nous utilisons la loi de probabilité suivante :
soient deux événements A et B, P(A ∪ B) = P(A) + P(B) - P(A ∩ B)

si A et B sont deux événements indépendants : P(A ∩ B) = P(A) × P(B)

si A et B sont deux événements incompatibles : P(A ∩ B) = 0

Appliquons cette règle à notre cas tel que A correspond à l'événement $(b \leq |R_1| \leq |R_2| \leq c)$, et B correspond à l'événement $(a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) \cup (a \leq |R_1| \leq c \cap c \leq |R_2| \leq d)$.

A et B étant incompatibles, nous obtenons :
$P(|R_1| \leq |R_2|) = P((a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) \cup (a \leq |R_1| \leq c \cap c \leq |R_2| \leq d)) + P(b \leq |R_1| \leq |R_2| \leq c)$

• Commençons par calculer la probabilité P((a ≤ |R_1| ≤ b ∩ b ≤ |R_2| ≤ d) ∪ (a ≤ |R_1| ≤ c ∩ c ≤ |R_2| ≤ d)) :

Les deux événements (a ≤ |R_1| ≤ b ∩ b ≤ |R_2| ≤ d) et (a ≤ |R_1| ≤ c ∩ c ≤ |R_2| ≤ d) sont compatibles. En appliquant la formule P(A ∪ B) = P(A) + P(B) - P(A ∩ B), nous obtenons :

P((a ≤ |R_1| ≤ b ∩ b ≤ |R_2| ≤ d) ∪ (a ≤ |R_1| ≤ c ∩ c ≤ |R_2| ≤ d)) = P(a ≤ |R_1| ≤ b ∩ b ≤ |R_2| ≤ d) + P(a ≤ |R_1| ≤ c ∩ c ≤ |R_2| ≤ d) - P(a ≤ |R_1| ≤ b ∩ b ≤ |R_2| ≤ d) × P(a ≤ |R_1| ≤ c ∩ c ≤ |R_2| ≤ d)

Les événements (a ≤ |R_1| ≤ b) et (b ≤ |R_2| ≤ d) sont indépendants.

Les événements (a ≤ |R_1| ≤ c) et (c ≤ |R_2| ≤ d) sont également indépendants.

Nous obtenons alors :

P((a ≤ |R_1| ≤ b ∩ b ≤ |R_2| ≤ d) ∪ (a ≤ |R_1| ≤ c ∩ c ≤ |R_2| ≤ d)) = P(a ≤ |R_1| ≤ b) × P(b ≤ |R_2| ≤ d) + P(a ≤ |R_1| ≤ c) × P(c ≤ |R_2| ≤ d) - P(a ≤ |R_1| ≤ b) × P(b ≤ |R_2| ≤ d) × P(a ≤ |R_1| ≤ c) × P(c ≤ |R_2| ≤ d)

* $P(a \leq |R_1| \leq b) = \int_a^b f_{R1}(t)dt = \int_a^b \frac{1}{c-a}dt = \frac{1}{c-a} \times [t]_a^b = \frac{b-a}{c-a}$
* $P(a \leq |R_1| \leq c) = \int_a^c f_{R1}(t)dt = \int_a^c \frac{1}{c-a}dt = \frac{1}{c-a} \times [t]_a^c = \frac{c-a}{c-a} = 1$
* $P(c \leq |R_2| \leq d) = \int_c^d f_{R2}(t)dt = \int_c^d \frac{1}{d-b}dt = \frac{1}{d-b} \times [t]_c^d = \frac{d-c}{d-b}$
* $P(b \leq |R_2| \leq d) = \int_b^d f_{R2}(t)dt = \int_b^d \frac{1}{d-b}dt = \frac{1}{d-b} \times [t]_b^d = \frac{d-b}{d-b} = 1$

