Index de Jointure Binaires: Stratégies de Sélection & Étude de Performances

Kamel Boukhalfa *, Ladjel Bellatreche** Benameur Ziani ***

* USTHB - Bp 32 El-Alia Alger, Algérie
boukhalk@ensma.fr
** LISI/ENSMA - Université de Poitiers
Futuroscope 86960 France
bellatreche@ensma.fr
*** Université de Laghouat - Algérie
bziani@mail.lagh-univ.dz

Résumé. La conception physique des entrepôts de données relationnels est basée essentiellement sur la sélection d'un ensemble d'index afin de réduire le coût d'exécution des requêtes OLAP complexes. Ces entrepôts sont généralement modélisés par un schéma en étoile caractérisé par une table de faits volumineuse et un ensemble de tables de dimension liées à la table des faits par leurs clés étrangères. Les requêtes définies sur ce schéma (appelées requêtes de jointure en étoile) comportent plusieurs jointures entre la tables des faits et les tables de dimension ce qui rend leur coût d'exécution considérable. Les index de jointure binaires sont très adaptés pour réduire le coût d'exécution de ces jointures. Ils sont défini sur la table de faits en utilisant un ou plusieurs attributs de dimension. Sélectionner une configuration d'index pour réduire le coût d'exécution d'un ensemble de requêtes est reconnu comme un problème NP-Complet. Dans ce papier, nous présentons d'abord le problème de sélection des index de jointure binaires et les principaux travaux effectués dans ce domaine. Nous présentons par la suite notre approche de sélection et les algorithmes que nous proposons. Nous effectuons des expériences pour comparer les différentes stratégies de sélection. Enfin, nous effectuons une validation réelle des différents algorithmes sous Oracle en utilisant les données issues du banc d'essai APB1.

1 Introduction

Les entrepôts de données stockent un volume de données très important essentiellement dans des modèles logiques relationnels comme les schémas en étoile ou flocon de neige (Kimball et Strehlo, 1995). Ces derniers sont accédés par des requêtes décisionnelles complexes caractérisées par de multiples opérations de sélection, de jointure et d'agrégation. Les requêtes définies sur un schéma en étoile sont appelées *requêtes de jointure en étoile* comportant un nombre de prédicats de sélection définis sur des tables de dimension et des prédicats de jointures entre la table des faits et les tables de dimension. Pour optimiser les sélections et les

jointures, on a un recours à une des premières structures d'optimisation proposées dans le contexte des bases de données : *les index*. Ces derniers ont en effet montré leurs performances dans les bases de données traditionnelles. On peut citer les arbres B et leurs variantes (Comer, 1979). Un index peut être défini sur une seule colonne d'une table, ou sur plusieurs colonnes d'une même table. Il pourra être clustérisé ou non clustérisé. Il peut être également défini sur plusieurs tables comme les index de jointure (Valduriez, 1987).

Avec l'arrivée des entrepôts de données, l'indexation constitue une option importante dans la phase de conception physique (Chaudhuri et Narasayya, 2007). Cependant, les techniques d'indexation utilisées dans les bases de données de type (OLTP) ne sont pas réellement adaptées aux environnements des entrepôts des données. En effet la plupart des transactions OLTP accèdent à un petit nombre de n-uplets, et les techniques utilisées (index B^+ par exemple) sont adaptées à ce type de situation. Les requêtes décisionnelles adressées à un entrepôt de données accèdent au contraire à un très grand nombre de n-uplets. Réutiliser les techniques des systèmes OLTP conduirait à des index avec un grand nombre de niveaux qui ne seraient donc pas très efficaces (O'Neil et Quass, 1997; Informix-Corporation, 1997). Plusieurs techniques d'indexation ont été proposées dans le cadre des entrepôts de données relationnels, certaines héritées des bases de données traditionnelles comme les *arbres B*. D'autres, proposées pour optimiser la sélection définie sur des attributs de faible cardinalité comme les *index binaire* (Chan et Ioannidis, 1998). Dans le souci d'optimiser simultanément la sélection et la jointure, les *index de jointure binaire* ont été proposés et supportés par la plupart des SGBD commerciaux. Ils sont bien adaptés pour optimiser les requêtes de jointure en étoile (Stöhr et al., 2000). Un index de jointure binaire (IJB) peut être défini sur une table des faits en utilisant un seul attribut (dans ce cas, appelé un *IJB mono-attribut*) ou plusieurs attributs appartenant à une ou plusieurs tables de dimension (appelé, *IJB multi-attributs*). Par conséquent le nombre d'IJB candidat peut être très important ce qui rend la tâche de sélection très compliquée.

En examinant les principaux travaux sur les IJB, nous avons identifié deux tendances : (i) des travaux concentrés sur leur structuration physique afin d'optimiser leur gestion et satisfaire d'autres types de requêtes (Canahuate et al., 2009; Chmiel et al., 2009). Les auteurs dans (Canahuate et al., 2009) ont proposé des techniques de fragmentation horizontale et verticale pour faciliter la gestion des IJB volumineux. Dans (Chmiel et al., 2009), un index binaire hiérarchisé, appelée *HOBI* (*Hierarchically Organized Bitmap Index for Indexing Dimensional Data*) est proposé. Il permet d'optimiser les requêtes de jointure définies sur des hiérarchies importantes des tables de dimension. HOBI est composé de plusieurs index binaires hiérarchiquement organisé dont un pour un niveau de dimension. La deuxième tendance concerne spécifiquement la sélection des IJB. A notre connaissance, seuls deux travaux existent permettant de sélectionner un ensemble d'IJB optimisant un ensemble de requêtes et satisfaisant une contrainte d'espace de stockage (Aouiche et al., 2005; Bellatreche et al., 2008). Ces travaux de sélection utilisent des techniques de fouille de données.

