La Fragmentation Horizontale Revisitée: Prise en Compte de l'Interaction de Requêtes

Amira Kerkad*, Ladjel Bellatreche*, Dominique Geniet*

*LIAS/ISAE-ENSMA, Futuroscope, Poitiers, France
(amira.kerkad, bellatreche, dominique.geniet)@ensma.fr

Résumé. La principale caractéristique des requêtes définies sur un entrepôt de données relationnel ($\mathcal{EDR}$) est le fait que leurs jointures passent systématiquement par la table des faits. Cette situation favorise l'interaction entre les requêtes. Cette interaction a été largement exploitée par les algorithmes d'optimisation de requêtes dans les bases de données traditionnelles, connue sous le nom d'optimisation multi-requêtes. Dans les $\mathcal{EDR}$, cette interaction a été utilisée pour définir des méthodes de sélection des vues matérialisées. Dans cet article, nous revisitons le problème de sélection de schéma de fragmentation horizontale. Après un état de l'art, où nous soulignons le fait que les algorithmes existants ignorent l'interaction entre les requêtes, nous proposons un nouvel algorithme basé sur cette interaction. Sa principale caractéristique est l'utilisation d'une structure de données incrémentale considérant l'interaction. Il utilise le principe d'élection des requêtes pour aiguiller le processus de fragmentation et propager le gain au mieux sur l'ensemble des requêtes de la charge. Finalement, une étude expérimentale est conduite pour prouver l'efficacité de notre approche.

1 Introduction

Comme pour tout développement de projet informatique, la conception des applications consommatrices des données (bases de données, entreposage de données, etc.) passe par un cycle de vie. Dans la première génération de bases de données, ce cycle contenait quatre étapes principales : **(1)** la collecte des besoins, **(2)** la modélisation conceptuelle, **(3)** la modélisation logique et **(4)** la modélisation physique. La dernière phase a connu une importance particulière avec l'apparition des applications décisionnelles. Ces dernières manipulent une masse importante de données, accédée par des requêtes complexes exigeant un temps réponse rapide pour satisfaire les décideurs. Plus la compétitivité entre entreprises augmente plus cette exigence est primordiale. Avec l'évolution des réseaux et des systèmes de stockage et de traitement, le cycle de vie a été étendu pour prendre en considération la phase de déploiement. Cette dernière inclut les architectures matérielles sur lesquelles les données sont déployées (systèmes distribués, systèmes parallèles, clusters, grille de calcul, informatique dans les nuages, etc.).

La fragmentation horizontale ($\mathcal{FH}$) a la particularité d'être utilisée au niveau logique, physique et déploiement. Au niveau logique, elle a été largement considérée comme une technique de conception de bases de données centralisées et réparties Karlapalem (1996). Dans les entrepôts de données, elle est positionnée comme une technique d'optimisation. Contrairement aux

autres structures d'optimisation (comme les index ou les vues matérialisées), où la sélection des schémas d'optimisation se fait généralement lorsque la base de données est opérationnelle (ou créée), la sélection d'un schéma de $\mathcal{FH}$ d'une base de données (ou d'un entrepôt de données) doit se décider *avant sa création* Corporation (2010). Cette situation rend sa sélection plus sensible que les autres structures. Au niveau du déploiement, elle est la *pré-condition* pour concevoir des bases de données sur des systèmes répartis. L'idée sous-jacente à la $\mathcal{FH}$ est de pouvoir constituer des ensembles de données partielles dont les n-uplets (ou instances d'objet) ont des propriétés géographiques communes. Plusieurs travaux se sont intéressés à la $\mathcal{FH}$. Ils proviennent de deux milieux différents : industriel Sanjay et al. (2004); Zilio et al. (2004) et académique (cf. Section 2). Les éditeurs de SGBD commerciaux (Oracle, DB2, SQL Server, Sybase, etc.) ou non commerciaux (PostgreSQL, MySql, etc.) s'intéressent de plus en plus à la $\mathcal{FH}$ en proposant des commandes au niveau du langage de définition de données (DDL) pour la supporter. Plusieurs modes de $\mathcal{FH}$ ont été proposés (Range, List, Hash, Composite). Récemment, Oracle 11g a fait évoluer la $\mathcal{FH}$ en proposant d'autres modes : (1) le fragmentation par une colonne virtuelle (virtual column partitioning) dans lequel, une table est fragmentée en utilisant un attribut virtuel qui est défini par une expression utilisant un ou plusieurs attributs. Cette colonne est stockée seulement dans les méta-données. (2) Le fragmentation par référence (referential partitioning) qui permet de fragmenter une table en utilisant une autre table (à condition qu'il y ait une relation de type père-fils entre les deux tables Ceri et al. (1982). Ce mode est adapté à la $\mathcal{FH}$ dérivée largement utilisée dans les entrepôts de données Bellatreche et al. (2009). Des opérations ont été définies pour manipuler des partitions. Nous pouvons ainsi citer l'opération de fusion de deux partitions (*Merge Partition*), l'opération d'éclatement d'une partition en deux partitions (*Split Partitio*n), l'opération de conversion d'une partition en une table (*Exchange Partition*), etc.

Au niveau académique, le problème de sélection de schéma $\mathcal{FH}$ a un grand intérêt sur le plan du développement d'algorithmes et de méthodologies. En nous penchant sur ces travaux, nous avons constaté trois principaux points : (1) la majorité des travaux est basée sur un ensemble de requêtes fréquentes ; (2) tous les attributs de sélection utilisés par les algorithmes de sélection ont la même probabilité d'être utilisés pour partitionner l'entrepôt de données, et (3) les modèles de coût utilisés pour quantifier la qualité du schéma de $\mathcal{FH}$ sont simples et souvent estiment le coût de requêtes d'une manière isolée. Dans le contexte des $\mathcal{EDR}$, une charge de requêtes typique se compose d'un mélange de plusieurs requêtes de différents types, qui s'exécutent simultanément et interagissent les unes avec les autres. Cela est dû au fait que la jointure passe systématiquement par la table centrale de l'entrepôt qui est la table des faits. Par conséquent, l'optimisation des performances nécessite de considérer des mélanges de requêtes et leurs interactions, plutôt que les requêtes individuelles ou des types de requêtes Ahmad et al. (2011). Cette interaction a été largement exploitée pour optimiser les requêtes traditionnelles Sellis (1988) et les requêtes sémantiques (en utilisant le langage SparQL) Le et al. (2012) en exploitant les expressions communes entre les requêtes (optimisation multi-requêtes). Elle a été également utilisée pour résoudre le problème de sélection de vues matérialisées Yang et al. (1997), l'ordonnancement de requêtes Phan et Li (2008), l'économie d'énergie Ahmad et al. (2011), la gestion des caches O'Gorman et al. (2002). Pour incorporer l'interaction de requêtes ($\mathcal{IER}$) dans le processus de sélection de schéma $\mathcal{FH}$, trois principales alternatives peuvent être distinguées : (1) l'amélioration de la qualité est déléguée au modèle de coût utilisé pour estimer l'ensemble le traitement des requêtes, (2) le développement d'algorithmes exploitant