Ainsi P((a ≤ |R_1| ≤ b ∩ b ≤ |R_2| ≤ d) ∪ (a ≤ |R_1| ≤ c ∩ c ≤ |R_2| ≤ d) = $\frac{b-a}{c-a} + \frac{d-c}{d-b} - \frac{b-a}{c-a} \times \frac{d-c}{d-b}$

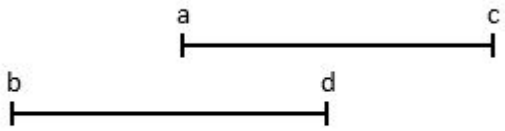
• Calculons maintenant P(b ≤ |R_1| ≤ |R_2| ≤ c) :

P(b ≤ |R_1| ≤ |R_2| ≤ c) = P(|R_1| ≤ |R_2| tel que b ≤ |R_1| ≤ c et b ≤ |R_2| ≤ c)

$= \int_b^c$ P(|R_1|=t et |R_2| $\geq$ t) dt = $\int_b^c \frac{1}{c-b} \times \frac{c-t}{c-b}dt$
$= \frac{1}{(c-b)^2} \times \left[\frac{-(c-t)^2}{2}\right]_b^c = \frac{1}{2}$

Nous obtenons comme résultat final P(|R_1| $\leq$ |R_2|)= $\frac{b-a}{c-a} + \frac{d-c}{d-b} - \frac{b-a}{c-a} \times \frac{d-c}{d-b} + \frac{1}{2}$

2. **Deuxième cas : a $\in$ [b, d] et c>d :**



Dans ce cas, P(|R_1| $\leq$ |R_2|) = P(a $\leq$ |R_1| $\leq$ |R_2| $\leq$ d) - P((a $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ a) $\cup$ (d $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ d))

• Calculons dans un premier temps P(a $\leq$ |R_1| $\leq$ |R_2| $\leq$ d) :

P(a $\leq$ |R_1| $\leq$ |R_2| $\leq$ d) = P(|R_1| $\leq$ |R_2| tel que a $\leq$ |R_1| $\leq$ d et a $\leq$ |R_2| $\leq$ d)
$= \int_a^d$ P(|R_1|=t et |R_2| $\geq$ t)dt = $\int_a^d \frac{1}{d-a} \times \frac{d-t}{d-a}dt$
$= \frac{1}{(d-a)^2} \times \left[\frac{-(d-t)^2}{2}\right]_a^d$
$= \frac{1}{2}$

• Calculons maintenant P((a $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ a) $\cup$ (d $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ d)) :

En appliquant la formule P(A $\cup$ B) = P(A) + P(B) - P(A $\cap$ B), nous obtenons :

P((a $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ a) $\cup$ (d $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ d)) = P(a $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ a) + P(d $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ d) - P(a $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ a) $\times$ P(d $\leq$ |R_1| $\leq$ c $\cap$ b $\leq$ |R_2| $\leq$ d)

Les événements (a $\leq$ |R_1| $\leq$ c) et (b $\leq$ |R_2| $\leq$ a) sont indépendants. Les événements

(d $\leq |R_1| \leq$ c) et (b $\leq |R_2| \leq$ d) le sont aussi, cela donne :

P(a $\leq |R_1| \leq$ c $\cap$ b $\leq |R_2| \leq$ a) = P(a $\leq |R_1| \leq$ c) $\times$ P(b $\leq |R_2| \leq$ a)

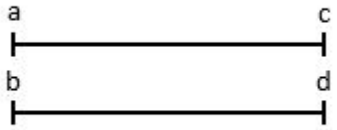
P(d $\leq |R_1| \leq$ c $\cap$ b $\leq |R_2| \leq$ d) = P(d $\leq |R_1| \leq$ c) $\times$ P(b $\leq |R_2| \leq$ d)