Le nombre limité d'algorithmes de sélection des IBJ existants, nous a motivé pour en proposer de nouveaux. Cet ensemble d'algorithmes de sélection, offre ainsi aux concepteurs une panoplie de choix de sélection plus large. La particularité de nos algorithmes est qu'ils sélectionnent à la fois des *IJB mono-attribut* et des *multi-attributs*.

L'article est organisé comme suit. La section 2 présente les IJB, leur construction et utilisation à travers un exemple. Dans la section 3 nous formalisons le problème de sélection d'IJB et nous montrons sa complexité. Nous présentons ensuite un aperçu des travaux effectués pour le

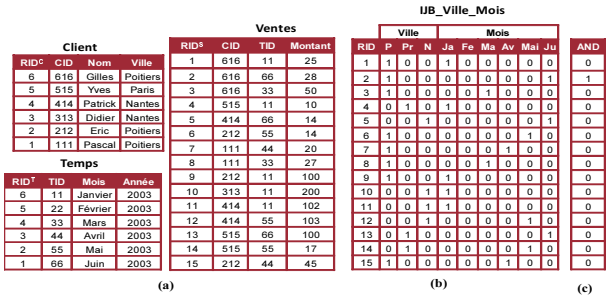

FIG. 1 – *Index de jointure binaire*

résoudre. La section 4 est consacrée à la présentation de notre approche de sélection des IJB. La section 5 présente un ensemble d'expériences que nous avons effectué. La section 6 conclut le papier.

2 Les index de jointure binaires

L'IJB sert à précalculer les jointures entre une ou plusieurs tables de dimension et la table des faits dans les entrepôts de données modélisés par un schéma en étoile (O'Neil et Graefe, 1995; O'Neil et Quass, 1997). Au contraire des index binaires standards où les attributs indexés appartiennent à la même table, l'IJB est défini sur un ou plusieurs attributs appartenant à plusieurs tables. Plus précisément, il est défini sur la table des faits en utilisant des attributs appartenant à une ou plusieurs tables de dimension.

Pour montrer comment un IJB est construit, supposons un attribut A ayant n valeurs distinctes $v_1, v_2, \cdots, v_n$ appartenant à une table de dimension D. Supposons que la table des faits F est composée de m instances. La construction de l'index de jointure binaire défini sur l'attribut A se fait de la manière suivante :

1. Créer n vecteurs composés chacun de m entrées ;
2. Le $i^{ème}$ bit du vecteur correspondant à une valeur v_k est mis à 1 si le n-uplet de rang i de la table des faits est joint avec un n-uplet de la table de dimension D tel que la valeur de A de ce n-uplet est égale à v_k. Il est mis à 0 dans le cas contraire.

La nature binaire des IJB permet d'améliorer les performances des requêtes en permettant d'appliquer des opérations logiques AND, OR, NOT, etc. Ces opérations permettent de rechercher des n-uplets vérifiant des conjonctions ou des disjonctions de prédicats. Les IJB sont très bénéfiques pour les requêtes de type *Count(*)* où l'accès à l'index binaire seulement permet de répondre à ces requêtes.

Notons que la taille d'un IJB est proportionnelle à la cardinalité des attributs indexés (nombre de valeurs distinctes). Pour cette raison, il sont souvent recommandés pour les attributs de faible cardinalité. Pour réduire la taille d'un index binaire, plusieurs techniques de compression ont été proposées (Lemire et al., 2010; Wu et al., 2010).

Pour comprendre l'utilisation des IJB, nous supposons l'exemple suivant.

Exemple 1 *Soit l'entrepôt de données composé d'une table de faits ventes et deux tables de dimension Client et Temps représenté dans la figure 1(a). Soit la requête $Q1$ suivante :*

```
Select count(*)
from Ventes V,Client C, Temps T
where V.CID=C.CID AND V.TID=T.TID
AND C.Ville='Poitiers' AND T.Mois='Juin'.
```

Pour réduire le coût de $Q1$ l'administrateur créé un IJB multi-attributs sur $Ville$ et $Mois$ comme suit :

```
CREATE BITMAP INDEX IJB_Ville_Mois
ON Ventes(Client.Ville, Temps.Mois)
FROM Ventes V, Client C, Temps T
WHERE V.CID= C.CID AND V.TID=T.TID
```

Cet index est représentée dans la figure 1(b). Pour répondre à $Q1$, l'optimiseur lit les vecteurs de bits associés aux valeurs "Poitiers" et "Juin", effectue une opération AND et calcule le nombre de "1" dans le vecteur résultats (Voir figure 1(c)).

3 Sélection des index de jointure binaires

Nous présentons dans cette section une formalisation du problème de sélection des IJB, ensuite nous étudions sa complexité et nous présentons quelques travaux effectués pour le résoudre.

3.1 Formalisation

La sélection d'une configuration d'IJB vise à optimiser la performance d'un ensemble de requêtes de jointure en étoile. Nous considérons l'espace de stockage réservé aux index comme une contrainte du problème de sélection des IJB. En conséquence, le problème de sélection d'une configuration d'IJB peut être formalisé comme suit :
Étant donné : (1) un entrepôt de données modélisé par un schéma en étoile ayant un ensemble de tables de dimension $D = \{D_1, D_2, ..., D_d\}$ et une table des faits F, (2) une charge de requêtes $Q = \{Q_1, Q_2, ..., Q_m\}$, où chaque requête Q_j a une fréquence d'accès f_j, et (3) une capacité de stockage S ;
Le problème de sélection des index de jointure consiste à trouver une configuration d'index CI minimisant le coût d'exécution de Q en respectant la contrainte de stockage ($Taille(CI) \leq S$). Notons que le coût d'exécution d'une requête Q_i est exprimé en nombre d'entrées-sorties nécessaires pour l'exécuter. Le coût d'exécution total de la charge de requête Q est donné par : $COUT(Q) = \sum_{i=1}^{m} Freq_i \times C(Q_i)$ où m, $Freq_i$, $C(Q_i)$ représentent respectivement le nombre de requêtes, la fréquence de la requête Q_i et le coût d'exécution de la requête Q_i.