une structure de données représentant cette interaction et (3) une variante hybride. La première alternative peut augmenter la complexité des algorithmes, alors que la deuxième et la troisième peuvent réduire considérablement cette complexité en offrant des schémas de fragmentation de haute qualité.

Dans cet article, nous proposons une nouvelle approche pour la $\mathcal{FH}$ dans les $\mathcal{EDR}$ basée sur l'interaction entre les requêtes. Cette interaction est intégrée dans une structure de données incrémentale.

Notre article est structuré comme suit. La section 2 donne un état de l'art conséquent sur le problème de la $\mathcal{FH}$ avec une nouvelle classification des algorithmes principaux. Dans la section 3, un exemple motivant notre approche est décrit afin de faciliter la description de nos algorithmes. La section 4 propose des définitions de base et une algèbre manipulant notre structure de données. La section 5 présente l'algorithme de fragmentation. La section 6 décrit les résultats de nos heuristiques et compare les résultats avec ceux obtenus par l'état de d'art. Enfin, la section 7 récapitule les principaux résultats et propose quelques perspectives à explorer.

2 Etat de l'art

Après avoir exploré la littérature, nous proposons une classification des algorithmes existant en deux catégories : *approches sans contrainte* et *approches avec contrainte*. Les premiers travaux sur la $\mathcal{FH}$ se situent dans la première catégorie. Ces travaux sélectionnent un schéma de fragmentation d'une base de données sans prendre en considération le nombre de fragments finaux de l'entrepôt. Nous identifions deux classes d'approches dans cette catégorie : *approches basées sur la génération des minterms* Özsu et Valduriez (1999) et *approches basées sur l'affinité* Karlapalem (1996).

Les approches basées sur les minterms sont simples, mais coûteuses en termes de temps de calcul pour générer un schéma de $\mathcal{FH}$, surtout lorsque le nombre de prédicats est important. Cette approche a été utilisée dans Papadomanolakis et Ailamaki (2004). Afin de réduire cette complexité, une autre approche Karlapalem (1996) a été proposé adaptant l'algorithme de fragmentation vertical Navathe et Ra (1989). Les prédicats de sélection ayant une grande affinité sont regroupés ensemble. L'affinité entre deux prédicats p_i et p_j est calculée comme la somme des fréquences des requêtes y accédant simultanément. Chaque groupe fournit un fragment horizontal comme une conjonction de ses prédicats. Cet algorithme a une complexité réduite Karlapalem et al. (1996), mais il ne prend en considération que les fréquences d'accès pour la génération des fragments horizontaux et ignore d'autres paramètres comme la taille des tables, les facteurs de sélectivité de prédicats, etc.

Les concepteurs qui utilisent les approches basées sur les *minterms* et les *affinités* ne connaissent pas le nombre de fragments au préalable. Ce dernier n'est connu qu'à la fin de l'exécution de l'algorithme. Face au besoin d'avoir un nombre de fragments raisonnable pour réduire les coûts de maintenance des fragments, de nouvelles approches *basées sur le seuil* ont vu le jour. Ce seuil représente le nombre maximal de fragments que le DBA permet à la $\mathcal{FH}$ de générer. Le principal objectif des approches *avec contraintes* est de partitionner une table en N fragments tels que N soit inférieur ou égal au seuil W imposé par le DBA. Deux classes principales existent dans cette catégorie : les classes *basées sur un modèle coût* Bellatreche et al. (2009) et celles *basées sur les techniques du data mining* Mahboubi et Darmont (2009).

Pour résumer, nous pouvons dire que la première génération d'algorithmes de $\mathcal{FH}$ ne considèrent pas l'interaction entre les requêtes. Ces travaux supposent que tous les prédicats utilisés par les requêtes ont la même probabilité d'occurrence pendant le traitement des requêtes. Les algorithmes dirigés par le coût représentent un progrès réel, et les modèles de coût pourraient être des candidats pour considérer l'interaction. Malheureusement, ils sont simples et supposent que les requêtes sont exécutées de manière isolée les unes des autres.

3 Exemple de Motivation

Pour mieux illustrer l'idée à la base de notre proposition, considérons l'exemple suivant. Soit un entrepôt de données avec une table de faits $Ventes$ et trois tables de dimensions $\{Date\,;\, Produit\,;\, Client\}$, et une charge de 10 requêtes de jointures en étoile. Les plans d'évaluation de chaque requête sont construits et fusionnés en un seul graphe dit : MVPP (*Multiple View Processing Plan*) Yang et al. (1997). Le MVPP est une représentation graphique de la charge proposée dans le contexte d'optimisation multi requêtes (*Multi-Query Optimization* Sellis (1988)). La Figure 1 montre le graphe obtenu à partir de la charge. Notons que les opérations de sélection sont poussées en bas après la construction du MVPP. Ce graphe contient quatre principaux niveaux : **(a)** les noeuds feuilles représentant les tables, **(b)** les noeuds de sélection pouvant participer dans la fragmentation ((e.g., $\{s_1, s_2 \ldots s_8\}$)), **(c)** les opérations binaires (e.g., $\{j_1; j_2 \ldots; j_9\}$) et **(d)** les groupements, tri et projections (e.g., $\{gop_1; gop_2; \ldots gop_{10}\}$).