* P(a $\leq |R_1| \leq$ c) = $\int_a^c f_{R1}(t)\mathrm{d}t = \int_a^c \frac{1}{c-a}\mathrm{d}t = \frac{1}{c-a} \times [t]_a^c = \frac{c-a}{c-a} = 1$
* P(d $\leq |R_1| \leq$ c) = $\int_d^c f_{R1}(t)\mathrm{d}t = \int_d^c \frac{1}{c-a}\mathrm{d}t = \frac{1}{c-a} \times [t]_d^c = \frac{c-d}{c-a}$
* P(b $\leq |R_2| \leq$ d) = $\int_b^d f_{R2}(t)\mathrm{d}t = \int_b^d \frac{1}{d-b}\mathrm{d}t = \frac{1}{d-b} \times [t]_b^d = \frac{d-b}{d-b} = 1$
* P(b $\leq |R_2| \leq$ a) = $\int_b^a f_{R2}(t)\mathrm{d}t = \int_b^a \frac{1}{d-b}\mathrm{d}t = \frac{1}{d-b} \times [t]_b^a = \frac{a-b}{d-b}$

Ainsi, P((a $\leq |R_1| \leq$ c $\cap$ b $\leq |R_2| \leq$ a) $\cup$ (d $\leq |R_1| \leq$ c $\cap$ b $\leq |R_2| \leq$ d)) = $\frac{a-b}{d-b} + \frac{c-d}{c-a} - \frac{a-b}{d-b} \times \frac{c-d}{c-a}$

Le résultat final est P($|R_1| \leq |R_2|$)= $\frac{1}{2} - \frac{a-b}{d-b} - \frac{c-d}{c-a} + \frac{a-b}{d-b} \times \frac{c-d}{c-a}$

3. **Troisième cas : [a, c] = [b, d] :**

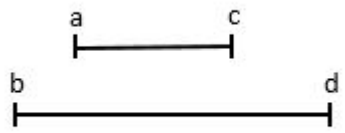


Dans ce cas, la probabilité P($|R_1| \leq |R_2|$) est donnée par :

$$\begin{aligned} P(|R_1| \leq |R_2|) &= \int_a^c P(|R_1|=t \text{ et } |R_2| \geq t)\mathrm{d}t = \int_a^c \frac{1}{c-a} \times \frac{c-t}{c-a}\mathrm{d}t \\ &= \frac{1}{(c-a)^2} \times \left[\frac{-(c-t)^2}{2}\right]_a^c \\ &= \frac{1}{2} \end{aligned}$$

Nous pouvons calculer ce cas à partir des cas précédents en remplaçant a par b (a=b) et c par d (c=d). Nous obtenons le même résultat : P($|R_1| \leq |R_2|$)= $\frac{1}{2}$

4. **Quatrième cas : [a, c] $\subset$ [b, d] :**



Dans ce cas, P($|R_1| \leq |R_2|$) = P($(a \leq |R_1| \leq |R_2| \leq c) \cup (a \leq |R_1| \leq c \cap c \leq |R_2| \leq d)$) - P($a \leq |R_1| \leq c \cap b \leq |R_2| \leq a$)

• Calculons P($(a \leq |R_1| \leq |R_2| \leq c) \cup (a \leq |R_1| \leq c \cap c \leq |R_2| \leq d)$) :
Les événements ($a \leq |R_1| \leq |R_2| \leq c$) et ($a \leq |R_1| \leq c \cap c \leq |R_2| \leq d$) sont incompatibles. En appliquant la formule P(A $\cup$ B) = P(A) + P(B), nous obtenons : P($(a \leq |R_1| \leq |R_2| \leq c) \cup (a \leq |R_1| \leq c \cap c \leq |R_2| \leq d)$) = P($a \leq |R_1| \leq |R_2| \leq c$) + P($a \leq |R_1| \leq c \cap c \leq |R_2| \leq d$)

$$\begin{aligned} P(a \leq |R_1| \leq |R_2| \leq c) &= P(|R_1| \leq |R_2| \text{ tel que } a \leq |R_1| \leq c \text{ et } a \leq |R_2| \leq c) \\ &= \int_a^c P(|R_1| = t \text{ et } |R_2| \geq t)dt = \int_a^c \frac{1}{c-a} \times \frac{c-t}{c-a}dt \\ &= \frac{1}{(c-a)^2} \times \left[\frac{-(c-t)^2}{2}\right]_a^c \\ &= \frac{1}{2} \end{aligned}$$