3.2 Complexité

La sélection d'une configuration d'IJB est généralement une tâche difficile comparée à d'autres types d'index. Cela est dû à plusieurs considérations :

– Un IJB est défini sur un ensemble d'*attributs indexables* qui représentent un sous-ensemble des attributs non clés des tables de dimension. Un attribut est indexable s'il est utilisé

par un prédicat de sélection[1]. Dans le contexte d'entrepôts de données, le nombre d'attributs indexables peut être important, vu le nombre de tables de dimension et le nombre d'attributs non clés de chaque table.
– Un IJB peut être défini sur un ou plusieurs attributs issus de différentes tables de dimension, ce qui augmente le nombre d'IJB possibles.
– Un attribut indexable peut figurer dans plusieurs IJB car ces derniers peuvent être non disjoints.
– La taille d'un IJB dépend de la cardinalité des attributs indexés. Un attribut de forte cardinalité rend l'index volumineux, donc difficile à stocker et à maintenir.

Soit $A = \{A_1, A_2, ..., A_K\}$ un ensemble d'attributs indexables candidats pour la sélection d'une configuration d'IJB. Chaque IJB dans cette configuration est constitué d'un sous-ensemble d'attributs de A, par conséquent cette configuration constitue une partition de A en un ensemble de groupes. Chaque groupe d'attributs représente un IJB potentiel. Nous pouvons considérer deux scenarii :

1. *Sélection d'un seul index de jointure* : si on veut utiliser un seul IJB, le nombre de possibilité est donné par :

$$N = \binom{K}{1} + \binom{K}{2} + ... + \binom{K}{K} = 2^K - 1 \quad (1)$$

Si le nombre d'attributs indexables est égal à 5 ($K = 5$), alors le nombre d'index possible est égal à 31.

2. *Sélection de plus d'un index de jointure* : pour sélectionner plus d'un IJB, le nombre de possibilités est donné par :

$$N = \binom{2^K-1}{1} + \binom{2^K-1}{2} + ... + \binom{2^K-1}{2^K-1} = 2^{2^K-1} \quad (2)$$

Par exemple, si le nombre d'attributs indexables est égal à 5, alors $N = 2^{31} > 1,2 \times 10^9$).

En nous penchant sur la littérature, nous trouvons que la plupart des travaux proposés pour résoudre le problème de sélection d'index concernent les index mono-tables (Chaudhuri, 2004; Chaudhuri et Narasayya, 1997; Labio et al., 1997; Valentin et al., 2000). Peu d'algorithmes ont été proposés pour la sélection des IJB. Nous pouvons citer deux algorithmes proposés dans (Aouiche et al., 2005) et (Bellatreche et al., 2008). Ces algorithmes sont dirigés par les *techniques de fouille de données*. Dans (Aouiche et al., 2005), l'algorithme CLOSE (Pasquier et al., 1999) de recherche des motifs fréquents fermés est utilisé pour élaguer l'espace de recherche des IJB. Pour déterminer si un motif est fréquent ou non, un support minimum $minsup$ doit être fixé. Notons que les motifs générés servent à construire l'ensemble des IJB candidats. Un algorithme glouton est ensuite exécuté pour sélectionner une configuration finale d'IJB.

Aouiche et al. (2005) considèrent seulement les fréquences d'accès des attributs comme critère de génération des motifs fréquents fermés. Bellatreche et al. (2008) ont montré à travers un exemple que la fréquence d'accès seule ne permet pas de sélectionner un ensemble d'IJB efficace. En effet, les IJB sont créés pour optimiser des jointures entre la table des faits et les

[1]Un prédicat de sélection possède la forme suivante : $Attribut\theta valeur$ où $\theta \in \{<, >, \leq, \geq, =, \neq\}$ et $valeur$ appartient au domaine de l'attribut

tables de dimension. En utilisant l'approche de Aouiche et al., l'algorithme peut éliminer des index sur des attributs non fréquemment utilisés mais qui appartiennent à des tables de dimension volumineuses, ce qui ne permet pas d'optimiser une opération de jointure. Pour pallier ce problème, Bellatreche et al. proposent l'algorithme $DynaClose$ qui permet d'inclure d'autres paramètres dans la génération des motifs fréquents comme la taille des tables de dimension, la taille de la page système, etc. Notons que certains travaux ont aussi étudié la combinaison des IJB avec d'autres techniques comme le traitement parallèle (Stöhr et al., 2000) et la fragmentation horizontale (Bellatreche et al., 2007).

Nous présentons dans la section suivante notre approche de sélection des IJB. Elle repose sur le principe de sélectionner une configuration initiale, puis de construire une configuration finale en améliorant cette configuration de manière itérative.

4 Notre approche de sélection des IJB

Nous présentons dans cette section notre approche de sélection d'une configuration d'IJB visant à réduire le coût d'exécution d'une charge de requêtes sous une contrainte de stockage. Nous proposons deux algorithmes de sélection, le premier sélectionne une configuration d'index mono-attributs et le deuxième un ensemble d'index multi-attributs. L'architecture générale de notre approche est représentée dans la figure 2. Notons que les deux algorithmes utilisent un modèle de coût pour évaluer la qualité des solutions obtenues. Ce modèle est proposé dans Aouiche et al. (2005) et permet d'estimer la taille des IJB ainsi que le coût d'exécution des requêtes en présence d'une configuration d'IJB. Notons que pour l'estimation des tailles des IJB et du coût des requêtes, nous avons considéré une distribution non uniforme des données sur l'entrepôt. Cette distribution repose sur les facteurs de sélectivités des prédicats de sélection calculés directement sur l'entrepôt de données réel.