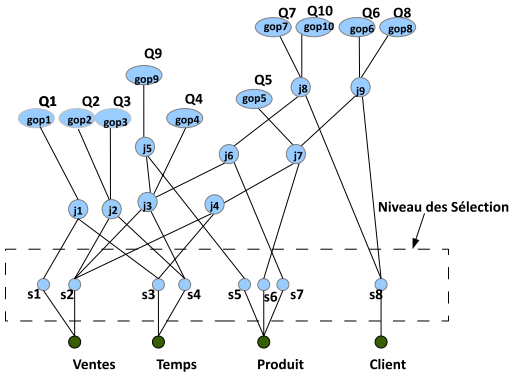

FIG. 1: Exemple de MVPP

Alias	Prédicat	Table
s_1	$\sigma(quantit < 100)$	Ventes
s_2	$\sigma(100 \leq quantit < 1000)$	Ventes
s_3	$\sigma(saison = "Automne")$	Dates
s_4	$\sigma(saison \in \{"Hiver"; "Printemps"\})$	Dates
s_5	$\sigma(type = "T1")$	Produit
s_6	$\sigma(couleur = "Couleur1")$	Produit
s_7	$\sigma(type = "T2")$	Produit
s_8	$\sigma(genre = "Femme")$	Client

TAB. 1: Sélections dans le MVPP

La principale observation qui s'impose après la construction de plans est l'existence de sous expressions communes entre les plans de requêtes comme la sélection s_2 et la jointure j_3. La jointure j_3 entre la table *Ventes* et *Temps* est utilisée simultanément par les requêtes Q_4, Q_7, Q_9 et Q_{10}. Pour satisfaire la requête Q_4 en optimisant les sélections $\{s_2, s_4\}$ impliquées dans la jointure j_3, le coût d'exécution de j_3 doit être considérablement réduit. Ceci aura des conséquences non seulement sur la requête Q_4, mais aussi sur toutes les requêtes qui lui sont corrélées dans ce noeud : Q_7, Q_9 et Q_{10}. Parmi les techniques d'optimisation existantes, celle qui s'adapte le mieux avec ce niveau de noeuds, i.e. les sélections, est la Fragmentation Horizontale.

Propager le gain en performance tout au long des plans de requêtes, tout en respectant le nombre maximal de fragments finaux est une tâche difficile. Pour ce faire, nous exploitons l'interaction entre les requêtes, pour grouper les requêtes en quatre sous-ensembles disjoints selon les jointures communes : $\{Q_1\}; \{Q_2; Q_3\}; \{Q_4; Q_7; Q_9; Q_{10}\}; \{Q_5; Q_6; Q_8\}$.

On observe que le nombre de groupes obtenu est égal au nombre de jointures directes avec la table des faits (Quatre dans l'exemple). La raison est que chaque paire de requêtes dans le même groupe est corrélée au moins dans la première jointure qui est très coûteuse. Si cette première jointure est optimisée par la $\mathcal{FH}$ sur ses sélections, toutes les requêtes du même groupe vont y bénéficier. L'observation de ces faits nous permet de déduire l'impact de telles propriétés sur le déroulement du processus de $\mathcal{FH}$. Dans ce sens, nous proposons une nouvelle approche pour la $\mathcal{FH}$ en exploitant l'interaction des requêtes dans une démarche "diviser pour mieux régner". La structure de données basée sur le MVPP peut être utilisée pour identifier les prédicats de sélection utilisés dans le processus de fragmentation. Dans le cas où un prédicat a un faible ou fort facteur de sélectivité, il peut être éliminé du processus.

4 Background

Dans cette section, nous introduisons d'abord les concepts de base qui faciliteront la compréhension de notre proposition, ainsi que l'algèbre qui gèrera les partitions.

4.1 Décomposition d'un domaine d'attribut en sous-domaines

Le schéma de $\mathcal{FH}$ d'un $\mathcal{ED}$ est basé sur les prédicats de sélection figurant dans la charge. Chaque prédicat est défini comme suit : $Att\ \theta\ Val$, où Att est l'attribut de sélection, $\theta \in \{=, <, >, \leq, \geq, \in \ldots\}$ et Val est une valeur, une liste, ou un intervalle dans le domaine des valeurs de l'attribut Att. La Table 1 contient l'ensemble des prédicats de sélection dans le MVPP de la Figure 1. La $\mathcal{FH}$ permet de décomposer les domaines des attributs en sous-domaines. Les administrateurs expérimentés arrivent à partitionner un entrepôt de données relationnel en proposant des décompositions des domaines des attributs en sous-domaines. Cette $\mathcal{FH}$ ne peut pas être utilisée dans la vraie vie où les attributs de fragmentation peuvent être importants. La Figure 2-a montre un exemple de décomposition de domaines obtenue à partir du MVPP. Les algorithmes actuels permettent d'assurer ces décompositions indépendamment du MVPP. Par conséquent, la décomposition est statique. Dans ce papier, nous proposons un codage incrémental qui fait la décomposition systématique établie à partir du MVPP.

4.2 Schéma de Codage

Souvent, les schémas de $\mathcal{FH}$ sont représentés par un codage fixe qui est une juxtaposition de vecteurs représentant les attributs dans le niveau des sélections (Figure 1), où chaque vecteur représente la décomposition des domaines de chaque attibut de $\mathcal{FH}$. La valeur de chaque cellule d'un vecteur donné représentant un attribut A_i^k appartient à $[1 \cdots n_i]$, où n_i représente le nombre de sous-domaines de l'attribut A_i^k. De cette représentation, le schéma de $\mathcal{FH}$ de chaque table de dimension D_j est généré comme suit :

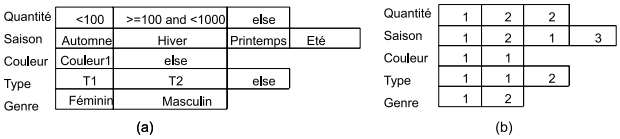

FIG. 2: Exemple de schéma de codage (b) obtenu à partir des attributs de sélection et leur sous-domaines (a)

- Toutes les cellules d'un attribut de fragmentation A_i^k de D_j ont des valeurs différentes : ce qui signifie que tous les sous-domaines seront utilisés pour partitionner D_j. A titre d'exemple, les cellules de chaque attribut de fragmentation dans la Figure 2-(b) sont différents.

- Toutes les cellules d'un attribut de fragmentation A_i^k ont la même valeur : ceci signifie que l'attribut ne participera pas dans le processus de fragmentation.

- Quelques cellules ont la même valeur : leurs sous-domaines seront fusionnés.

Un exemple de schéma de $\mathcal{FH}$ est donné dans la Figure 2-(b).