($a \leq |R_1| \leq c$) et ($c \leq |R_2| \leq d$) sont deux événements indépendants. Ainsi P($a \leq |R_1| \leq c \cap c \leq |R_2| \leq d$)

$$\begin{aligned} &= P(a \leq |R_1| \leq c) \times P(c \leq |R_2| \leq d) \\ &= \int_a^c \frac{1}{c-a}dt \times \int_c^d \frac{1}{d-b}dt = \frac{1}{c-a} \times [t]_a^c \times \frac{1}{d-b} \times [t]_c^d \\ &= \frac{d-c}{d-b} \end{aligned}$$

• Calculons P($a \leq |R_1| \leq c \cap b \leq |R_2| \leq a$) :

Les événements ($a \leq |R_1| \leq c$) et ($b \leq |R_2| \leq a$) sont indépendants.

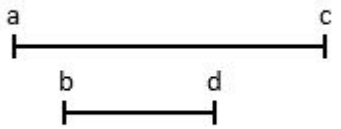
Ainsi $P(a \leq |R_1| \leq c \cap b \leq |R_2| \leq a) = P(a \leq |R_1| \leq c) \times P(b \leq |R_2| \leq a)$

$= \int_a^c \frac{1}{c-a} \times \int_b^a \frac{1}{d-b} dt = \frac{c-a}{c-a} \times \frac{a-b}{d-b}$

$= \frac{a-b}{d-b}$

Le résultat final est $P(|R_1| \leq |R_2|) = \frac{1}{2} + \frac{d-c}{d-b} - \frac{a-b}{d-b}$

5. **Cinquième cas : [b, d] $\subset$ [a, c] :**



Dans ce cas $P(|R_1| \leq |R_2|) = P((a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) \cup (b \leq |R_1| \leq |R_2| \leq d))$ - $P(d \leq |R_1| \leq c \cap b \leq |R_2| \leq d)$

• Calculons $P((a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) \cup (b \leq |R_1| \leq |R_2| \leq d))$:

Les événements ($a \leq |R_1| \leq b \cap b \leq |R_2| \leq d$) et ($b \leq |R_1| \leq |R_2| \leq d$) sont incompatibles. En appliquant la formule $P(A \cup B) = P(A) + P(B)$, nous obtenons : $P((a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) \cup (b \leq |R_1| \leq |R_2| \leq d)) = P(a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) + P(b \leq |R_1| \leq |R_2| \leq d)$

$P(b \leq |R_1| \leq |R_2| \leq d) = P(|R_1| \leq |R_2|$ tel que $b \leq |R_1| \leq d$ et $b \leq |R_2| \leq d)$

$= \int_b^d P(|R_1| = t$ et $|R_2| \geq t) dt = \int_b^d \frac{1}{d-b} \times \frac{d-t}{d-b} dt$

$= \frac{1}{(d-b)^2} \times \left[\frac{-(d-t)^2}{2}\right]_b^d$

$= \frac{1}{2}$

$P(a \leq |R_1| \leq b \cap b \leq |R_2| \leq d) = P(a \leq |R_1| \leq b) \times P(b \leq |R_2| \leq d)$ car $(a \leq |R_1| \leq b)$ et $(b \leq |R_2| \leq d)$ sont deux événements indépendants.

$$= \int_a^b \frac{1}{c-a} \mathrm{d}t \times \int_b^d \frac{1}{d-b} \mathrm{d}t = \frac{1}{c-a} \times [t]_a^b \times \frac{1}{d-b} \times [t]_b^d$$
$$= \frac{b-a}{c-a}$$

• Calculons $P(d \leq |R_1| \leq c \cap b \leq |R_2| \leq d)$:

Les événements $(d \leq |R_1| \leq c)$ et $(b \leq |R_2| \leq d)$ sont indépendants.