4.1 Sélection d'une configuration d'index mono-attributs

La sélection d'une configuration d'index mono-attributs se déroule en trois étapes :(1) l'identification des attributs indexables, (2) l'initialisation de la configuration et (3) l'enrichissement de la configuration actuelle par l'ajout de nouveaux index. Dans la première étape, l'ensemble des requêtes est analysé afin d'extraire les attributs indexables. Ces attributs sont les attributs sur lesquels un prédicat de sélection est défini dans la charge de requêtes. Les attributs indexables candidats sont choisis parmi les attributs indexables de faible et de moyenne cardinalité. L'algorithme commence par une configuration initiale composée d'un index mono-attribut défini sur l'attribut ayant une cardinalité minimum noté IJB_{min} (étape 2). La configuration initiale est améliorée itérativement par l'ajout d'index définis sur d'autres attributs non encore indexés (étape 3). L'algorithme s'arrête lorsque l'une des deux conditions suivantes sont satisfaites : aucune amélioration n'est possible et l'espace de stockage est consommé.

4.2 Sélection d'une configuration d'index multi-attributs

Un IJB multi-attributs peut être défini sur plusieurs attributs $\{A_1, A_2, \cdots, A_n\}$ où chaque attribut A_j peut appartenir à n'importe quelle table de dimension. La sélection des IJB commence par une configuration initiale qui permet d'optimiser toutes les requêtes, sans tenir

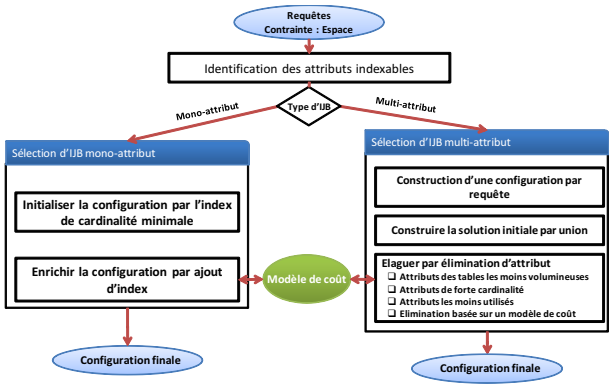

FIG. 2 – *Architecture de notre approche de sélection d'IJB*

compte de son coût de stockage. Pour chaque requête, un index permettant de précalculer toutes les jointures de la requête est sélectionné. La configuration initiale peut satisfaire toutes les requêtes mais risque d'être très volumineuse et viole la contrainte de stockage. Par conséquent, nous procédons à une réduction itérative de sa taille. Quatre étapes caractérisent notre approche de sélection : (1) l'identification des attributs indexables, (2) la construction d'une configuration par requête, (3) la construction d'une configuration initiale et (4) la construction d'une configuration finale.

(1) Identification des attributs indexables : Cette étape se fait de la même manière que dans l'approche de sélection des IJB mono-attribut.

(2) Construction d'une configuration par requête : Dans cette étape, la charge de requête Q est éclatée en m sous-charges. Chaque sous-charge est composée d'une seule requête. Pour chaque requête Q_i nous sélectionnons l'index qui couvre tous les attributs indexables utilisés par cette requête. Cela est motivé par le fait que cet index pré-calcule toutes les jointures de la requête. Par conséquent, aucune jointure n'est effectuée pour exécuter la requête, ce qui permet de réduire considérablement le coût d'exécution de la requête.

Exemple 2 *Supposons 5 attributs indexables : Temps.Mois, Temps.Jour, Produit.Type, Client.Ville et Client.Genre et dix requêtes* $\{Q1, Q2, ..., Q10\}$. *Les attributs indexables utilisés par chaque requête sont représentés dans la matrice requête-attribut représentée dans la figure 3(a). L'application de la première étape sur cette charge permet de sélectionner 10 IJB, chacun correspond à une requête et est défini sur tous les attributs indexables de la requête.*

(3) Construction d'une configuration initiale : Le but de cette étape est de construire une configuration initiale à partir de l'ensemble des index générés lors de l'étape précédente. Cette configuration initiale est générée en effectuant l'union des index obtenus afin que chaque requête puisse être optimisée via l'IJB contenant tous les attributs indexables qu'elle utilise. Notons que le nombre d'index dans la configuration initiale est inférieur ou égal au nombre de requêtes de la charge, car certaines requêtes partagent les mêmes index.

Exemple 3 *L'application de cette étape sur l'exemple précédant permet de créer une configuration initiale composée de 7 IJB, car les requêtes* $(Q5, Q8)$, $(Q6, Q9)$ *et* $(Q7, Q10)$ *partagent*

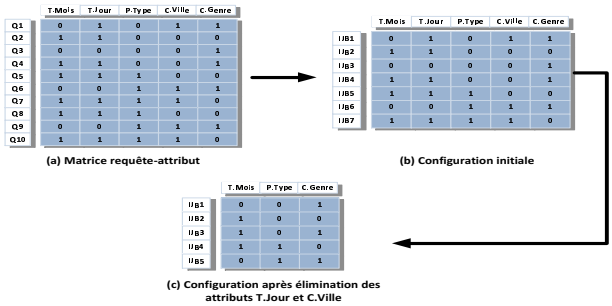

FIG. 3 – *Exemple de génération d'une configuration initiale et d'élimination d'attributs*

respectivement les mêmes index. La configuration initiale générée est représentée dans la figure 3(b).