4.3 Algèbre

Pour éviter le codage statique, nous proposons un codage incrémental conçu à l'arrivée des requêtes du MVPP. Avant de proposer ce codage, la définition de l'algèbre est requise. Cette algèbre a un double rôle : (1) fournir un schéma de codage, et (2) générer un schéma de fragmentation. Dans la majeure partie des SGBD, la $\mathcal{FH}$ est supportée avec des primitives pour gérer les fragments assurées par deux fonctions : *Merge* et *Split*. La première consiste à fusionner deux partitions en une, et la deuxième permet d'éclater une partition en deux.

Ces opérations sont effectuées au niveau physique. Il serait intéressant de les propager au niveau structurel afin de fournir un schéma de codage incrémental. L'effet de ces opérations sur notre codage est de générer de nouveaux sous-domaines en effectuant un *Split* (aussi dit Split Horizontal) d'un sous-domaine ou *Merge* (fusion) de deux sous domaines en un (Figure 3). Plus formellement, les fonctions de fusion et d'éclatement ont les signatures suivantes :

$Merge(sd_i^k, sd_j^k, A^k, PS) \rightarrow PS'$, où la fonction *Merge* est appliquée sur deux sous-domaines sd_i^k et sd_j^k de l'attribut A^k pour les fusionner en une même partition.

$Split(sd_i^k, A^k, PS) \rightarrow PS'$, où la fonction *Split* est appliquée sur le sous-domaine sd_i^k pour le scinder en deux sous domaines si et seulement s'il couvre au moins deux autres sous-domaines de l'attribut A^k.

Une autre opération pouvant être appliquée sur notre codage, dite *Split Vertical*, consiste à ajouter un nouvel attribut de sélection au codage. Initialement, le domaine de l'attribut est composé de deux sous-domaines : le premier est décrit par l'opération de sélection et le second pour le champs *else* afin d'assurer la complétude. Le champ *else* peut être scindé quand de nouveaux prédicats sont considérés (Figure 3).

5 L'Algorithme basée sur la Requête Elue

Dans cette section, nous détaillons notre proposition en plusieurs niveaux. Nous commençons par décrire le codage utilisé et la façon dont nous traitons les attributs de sélection. Puis,

nous présentons comment exploiter l'interaction des requêtes dans le processus de $\mathcal{FH}$ en privilégiant les noeuds de jointure communs. Pour traiter ces noeuds, nous proposant une approche "diviser pour mieux régner" dite : *"Algorithme de Requête Elue"* (ARE) afin d'élaguer l'espace des attributs pertinents et leurs sous-domaines. Nous décrivons comment l'algorithme ARE conduit la $\mathcal{FH}$ et donnons ses différents détails.

5.1 Codage Incrémental

L'un des problèmes majeurs de la $\mathcal{FH}$ est la représentation des attributs de sélection et leurs sous-domaines pour le processus d'optimisation. Notons que notre codage concerne uniquement les attributs qui figurent dans le MVPP. La construction du codage est faite de manière incrémentale en explorant les plans de requêtes un à un, et en adaptant le codage avec les nouveaux attributs et sous-domaines. L'extraction de l'ensemble des attributs de sélection (S) se fait à partir du MVPP, et le codage se construit à l'aide de l'algèbre décrite dans la Section 4.3.

Le principe de ce codage est de commencer par un ensemble d'attributs vide, et pour chaque requête y ajouter ses attributs de sélection en créant de nouveaux vecteurs, chacun contenant un seul champs. A chaque fois qu'une sélection est retrouvée dans la requête en cours, l'une des trois opérations est effectuée :

1. Si l'attribut n'existe pas dans le schéma, appliquer un Split Vertical pour étendre le schéma en construisant un nouvel attribut. Le champ est scindé en plusieurs parties pour couvrir les sous-domaines retrouvés. Un champ *else* est rajouté afin d'assurer la complétude. La Figure 3 illustre l'ajout d'attributs dans l'ensemble initialement vide S par la requête Q_1 (*quantité, saison*).
2. Si l'attribut existe déjà, appliquer le Split Horizontal sur le champ *else* pour rajouter les nouveaux sous-domaines. La Figure 3 montre un exemple de sous-domaines rajoutés par la requête Q_2 pour l'attribut *saison*.
3. Finalement, si l'administrateur connaît la valeur restante dans les champs *else*, il la remplace par cette valeur. Dans la Figure 3, le *else* est remplacé par *"Masculin"* pour l'attribut *genre*, et par *"Eté"* pour l'attribut *saison*.

A la fin de cette phase, un schéma de codage est généré contenant l'ensemble des attributs de sélection et les sous-domaines associés. Pour dérouler la procédure de codage sur notre exemple, nous commençons par un ensemble vide d'attributs. Quand la première requête Q_1 arrive, deux nouveaux attributs et leurs sous-domaines de Q_1 sont rajoutés. La requête Q_2 ne contient pas de nouveaux attributs mais de nouveaux prédicats qui déclenchent un Split Horizontal sur le schéma existant (Figure 3). Finalement, quand la dernière requête Q_{10} est traitée, le schéma est ajusté avec les valeurs connues remplaçant les *else* (Figure 3).

5.2 Privilègier les Noeuds de Jointures avec Grande Interaction

A partir de l'ensemble des attributs candidats obtenu, un schéma de fragmentation doit être généré afin de réduire le plus possible le coût d'exécution. Comme nous l'avons déjà mentionné, les solutions existantes sont soit (1) simplistes, soit (2) utilisent des heuristiques basées sur des modèles de coût simplifiés. Nous proposons de rajouter un nouveau critère pour conduire le processus de $\mathcal{FH}$, pour parvenir à un compromis entre les deux approches, avec une efficacité meilleure et une complexité réduite.