Ainsi $P(d \leq |R_1| \leq c \cap b \leq |R_2| \leq d) = P(d \leq |R_1| \leq c) \times P(b \leq |R_2| \leq d)$

$$= \int_d^c \frac{1}{c-a} \times \int_b^d \frac{1}{d-b} \mathrm{d}t = \frac{c-d}{c-a} \times \frac{d-b}{d-b}$$
$$= \frac{c-d}{c-a}$$

Le résultat final est $P(|R_1| \leq |R_2|) = \frac{1}{2} + \frac{b-a}{c-a} - \frac{c-d}{c-a}$

Bibliographie

[ABD+10] M. Abhirama, Sourjya Bhaumik, Atreyee Dey, Harsh Shrimal, and Jayant R. Haritsa. On the stability of plan costs and the costs of plan stability. *Proc. VLDB Endow.*, 3(1-2) :1137–1148, September 2010.

[AH00] Ron Avnur and Joseph M. Hellerstein. Eddies : Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 261–272, New York, NY, USA, 2000. ACM.

[AHW15a] Khaled H. Alyoubi, Sven Helmer, and Peter T. Wood. Ordering selection operators under partial ignorance. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1521–1530, New York, NY, USA, 2015. ACM.

[AHW15b] Khaled H. Alyoubi, Sven Helmer, and Peter T. Wood. Ordering selection operators using the minmax regret rule. *CoRR*, abs/1507.08257, 2015.

[AKB+12] Sameer Agarwal, Srikanth Kandula, Nicolas Bruno, Ming-Chuan Wu, Ion Stoica, and Jingren Zhou. Reoptimizing data parallel computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 281–294, 2012.

[AU10] Foto N. Afrati and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Exten-*

ding Database Technology, EDBT '10, pages 99–110, New York, NY, USA, 2010. ACM.

[BAK+12] Nicolas Bruno, Sameer Agarwal, Srikanth Kandula, Bing Shi, Ming-Chuan Wu, and Jingren Zhou. Recurring job optimization in scope. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 805–806, 2012.

[BB05] Shivnath Babu and Pedro Bizarro. Adaptive query processing in the looking glass. In *CIDR*, pages 238–249, 2005.

[BBD05] Shivnath Babu, Pedro Bizarro, and David DeWitt. Proactive re-optimization. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 107–118, New York, NY, USA, 2005. ACM.

[BC05] Brian Babcock and Surajit Chaudhuri. Towards a robust query optimizer : A principled and practical approach. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 119–130, New York, NY, USA, 2005. ACM.

[BCG01] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Stholes : A multidimensional workload-aware histogram. *SIGMOD Rec.*, 30(2) :211–222, May 2001.

[BJZ13] Nicolas Bruno, Sapna Jain, and Jingren Zhou. Continuous cloud-scale query optimization and processing. *PVLDB*, 6(11) :961–972, 2013.

[CG94] Richard L. Cole and Goetz Graefe. Optimization of dynamic query evaluation plans. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, SIGMOD '94, pages 150–160, New York, NY, USA, 1994. ACM.

[CHG02] Francis Chu, Joseph Halpern, and Johannes Gehrke. Least expected cost query optimization : What can we expect ? In *Proceedings of the Twenty-*

first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '02, pages 293–302, New York, NY, USA, 2002. ACM.

[Chr84] S. Christodoulakis. Implications of certain assumptions in database performance evauation. *ACM Trans. Database Syst.*, 9(2) :163–186, June 1984.