(4) Construction d'une configuration finale : La configuration initiale obtenue dans l'étape précédente ne prend pas en considération la contrainte d'espace qui peut être violée à cause du nombre important d'IJB sélectionnés pour satisfaire toutes les requêtes. Le but de cette étape est donc d'améliorer la configuration initiale afin de réduire le coût d'exécution des requêtes et de respecter la contrainte de stockage. Notons que si la taille de la configuration initiale est inférieure à la capacité de stockage, alors elle est automatiquement choisie comme configuration finale. Dans le cas contraire, des opérations de réduction de la taille de la configuration sont effectuées jusqu'à ce que la contrainte de stockage soit satisfaite.

La réduction de la taille d'un IJB peut être faite en éliminant certains attributs rentrant dans sa définition. Vu le nombre important d'attributs utilisés par la configuration initiale, l'administrateur est confronté au problème du choix des attributs à éliminer. Plusieurs stratégies peuvent être suivies pour effectuer ce choix. Dans notre étude, nous avons considéré les quatre stratégies suivantes :

- Elimination des attributs de forte cardinalité (AFC) : La principale cause de l'explosion de la taille des IJB est la cardinalité des attributs indexés. Dans cette stratégie, l'attribut de forte cardinalité est éliminé de tous les index constituant la configuration initiale. Pour cela, les attributs sont triés par ordre décroissant sur leur cardinalité, ils seront éliminés dans l'ordre jusqu'à ce que la taille de la configuration devienne inférieure à la capacité de stockage.

- Elimination des attributs appartenant aux tables moins volumineuses (TMV) : le principe de cette stratégie est de donner une priorité aux index précalculant des jointures entre la table des faits et des tables de dimension volumineuses. Le coût de ces jointures est important par rapport aux jointures impliquant des tables de dimension de faible volume. Si plusieurs attributs d'une même table sont candidats pour la suppression, alors ceux ayant une forte cardinalité sont éliminés les premiers.

- Elimination des attributs les moins utilisés (AMU) : cette stratégie suppose que les attributs les plus utilisés doivent être indexés pour satisfaire plus de requêtes. Par conséquent, les attributs les moins utilisés sont éliminés en premier.

- Elimination des attributs apportant moins de réduction de coût(MC) : l'inconvénient des stratégies que nous venons de présenter est qu'elles éliminent des attributs sans avoir une évaluation des configurations intermédiaires issues de chaque élimination. Cela nécessite de

définir une stratégie utilisant un modèle de coût pour choisir les meilleures configurations. Pour avoir ces configurations, les éliminations d'attributs qui engendrent des configurations de mauvaise qualité seront retenues. Pour k attributs candidats, k éliminations sont effectuées et celle engendrant un coût d'exécution maximum sera retenue.

Exemple 4 *Considérons les 10 requêtes dont la matrice requête-attribut est représentée dans la figure 3(a) et la configuration initiale se trouvant sur la figure 3(b). Supposons que les cardinalités des attributs Mois, Jour, Type, Ville et Genre sont respectivement 12, 31, 5, 50 et 2. En appliquant la stratégie AFC, les attributs sont éliminés dans l'ordre suivant : Ville, Jour, Mois, Type, Genre. Après l'élimination des attributs Ville et Jour par exemple, nous obtenons la configuration représentée sur la figure 3(c). Nous constatons que ces deux attributs ne sont plus indexés et que le nombre d'IJB dans la configuration est descendu de 7 à 5, ce qui permet de réduire le coût de stockage de la configuration résultante.*

5 Étude de Performance

Nous avons implémenté notre approche de sélection d'IJB sous Visual C++, une machine Core 2 Duo et une RAM de 2 Go. Nous avons développé une architecture modulaire basée sur cinq modules (voir figure 4) : (1) Module d'interrogation de la méta-base, (2) Module de gestion des requêtes, (3) Module de sélection de configuration d'IJB, (4) Module d'indexation et (5) Module de réécriture des requêtes. Le module d'interrogation de la méta-base permet de collecter un certain nombre d'informations à partir de la méta-base. Ces informations touchent deux niveaux, logique et physique. Les informations du niveau logique regroupent la liste des tables du schéma logique que l'utilisateur gère ainsi que les attributs appartenant à ces tables. Les informations du niveau physique représentent les techniques d'optimisation utilisées ainsi qu'un ensemble de statistiques sur les tables et attributs de l'entrepôt.

Le module de gestion des requêtes permet de définir la charge de requêtes les plus fréquentes (Q) sur laquelle l'optimisation est basée. Le module permet une édition manuelle des requêtes ou une importation à partir de fichiers externes. Le module de sélection d'IJB nécessite en entrée un schéma de l'entrepôt, une charge de requêtes (Q) et un espace de stockage S que l'administrateur réserve pour les index. Il sélectionne une configuration d'IJB ($CIJB$) permettant de minimiser le temps d'exécution des requêtes en entrée et occupant un espace de stockage ne dépassant pas S en utilisant les différentes stratégies de sélection. Le module d'indexation est responsable de la création physique des IJB sélectionnés par le module de sélection. Il génère les requêtes de création des index sur l'entrepôt de données. Le module de réécriture permet d'ajouter un $Hint$ dans la clause SELECT pour forcer l'utilisation des index sélectionnés [2].

Notre étude expérimentale est effectuée sur l'entrepôt de données issu du banc d'essais APB-1 (OLAP-Council, 1998). Rappelons que cet entrepôt est composé d'une table de faits *Actvars* et quatre tables de dimension *ProdLevel*, *TimeLevel*, *CustLevel* et *ChanLevel*. Nous avons considéré 12 attributs candidats à l'indexation (*ClassLevel, GroupLevel, FamilyLevel, LineLevel, DivisionLevel, YearLevel, MonthLevel, QuarterLevel, RetailerLevel, CityLevel, Gen-*

[2] Un $hint$ est un commentaire ajouté à une requête pour forcer un certain plan d'exécution. Le hint *INDEX* permet de forcer l'utilisation d'un ou plusieurs index définis sur les tables référencées par la requête.