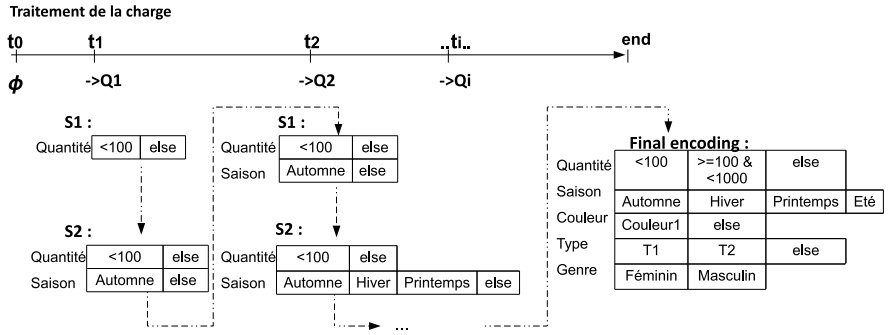

FIG. 3: Codage Incrémental par Split Horizontal et Vertical sur les sous-domaines et les attributs

Comme nous avons déjà vu dans l'exemple de motivation, l'opération de jointure est très coûteuse, et en même temps, les requêtes en interaction partagent au moins la première jointure. La considération du noeud de première jointure est cruciale car il servira toutes les requêtes partageant ce noeud. Pour capturer et exploiter l'interaction des requêtes et réduire la complexité des algorithmes de résolution, nous proposons un algorithme nommé *"Algorithme de Requête Elue"* (ARE). A partir du MVPP, l'ARE génère l'ensemble des groupes de requêtes corrélées, tel que chaque couple de requêtes ayant au moins un noeud de jointure commun est dans le même groupe. La Figure 4 montre l'ensemble des groupes obtenus à partir du MVPP de l'exemple. Dans chaque groupe, l'algorithme élit une requête qui permettra d'aiguiller le processus de $\mathcal{FH}$. Cette requête, que l'on appelle "*Requête Elue*" (RE) est considérée comme la plus importante dans son groupe. Le choix de la requête élue RE_i du groupe i est effectué selon le critère du coût d'exécution. Dans chaque groupe, la requête ayant le coût minimal est considérée comme l'élue. Généralement, la requête ayant un coût minimal contient moins d'opérations de jointure et de sélections. En conséquence, elle peut contribuer à l'élagage des prédicats de sélection. Le nombre de noeuds communs n'est pas considéré dans le choix de la requête élue car il existe au moins un noeud partagé entre cette requête et les requêtes du même groupe.

5.3 Effet de l'ARE sur le Codage : Elagage de l'Espace de Recherche

L'ARE vise à fournir un sous ensemble d'attributs de sélection et de sous-domaines pertinents pour aiguiller la $\mathcal{FH}$. L'ensemble de requêtes élues retourné permet d'élaguer l'espace de recherche et d'éliminer les possibilités les moins efficaces comme les sélections et les jointures les moins fréquentes.

Les attributs obtenus à partir du MVPP de la phase précédente, seront élagués encore une fois en considérant dans un premier temps, seulement les attributs requis par les requêtes élues. Dans le processus de fragmentation, l'algorithme satisfait les requêtes élues au lieu de choisir aléatoirement les prédicats. Cette phase d'élagage réduit considérablement la complexité de l'algorithme en réduisant la taille de l'espace de recherche. Les détails de l'algorithme de $\mathcal{FH}$ en utilisant ARE sont présentés dans la section suivante.

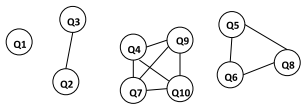

FIG. 4: Generation des groupes de requêtes en interaction à partir du MVPP

5.4 Exploiter l'ARE dans la Fragmentation Horizontale

L'algorithme de $\mathcal{FH}$ que nous proposons, dit "*Requêtes Elues pour la Fragmentation Horizontale*" (REFH), commence à partir du MVPP, et effectue l'élagage de l'ensemble des attributs de sélection selon les requêtes apportant le plus de gain en utilisant ARE.

L'ARE regroupe les requêtes en sous ensembles disjoints, puis trie les requêtes à l'intérieur par coût minimal pour retourner l'ensemble de requêtes élues RE_i de chaque groupe i. L'ensemble des requêtes élues obtenu par ARE est trié ensuite par coût maximal. Ceci permet de commencer par optimiser les requêtes les plus coûteuses avant d'épuiser le nombre de fragments possible $\mathcal{W}$. L'ARE traite le premier ensemble de requêtes élues et effectue l'élagage selon les besoins de ses requêtes, i.e. seuls les attributs requis par les requêtes élues sont pris, les autres sont ignorés. Chaque sous-domaine est indexé par le nombre de requêtes élues qui l'utilise.

Soit u_{ij} le nombre de RE utilisant le sous-domaine. Et soit k_i la valeur maximale de u_{ij} pour l'attribut a_i. L'ensemble d'attributs est trié par usage maximal (valeur de k_i) afin de commencer le fragmentation par les attributs les plus utilisés avant que $\mathcal{W}$ ne soit épuisé. Après avoir trié les attributs, chaque attribut a_i subit des opérations de fusion/éclatement (Merge/Split) comme suit :

1. Les sous-domaines qui ne sont utilisés par aucune requête élue ($u_{ij} = 0$) sont regroupés en une même partition P_0 ;

2. Les sous-domaines les plus utilisés, ayant k_i requêtes élues y accédant, sont regroupés en une seule partition P_k (k_i est la valeur maximale d'usage de l'attribut a_i) ;

3. Finalement, si $N > W$ ou $k_i \neq 0$ alors, les sous-domaines restant sont fusionnés avec la partition P_0 ; sinon, les sous-domaines accédés par $(k_i - 1)$ requêtes élues sont regroupés dans une nouvelle partition. La même opération est répétée jusqu'à ce que $k_i = 0$ ou $N > W$. Ceci permet de créer des partitions en se basant sur les besoins des requêtes les plus importantes. Si le fragmentation est toujours possible ($N < W$), les partitions obtenues sont éclatées selon la corrélation entre les requêtes accédant chaque sous-domaine de la partition. Si deux sous-ensembles de requêtes nécessitent quelques sous-domaines indépendamment, alors une nouvelle partition est créée contenant les sous-domaines utilisés indépendamment des autres.

Si $N < W$ après ces opérations de fusion et d'éclatement en considérant seulement les requêtes élues, alors le fragmentation reste possible. Pour cette raison, le processus d'optimisation enchaîne avec un nouvel ensemble de requêtes élues à satisfaire.

L'ensemble suivant des requêtes est celui des successeurs des RE courantes. Si au moins un groupe contient encore un successeur (parmi les requêtes triées par coût minimal), alors un nouvel ensemble de requêtes élues est généré par les successeurs retrouvés de tous les groupes. Le même processus est appliqué pour étendre le schéma de codage avec les nouveaux attributs et sous-domaines requis de façon incrémentale. Le processus réitère jusqu'à ce que $N < W$ ou qu'aucune requête ne reste dans **aucun** groupe.