[CK14] Doulkeridis Christos and Norvag Kjetil. A survey of large-scale analytical query processing in mapreduce. *The VLDB Journal*, 23(3) :355–380, June 2014.

[CNR08] Surajit Chaudhuri, Vivek Narasayya, and Ravi Ramamurthy. A pay-as-you-go framework for query execution feedback. *Proc. VLDB Endow.*, 1(1) :1141–1152, August 2008.

[CR94] Chungmin Melvin Chen and Nick Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, SIGMOD '94, pages 161–172, New York, NY, USA, 1994. ACM.

[DDH08] Harish D., Pooja N. Darera, and Jayant R. Haritsa. Identifying robust plans through plan diagram reduction. *Proc. VLDB Endow.*, 1(1) :1124–1140, August 2008.

[DGR01] Amol Deshpande, Minos N. Garofalakis, and Rajeev Rastogi. Independence is good : Dependency-based histogram synopses for high-dimensional data. In Sharad Mehrotra and Timos K. Sellis, editors, *SIGMOD Conference*, pages 199–210. ACM, 2001.

[DH14] Anshuman Dutt and Jayant R. Haritsa. Plan bouquets : Query processing without selectivity estimation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1039–1050. ACM, 2014.

[DH16] Anshuman Dutt and Jayant R. Haritsa. Plan bouquets : A fragrant approach to robust query processing. *ACM Trans. Database Syst.*, 41(2) :11 :1–11 :37, May 2016.

[DIR07] Amol Deshpande, Zachary Ives, and Vijayshankar Raman. Adaptive query processing. *Found. Trends databases*, 1(1) :1–140, January 2007.

[DNH14] Anshuman Dutt, Sumit Neelam, and Jayant R. Haritsa. Quest : An exploratory approach to robust query processing. *Proc. VLDB Endow.*, 7(13) :1585–1588, August 2014.

[EDNO97] Cem Evrendilek, Asuman Dogac, Sena Nural, and Fatma Ozcan. Multidatabase query optimization. *Distrib. Parallel Databases*, 5(1) :77–114, January 1997.

[GMWU99] Hector Garcia-Molina, Jennifer Widom, and Jeffrey D. Ullman. *Database System Implementation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[Gra11] Goetz Graefe. Robust query processing. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, page 1361, 2011.

[GSW12] Zhongxian Gu, Mohamed A. Soliman, and Florian M. Waas. Testing the accuracy of query optimizers. In *Proceedings of the Fifth International Workshop on Testing Database Systems*, DBTest '12, pages 11 :1–11 :6, New York, NY, USA, 2012. ACM.

[GTK01] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 461–472, New York, NY, USA, 2001. ACM.

[HCW+13] Hakan Hacigumus, Yun Chi, Wentao Wu, Shenghuo Zhu, Junichi Tatemura, and Jeffrey F. Naughton. Predicting query execution time : Are optimizer cost models really unusable ? In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 1081–1092, Washington, DC, USA, 2013. IEEE Computer Society.

[HS02] Arvind Hulgeri and S. Sudarshan. Parametric query optimization for linear and piecewise linear cost functions. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 167–178. VLDB Endowment, 2002.

[IC91] Yannis E. Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, SIGMOD '91, pages 268–277, New York, NY, USA, 1991. ACM.

[IFF+99] Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Daniel S. Weld. An adaptive query execution system for data integration. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 299–310, New York, NY, USA, 1999. ACM.

[IHW04] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Adapting to source properties in processing data integration queries. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 395–406, New York, NY, USA, 2004. ACM.

[INSS92] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric query optimization. In *Proceedings of the 18th International Conference on Very Large Data Bases*, VLDB '92, pages 103–114, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[Ioa96] Yannis E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1) :121–123, March 1996.

[JKM+98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 275–286, San Francisco, CA, USA, 1998.