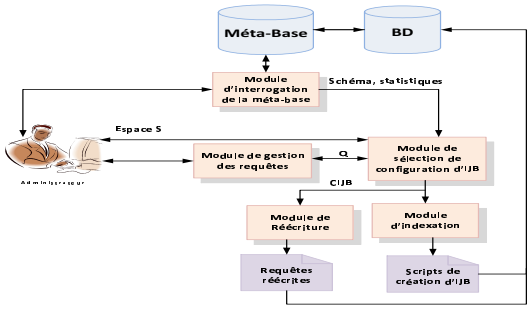

FIG. 4 – *Architecture d'implémentation de notre approche*

derLevel et *ALLLevel*) dont les cardinalités sont respectivement : 605, 300, 75, 15, 4, 2, 12, 4, 99, 4, 2, 5.

Nous avons considéré aussi une charge de 60 requêtes de jointure en étoile définies sur cet entrepôt. La charge de requêtes a été choisie pour couvrir les 12 attributs candidats où chaque requête utilise un, deux ou plusieurs prédicats de sélection définis sur un ou plusieurs attributs. Nous présentons ci-dessous deux types d'expériences, une évaluation est effectuée en utilisant le modèle de coût théorique enfin une validation est faite en utilisant un entrepôt réel sous Oracle 10g.

5.1 Évaluation Théorique

Nous avons effectué plusieurs expériences en utilisant le modèle de coût théorique. Ces expériences visent à comparer les performances de chaque stratégie de sélection. Nous avons implémenté trois algorithmes de sélection : (1) l'algorithme de sélection d'une configuration mono-attributs (MI), (2) l'algorithme glouton de sélection d'une configuration d'index multi-attributs avec quatre stratégies différentes d'amélioration : TMV, AFC, AMU et MC et (3) l'algorithme de datamining (DM) défini par (Aouiche et al., 2005).

L'application de notre approche de sélection d'IJB multi-attributs génère une solution initiale occupant *26,4 Go*. Si l'espace de stockage disponible est supérieur ou égal à cette valeur, alors toutes les stratégies de sélection d'IJB que nous avons considéré donnent la même configuration. Dans le cas contraire, les différentes stratégies proposées permettent d'éliminer des attributs indexables pour satisfaire la contrainte de stockage.

Dans la première expérimentation, nous avons considéré plusieurs valeurs de l'espace de stockage nécessaire pour les index sélectionnés (de 0 à 4 Go). La figure 5 montre les résultats obtenus. L'algorithme DM ne prend pas en considération la contrainte d'espace dans la stratégie de sélection. Les résultats contenus dans cette figure ont été obtenus avec *minsup*=0.25. Les résultats montrent que les meilleures performances ont été obtenues par l'algorithme MI et MC. Cela s'explique par le fait que ces deux algorithmes sont basés sur un modèle de coût. Ce modèle prend en considération plusieurs paramètres comme les facteurs de sélectivité, les cardinalités, les tailles des tables de dimension, etc. Les autres algorithmes se basent chacun sur un seul paramètre. Par exemple, l'algorithme AMU est moins bon que les deux derniers algorithmes, car il ne prend en considération que la fréquence d'utilisation comme seul paramètre

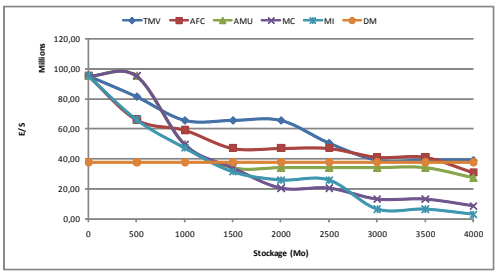

FIG. 5 – *Comparaison de performance en fonction de l'espace stockage*

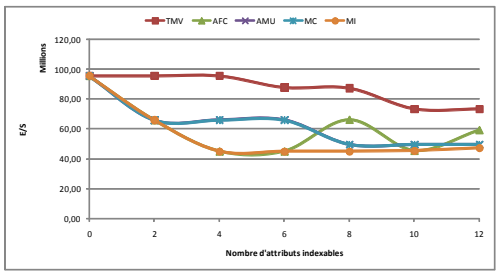

FIG. 6 – *Nombre d'attributs indexables vs performance*

de sélection. Les algorithmes TMV et AFC donnent les plus mauvais résultats du fait qu'ils reposent respectivement sur la taille des tables de dimension et la cardinalité des attributs comme paramètres de sélection. Nous pouvons conclure que la sélection d'une configuration d'index doit considérer en même temps plusieurs paramètres pour garantir une meilleure performance.

L'algorithme *DM* donne des résultats qui ne prennent pas en considération l'espace de stockage, ce qui explique la ligne horizontale sur le graphe. Lorsque l'espace de stockage est réduit, la configuration générée par *DM* est meilleure, car les autres algorithmes sélectionnent peu d'index. Lorsque le quota d'espace augmente, ces algorithmes permettent de sélectionner une configuration meilleure.

Dans la deuxième expérience, nous avons changé le nombre d'attributs candidats pour la sélection d'IJB. Nous avons changé ce nombre de 1 à 12. La figure 6 montre la variation du coût d'exécution en fonction du nombre d'attributs candidats. La performance des requêtes augmente lorsque le nombre d'attributs candidats augmente. Cela est dû au fait que lorsque le nombre d'attributs candidats augmente, les index sélectionnés couvrent plusieurs attributs, par conséquent les requêtes utilisant ces attributs exploitent les index créés et évitent d'effectuer des jointures entre la table des faits et les tables de dimension de ces attributs. Les résultats montrent aussi que lorsque le nombre d'attributs dépasse un certain nombre (8 dans notre cas), peu d'amélioration est constatée. Cela est dû à l'espace de stockage qui une fois saturé ne permet pas de sélection d'autres index.