6 Etude Expérimentale

Dans cette section, nous conduisons une étude expérimentale intensive validant notre proposition. Nous commençons par présenter notre jeu de données (le schéma de l'entrepôt, la charge de requêtes, l'environnement ...), ainsi que notre simulateur connecté à Oracle11g. Les résultats obtenus sont discutés et justifiés pour révéler les différentes particularités de notre approche.

Dans notre étude expérimentale, nous travaillons sur le benchmark SSB [1] avec un facteur d'échelle de 100, afin d'obtenir 100 GB de données. L'entrepôt contient une table de faits *Lineorder* (600 000 000 instances), et quatre tables de dimension : *Customer (3 000 000 instances), Supplier (200 000 instances), Part (600 000 instances)* et *Dates* (2 556 instances) stockées sous Oracle11g sur un serveur de 32GB de RAM et un processeur Intel® Xeon® CPU de 2x2.40GHz. La charge ayant 22 requêtes de jointures en étoile s'exécutant sous Oracle11g.

Pour effectuer les différents tests, nous avons développé un simulateur connecté à un SGBD Oracle11g. Ses entrées sont l'entrepôt et sa charge de requêtes. Le seuil $\mathcal{W}$ est varié afin de montrer son impact sur la performance. Notre modèle de coût fournit le nombre d'Entrées/Sorties requis par chaque requête pour mesurer la qualité des schémas. Les algorithmes utilisés sont : REFH et le Recuit Simulé (RS). Le choix de cet algorithme repose sur le fait qu'il est largement utilisé dans ce genre de problèmes d'optimisation. De plus, il est moins gourmand qu'un algorithme génétique Bellatreche et al. (2009). Pour paramétrer notre RS, nous avons fixé le nombre d'itération à 500 et la température à 300. Comme l'algorithme est non déterministe, il est exécuté plusieurs fois (entre 5 et 10 selon la variance) pour prendre une performance moyenne. Contrairement à notre algorithme qui génère son propre codage, le RS nécessite un codage pré-établi comme entrée additionnelle pour s'exécuter dessus.

Coût d'exécution pour des Seuils variables Dans la première expérience, nous comparons les schémas de $\mathcal{FH}$ retournés par le RS et notre REFH. Dans la Figure 5, le coût d'exécution de la charge est donné en terme d'E/S. Les résultats montrent que l'algorithme REFH est plus efficace que le RS sauf pour les valeurs de seuil très grandes ($\mathcal{W} \geq 300$), avec des différences restreintes en performance. La raison de l'efficacité de REFH est qu'il effectue les opérations Split/Merge à la demande, i.e. il vise à satisfaire le plus grand nombre de requêtes parmi les plus efficaces (élues) qui contribuent considérablement à l'amélioration de performance de la charge.

Quand $\mathcal{W} < 300$, le RS effectue un nombre d'itération assez grand en cherchant aléatoirement la meilleure solution, alors que le REFH détecte et effectue systématiquement les mouvements nécessaires (Split/Merge) pour obtenir le schéma de fragmentation bénéfique.

Si $\mathcal{W}$ est très grand (≥ 300), le REFH s'arrête lorsque toutes les requêtes élues et leurs prédicats sont satisfaits. Cependant, plusieurs opérations de Split/Merge restent encore possibles pour réduire le coût de la charge. Ces opérations additionnelles ne peuvent être effectuées par le REFH, mais peuvent être atteintes aléatoirement par le RS.

Nous rappelons que $\mathcal{W}$ est le nombre de fragments de faits, qui représente le nombre de sous-schémas générés. Le nombre total de fragments dans le schéma est plus grand que $\mathcal{W}$.

1. http://www.cs.umb.edu/~poneil/StarSchemaB.pdf

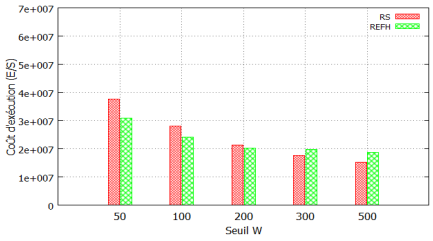

FIG. 5: Comparaison de performance des algorithmes en variant le Seuil

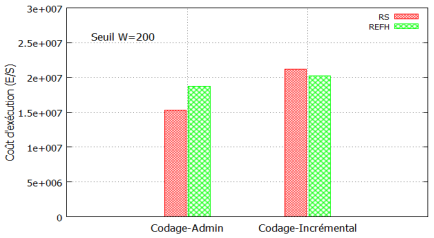

FIG. 6: Sensibilité des deux algorithmes au codage

Nous concluons de cette expérience que le REFH peut atteindre un niveau élevé d'efficacité comparé aux heuristiques. Cependant, pour les valeurs de Seuil très grandes $\mathcal{W}$, le RS est plus efficace avec des différences restreintes.

Sensibilité au codage Un autre avantage majeur de notre approche est que l'algorithme génère son propre codage. Ce qui signifie que la phase de codage est une partie entière de l'algorithme et fournit un schéma de codage dépendant de la charge, contrairement au RS qui nécessite un codage pré-établi pour son déroulement. Le schéma donné au RS est généralement fourni par l'administrateur, et peut contenir les répartitions de sous-domaines meilleures pour les attributs de sélection.

Afin d'étudier la sensibilité des algorithmes de fragmentation au codage, nous avons effectué deux expériences : (1) La première en utilisant le processus de codage incrémental de REFH. Le codage obtenu est fourni au RS pour s'exécuter dessus. (2) La seconde en utilisant le codage de l'administrateur dans les deux algorithmes. La Figure 6 montre que l'utilisation du codage de l'administrateur rend le RS plus efficace, car il fournit une décomposition des sous-domaines plus appropriée s'il connaît sa charge et ses données. Cependant, ce codage pénalise le REFH car la décomposition peut ne pas convenir aux besoins de certaines requêtes.

A partir de cette expérience, nous concluons que les algorithmes de résolution de la fragmentation horizontale sont en effet sensibles au codage. Le RS ne peut générer son propre codage et fait appel à l'administrateur pour le lui fournir. Cependant, dans la vraie vie, l'administrateur peut ne pas connaître la meilleure décomposition des sous-domaines, ce qui rend le REFH plus efficace de par son autonomie.