[JMP13] Richard A. Tapia Joanna M. Papakonstantinou. Origin and evolution of the secant method in one dimension. *The American Mathematical Monthly*, 120(6) :500–518, 2013.

[KBK+14] Konstantinos Karanasos, Andrey Balmin, Marcel Kutsch, Fatma Ozcan, Vuk Ercegovac, Chunyang Xia, and Jesse Jackson. Dynamically optimizing queries over large scale data platforms. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 943–954, New York, NY, USA, 2014. ACM.

[KD98] Navin Kabra and David J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 106–117, New York, NY, USA, 1998. ACM.

[LGM+15] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really ? *Proc. VLDB Endow.*, 9(3) :204–215, November 2015.

[LKC99] Ju-Hong Lee, Deok-Hwan Kim, and Chin-Wan Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 205–214, New York, NY, USA, 1999. ACM.

[MFTU96] Laurent Amsaleg Michael, Michael J. Franklin, Anthony Tomasic, and Tolga Urhan. Scrambling query plans to cope with unexpected delays. In *In PDIS*, pages 208–219. IEEE Computer Society, 1996.

[MMH16] Chiraz Moumen, Franck Morvan, and Abdelkader Hameurlain. Estimation error-aware query optimization : an overview. *Comput. Syst. Sci. Eng.*, 31(3), 2016.

[MRS+04] Volker Markl, Vijayshankar Raman, David Simmen, Guy Lohman, Hamid Pirahesh, and Miso Cilimdzic. Robust query processing through progressive optimization. In *Proceedings of the 2004 ACM SIGMOD Internatio-*

nal Conference on Management of Data, SIGMOD '04, pages 659–670, New York, NY, USA, 2004. ACM.

[NGl13] Thomas Neumann and Cesar Galindo-legaria. Taking the edge off cardinality estimation errors using incremental execution. In *In Proc. der GI-Fachtagung Datenbanksysteme für Büro, Technik und Wissenschaft (BTW*, pages 73–92, 2013.

[PHIS96] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 294–305, New York, NY, USA, 1996.

[PI97] Viswanath Poosala and Yannis E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 486–495, 1997.

[SAC+79] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, SIGMOD '79, pages 23–34, New York, 1979.

[SB99] Jennifer M. Schopf and Francine Berman. Using stochastic intervals to predict application behavior on contended resources. In *Proceedings of the Workshop on Advances in Parallel Computing Models*, ISPAN 99, 1999.

[SLMK01] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. Leo - db2's learning optimizer. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 19–28, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[TDJ11] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB*, page 2011, 2011.

[TDJ13] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1) :3–27, February 2013.

[TK15] Immanuel Trummer and Christoph Koch. Probably approximately optimal query optimization. *CoRR*, abs/1511.01782, 2015.

[WCHN13] Wentao Wu, Yun Chi, Hakan Hacígümüş, and Jeffrey F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *Proc. VLDB Endow.*, 6(10) :925–936, August 2013.

[WLMO11] Sai Wu, Feng Li, Sharad Mehrotra, and Beng Chin Ooi. Query optimization for massively parallel data processing. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 12 :1–12 :13, New York, 2011.

[WNS16] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. Sampling-based query re-optimization. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1721–1736, New York, 2016.

[WWHN14] Wentao Wu, Xi Wu, Hakan Hacigümüş, and Jeffrey F. Naughton. Uncertainty aware query execution time prediction. *Proc. VLDB Endow.*, 7(14) :1857–1868, October 2014.

[WY76] Eugene Wong and Karel Youssefi. Decomposition&mdash ;a strategy for query processing. *ACM Trans. Database Syst.*, 1(3) :223–241, 1976.

[YHM15] Shaoyi Yin, Abdelkader Hameurlain, and Franck Morvan. Robust query optimization methods with respect to estimation errors : A survey. *SIGMOD Rec.*, 44(3) :25–36, December 2015.

[ZKP00] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.