Nous avons effectué des expériences pour montrer l'effet du nombre des tables de dimension dans la sélection des index. Nous avons choisi 1, 2, 3 ensuite 4 tables et à chaque choix nous exécutons les différents algorithmes. La figure 7 montre les résultats obtenus. Généralement, lorsque le nombre de tables augmente, la performance augmente, car plus de tables sont indexées, en conséquence plus de jointures sont pré-calculées. Cependant, nous constatons que l'algorithme *TMV* n'obéit pas à cette règle, en utilisant une seule table il donne de meilleurs résultats que 2 ou plus. Cela est dû au fait que lorsque le nombre de tables augmente l'algorithme élimine les tables les moins volumineuses qui sont peut-être intéressantes à indexer. La performance ne s'améliore pas considérablement à partir d'un certain nombre de tables compte tenu de à la contrainte d'espace de stockage, qui ne permet pas de sélectionner d'autres index malgré l'ajout de tables.

Dans la dernière expérience, nous avons modifié la cardinalité des attributs indexables. Nous avons considéré que tous les attributs ont la même cardinalité et nous avons choisi plu-

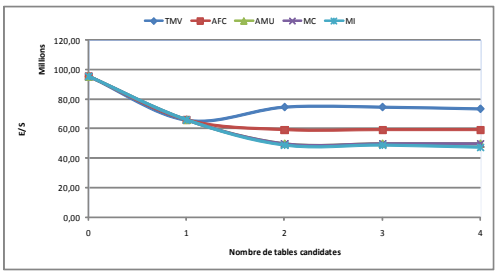

FIG. 7 – *Nombre de tables de dimension vs performance*

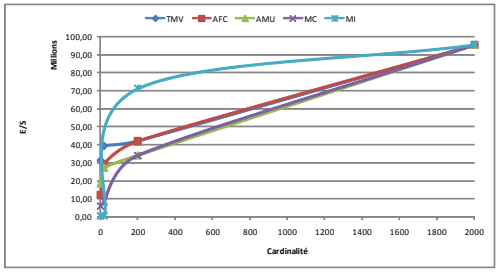

FIG. 8 – *Cardinalité vs performance*

sieurs valeurs (de 2 à 2000) et pour chaque valeur nous avons exécuté nos différents algorithmes. La figure 8 montre les résultats obtenus dans cette expérimentation. La performance se détériore considérablement avec l'augmentation des cardinalités. Cela est dû à l'augmentation de la taille des index sélectionnés qui provoque la saturation rapide de l'espace de stockage réservé et un coût de chargement énorme des index créés. Lorsque la cardinalité dépasse un certain seuil, soit l'espace de stockage disponible ne permet de sélectionner aucun index, soit les index sélectionnés sont très volumineux, ce qui rend l'utilisation de ces index non avantageuse.

5.2 Validation sous Oracle 10g

Pour valider notre approche sur un entrepôt de données réel, nous avons considéré celui issu du banc d'essais APB-1 (OLAP-Council, 1998). Nous avons considéré les 12 attributs indexables et 60 requêtes de jointure en étoile appartenant à plusieurs catégories : requêtes de type count(*) avec et sans agrégation, requêtes utilisant les fonctions d'agrégation comme Sum, Min, Max, requêtes ayant des attributs de dimension dans la clause SELECT, etc.

Nous avons appliqué notre approche de sélection avec un espace de stockage de *3 Go*. Nous avons exécuté nos algorithmes pour chaque stratégie de sélection. Pour l'algorithme *DM*, nous avons considéré une solution qui respecte l'espace de stockage disponible avec *minsup*=0,20. Le tableau 1 montre les index sélectionnés dans chaque stratégie.

Nous avons créé physiquement les configurations générées par chaque algorithme puis nous avons exécuté les 60 requêtes pour chaque configuration. Nous avons forcé l'utilisation des IJB en utilisant les *hint*. Pour vérifier que les IJB sont bien utilisés par Oracle pour exécuter une requête donnée, nous avons utilisé l'outil *Explain Plan* fournit avec Oracle. Cet outil permet de visualiser le plan d'exécution détaillé d'une requête. Les figures 9 et 10 montrent respectivement le temps d'exécution de l'ensemble des requêtes et le taux de réduction de ce coût en utilisant la configuration d'IJB issue de chaque stratégie. L'exécution des 60 requêtes sur toutes les configurations d'index nous a permis de confirmer la grande utilité des IJB pour les requêtes de type *Count(*)*. Le coût d'exécution de ces requêtes en utilisant les IJB est négligeable devant le coût sans index. Ces requêtes sont celles les plus bénéficiaires des IJB, car aucun accès aux données n'est effectué (l'accès aux IJB suffit pour répondre à ces requêtes). Les requêtes les moins bénéficiaires des IJB sont celles référençant des attributs de dimension dans la clause SELECT ou celles ayant moins d'attributs indexés, car elles nécessitent des

Stratégie	Nombre d'IJB sélectionnés	Attributs indexés	Tables de dimension indexées
MV	4 index mono-attributs	GroupLevel, FamilyLevel, LineLevel, DivisionLevel	Prodlevel
DM	3 index mono-attributs	MonthLevel, AllLevel, RetailerLevel	Timelevel, Chanlevel, Custlevel
AFC	4 index mono-attributs, 4 index multi-attributs	QuarterLevel, DividionLevel, YearLevel, GenderLevel	Timelevel, ProdLevel, Custlevel
AMU	3 index mono-attributs	YearLevel, MonthLevel	Timelevel
MC	4 index mono-attributs, 5 index multi-attributs	QuarterLevel, YearLevel, MonthLevel, AllLevel	Timelevel, Chanlevel
MI	9 index mono-attributs	FamilyLevel, LineLevel, DivisionLevel, YearLevel, MonthLevel, QuarterLevel, GenderLevel, CityLevel, AllLevel	Prodlevel, Chanlevel, TimeLevel, Custlevel