Coût d'exécution en variant l'interaction des requêtes Afin d'étudier l'impact de l'interaction dans notre approche, nous utilisons une nouvelle charge de requêtes n'ayant aucune jointure commune entre-elles. La Figure 7 montre le coût d'exécution de la nouvelle charge optimisée par les deux algorithmes. Les résultats montrent que le REFH n'est pas suffisamment efficace comparé au RS dans le cas où l'interaction n'existe pas entre les requêtes. Ceci est dû au fait que seul un sous-ensemble de requêtes est optimisé, mais le gain n'a pu se propager parmi les autres.

Nous comparons maintenant les performances de REFH avec et sans interaction. La Figure 8 montre que le REFH est meilleur sur une charge corrélée (en interaction), mais son efficacité est limitée sans interaction entre les requêtes.

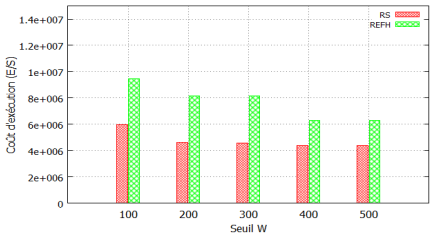

FIG. 7: Rendement du RS et REFH sur une charge sans jointures communes

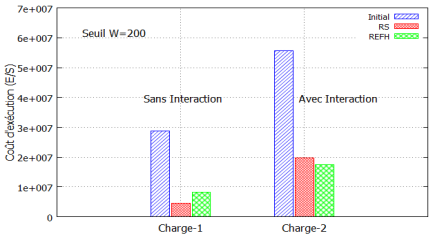

FIG. 8: L'impact de l'interaction entre les requêtes sur la performance

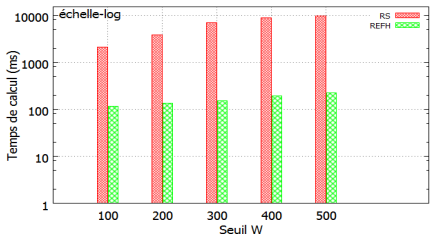

FIG. 9: Comparaison de la réactivité du RS et de l'algorithme de REFH

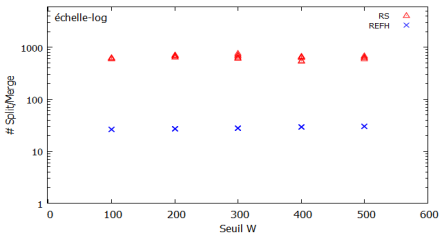

FIG. 10: Nombre de mouvements Split et Merge effectués par chaque algorithme

A partir de ces expérimentations, nous montrons que notre approche est plus efficace dans le cas où la charge est corrélée, car le processus d'optimisation de notre approche est guidé principalement par l'interaction.

Réactivité Dans l'expérience qui suit, nous étudions la réactivité des algorithmes. La Figure 9 montre le temps de réponse des algorithmes. En variant le Seuil $\mathcal{W}$, le temps de réponse du RS augmente considérablement, contrairement au REFH qui reste relativement faible.

Ce temps de réponse est courant dans cette famille d'algorithmes, contrairement à celui de notre algorithme qui est très rapide. Ceci est dû aux phases d'élagage de l'espace de recherche et au processus de fragmentation guidé par l'interaction.

Dans l'expérience qui suit, nous allons encore étudier la rapidité des algorithmes, mais en terme de mouvements et d'opérations de Split et/ou Merge. La Figure 10 montre le nombre d'opérations Split/Merge effectuées par les algorithmes à différentes valeurs de $\mathcal{W}$.

Comme le RS est non déterministe, il est exécuté à mainte reprises, et le nombre d'opérations (split/merge) varie pour chaque valeur de $\mathcal{W}$. D'autre part, le REFH est déterministe et une solution unique est fournie pour une valeur $\mathcal{W}$ donnée, ainsi, un nombre fixe de mouvements (split/merge) est effectué.

Comme le montre la Figure 10, le RS est très gourmand en terme de mouvements et d'opération d'éclatement et de fusion comparé au REFH pour les mêmes raisons que celles de l'expérience précédente ; i.e. notre approche est orientée-interaction contrairement aux heuristiques qui explorent un espace très large de manière aléatoire.

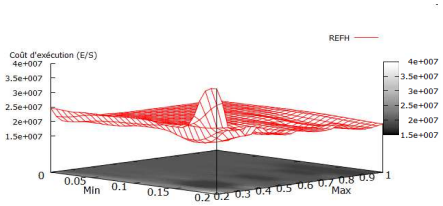

FIG. 11: Les intervalles de Facteurs de Sélectivité pour le fragmentation

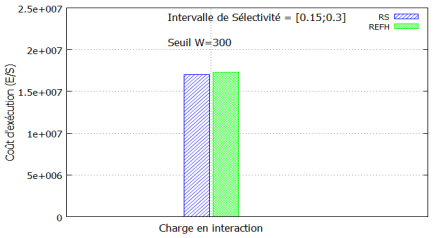

FIG. 12: Amélioration du REFH en définissons les intervalles de sélectivité

A partir de cette dernière expérience, nous avons montré que notre approche reste plus rapide que le RS notamment quand le $\mathcal{W}$ est très grand ce qui rend le temps d'exécution de l'algorithme en croissance considérable.

La sélectivité des prédicats Quelques partitions pertinentes pour la REFH ne conviennent pas au processus de fragmentation car leurs facteurs de sélectivité sont soit trop élevés soit trop faibles. De tels sélectivités génèrent des partitions non homogènes (trop larges ou trop petites) ce qui pénalise certaines requêtes. Pour éviter cette situation, nous étendons notre algorithme en ajoutant un nouveau critère d'élagage qui est l'intervalle de sélectivité.

Nous étudions les facteurs de sélectivité pour notre charge en variant leurs valeurs minimales et maximales. Les résultats sont présentés dans la Figure 11. En effet, les valeurs élevées (≈ 1) ou faibles (≈ 0) pénalisent la charge. Cependant, la meilleure performance est obtenue dans l'interval $[0.15; 0.30]$ qui représente les facteurs de sélectivité des prédicats le plus adaptés pour le fragmentation. Nous comparons le REFH amélioré avec le RS dans l'expérience illustrée dans la Figure 12, qui montre que le rendement du REFH est amélioré avec le nouveau critère même avec des valeurs plus grandes de Seuil ($\mathcal{W} = 300$) et devient plus proche que celui du RS.

Nous concluons que notre algorithme est déterministe, capable de générer son propre codage, rapide et permet d'atteindre de hauts niveaux de performance comparables à ceux des méta-heuristiques en se basant sur l'interaction des requêtes.

Validation sous Oracle11G Dans le but de valider nos résultats de simulation, nous déployons les solutions obtenues sur un SGBD Oracle11g. Le même jeu de données est utilisé, i.e. schéma SSB (100G) avec 22 requêtes sur notre serveur. Les résultats dans la Figure 13 donnent les mêmes performances que celles obtenues par la simulation. Notre algorithme est donc plus efficace que le RS avec un seuil W=100 dans un système réel. Dans une autre expérience, nous regroupons les requêtes par nombre de jointures. La Figure 14 montre le temps d'exécution requis par classe (1-J à 4-J). Nous constatons que notre algorithme apporte moins de gain aux requêtes ayant un grand nombre de jointures (ex. 4-J), car il commence par satisfaire les requêtes les moins coûteuses, ce qui est généralement le cas des requêtes ayant peu de jointures. D'autre part, nous constatons aussi que notre approche permet de propager le de plus gains à travers la charge.

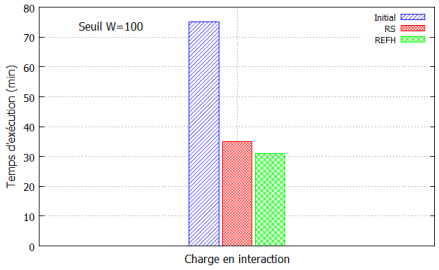

FIG. 13: Validation des résultats de simulation sous Oracle11g

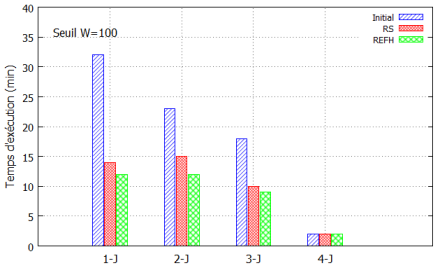

FIG. 14: Gain en performance par classes de requêtes sous Oracle11g

7 Conclusion et Perspectives

Dans cet article, nous avons identifié une piste de recherche intéressante caractérisée par la prise en compte de l'interaction de requêtes dans la conception physique des entrepôts de données relationnels. Nous avons considéré le problème de fragmentation horizontale. Notre état de l'art sur les travaux existants sur ce problème a montré l'ignorance de cette interaction. Nous avons alors proposé un algorithme basé sur une structure de données incrémentale construite à partir du graphe unifiant les arbres algébriques de requêtes considérées dans le processus de fragmentation. Cet algorithme considère les prédicats de requêtes ayant une forte interaction avec les autres. Une étude expérimentale a été menée en utilisant un simulateur connecté à un SGBD Oracle11g et les résultats obtenus par cette simulation sont validées directement sur Oracle11g. Les résultats obtenus sont prometteurs et montrent l'intérêt de nos algorithmes.

Nous sommes entrain d'adapter nos propositions pour le processus de sélection des index de jointure binaires. Une piste intéressante est utiliser nos algorithmes pour déployer un entrepôt de données sur des systèmes répartis.

Références

Ahmad, M., A. Aboulnaga, S. Babu, et K. Munagala (2011). Interaction-aware scheduling of report-generation workloads. *VLDB Journal 20*(4), 589–615.

Bellatreche, L., K. Boukhalfa, et P. Richard (2009). Referential horizontal partitioning selection problem in data warehouses : Hardness study and selection algorithms. *International Journal of Data Warehousing and Mining 5*(4), 1–23.

Ceri, S., M. Negri, et G. Pelagatti (1982). Horizontal data partitioning in database design. In *SIGMOD*, pp. 128–136. ACM.

Corporation, O. (2010). Partitioning with oracle database 11g release 2. *An oracle White Paper. Retrieved from www.oracle.com/technetwork/ middleware /bi-foundation/twp-partitioning-11gr2-2009-09-130569.pdf*.

Karlapalem, K. (1996). Redesign of distributed relational databases. Phd. thesis, Georgia Institute of Technology.

Karlapalem, K., S. B. Navathe, et M. Ammar (1996). Optimal redesign policies to support dynamic processing of applications on a distributed database system. *Information Systems 21*(4), 353–367.

Le, W., A. Kementsietsidis, S. Duan, et F. Li (2012). Scalable multi-query optimization for sparql. In *ICDE*, pp. 666–677. IEEE.

Mahboubi, H. et J. Darmont (2009). Enhancing xml data warehouse query performance by fragmentation. In *SAC*, pp. 1555–1562. ACM.

Navathe, S. B. et M. Ra (1989). Vertical partitioning for database design : a graphical algorithm. In *SIGMOD*, pp. 440–450. ACM.

O'Gorman, K., D. Agrawal, et A. El Abbadi (2002). Multiple query optimization by cache-aware middleware using query teamwork. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pp. 274.

Özsu, M. T. et P. Valduriez (1999). *Principles of Distributed Database Systems : Second Edition*. Prentice Hall.

Papadomanolakis, S. et A. Ailamaki (2004). Autopart : Automating schema design for large scientific databases using data partitioning. In *SSDBM*, pp. 383–392. IEEE.

Phan, T. et W.-S. Li (2008). Dynamic materialization of query views for data warehouse workloads. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pp. 436–445.

Sanjay, A., V. R. Narasayya, et B. Yang (2004). Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 359–370.

Sellis, T. K. (1988). Multiple-query optimization. *ACM Transactions on Database Systems 13*(1), 23–52.

Yang, J., K. Karlapalem, et Q. Li (1997). Algorithms for materialized view design in data warehousing environment. In *Proceedings of the International Conference on Very Large Databases*, pp. 136–145.

Zilio, D. C., J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. Garcia-Arellano, et S. Fadden (2004). Db2 design advisor : Integrated automatic physical database design. In *Proceedings of the International Conference on Very Large Databases*, pp. 1087–1097.

Summary

The queries running in relational data warehouses ($\mathcal{RDW}$) interact with each other. This interaction has been largely exploited to optimize queries in traditional databases. In the context of the $\mathcal{RDW}$, it has been also used to select materialized views. In this paper, we revisit the problem of selecting horizontal partitioning. Firstly, we propose a rich state of art, where we show the ignorance of most existing research studies of the query interaction. Secondly, a new algorithm is proposed based on the query interaction. Finally, intensive experiments were conducted to show the efficiency of our proposal.