TAB. 1 – *Caractéristiques des configurations générées par chaque stratégie*

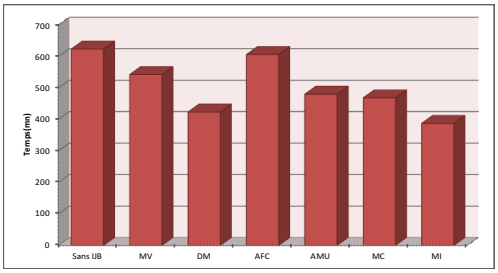

FIG. 9 – *Temps d'exécution des requêtes sous Oracle*

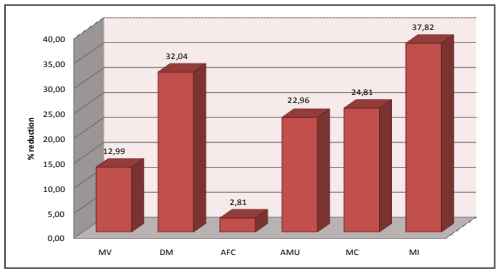

FIG. 10 – *Pourcentage de réduction du coût d'exécution*

jointures supplémentaires entre les tables de dimension et la table des faits. Nous avons aussi constaté que les IJB multi-attributs sont plus efficaces lorsqu'ils couvrent tous les attributs indexables de la requête. Dans le cas contraire, Oracle utilise souvent des jointures par hachage. L'utilisation des IJB mono-attributs est généralement bénéfique car elle permet d'utiliser les opérations logiques pour répondre à des requêtes référençant plusieurs attributs indexés.

Les résultats globaux obtenus sous Oracle montrent que l'approche de sélection MI ainsi que l'algorithme *DM* donnent les meilleurs résultats. Les index générés par ces deux algorithmes sont tous mono-attributs et couvrent respectivement trois et quatre tables de dimension. Pour l'algorithme de génération d'IJB multi-attributs, la stratégie MC est la plus bénéfique car elle repose sur un modèle de coût et prend en considération plusieurs paramètres. La stratégie *AMU* donne des résultats proches de ceux donnés par MC. Cela est dû au fait que les attributs et la table indexée sont ceux les plus utilisés par les 60 requêtes, ce qui permet d'optimiser une bonne partie d'entre elles. Cela montre d'un coté qu'il est intéressant de considérer la fréquence d'utilisation dans la sélection d'index et de l'autre coté de considérer d'autres paramètres (taille des tables, les cardinalités, etc.). Les résultats montrent aussi que les stratégies TMV et AFC donnent des résultats de très mauvaise qualité. La stratégie TMV élimine les tables moins volumineuses, or leur indexation peut être très bénéfique (la table *Chanlevel* par exemple, est indexée dans *MC*, *MI* et *DM* qui sont les stratégies donnant les meilleures performances). La stratégie AFC élimine les attributs de forte cardinalité du processus d'indexation,

or indexer un attribut de faible cardinalité rarement utilisé peut être bénéfique à quelques requêtes seulement mais pas pour toute la charge. Par exemple l'algorithme DM indexe des attributs *RetailerLevel* et *MonthLevel* ayant une cardinalité importante par rapport aux autres (99 et 12) mais cet algorithme donne de meilleurs résultats car ces attributs sont utilisés par un nombre important de requêtes.

6 Conclusion

Les IJB sont des structures redondantes permettant de précalculer les jointures en étoile. Ces index sont caractérisés par une représentation binaire permettant l'utilisation des opérations logiques pour évaluer des conjonctions ou disjonctions de prédicats contenus dans les requêtes de jointure en étoile. Ils sont généralement recommandés pour les attributs de faible cardinalité. Nous avons étudié dans ce papier le problème de sélection d'une configuration d'IJB sous une contrainte d'espace de stockage. Nous avons formalisé la sélection d'IJB comme un problème d'optimisation et nous avons décrit sa complexité par rapport au nombre d'attributs indexables. Nous avons proposé une approche de sélection d'une configuration d'index en utilisant deux algorithmes gloutons. Le premier permet de sélectionner une configuration d'index mono-attributs et le deuxième une configuration multi-attributs. Les deux algorithmes permettent d'abord de sélectionner une configuration initiale, ensuite de procéder à son amélioration tant que la contrainte d'espace n'est pas violée. L'amélioration dans le deuxième algorithme passe par l'élimination d'un ensemble d'attributs indexés. Nous avons proposé plusieurs stratégies d'élimination d'attributs. Nous avons mené plusieurs expériences en utilisant un modèle de coût théorique puis sous *Oracle* pour montrer les performances des différents algorithmes et stratégies d'élimination. Les résultats montrent que les algorithmes basés sur le modèle de coût donnent les meilleurs résultats, car ils prennent en considération plusieurs paramètres dans le processus de sélection, tandis que les autres se basent sur un paramètre chacun.

Ce travail peut être amélioré en prenant en compte le coût de mise à jour des IJB comme contrainte supplémentaire de sélection. Puisque la sélection des IJB dépend d'une charge de requêtes fixée en entrée, alors une étude de l'influence de l'évolution de cette charge sur la performance de la configuration obtenue est nécessaire. Cette étude permettra de proposer une approche dynamique de sélection des IJB.

Références

Aouiche, K., O. Boussaid, et F. Bentayeb (2005). Automatic selection of bitmap join indexes in data warehouses. *7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK 05)*, 64–73.

Bellatreche, L., K. Boukhalfa, et M. K. Mohania (2007). Pruning search space of physical database design. In *18th International Conference On Database and Expert Systems Applications (DEXA'07)*, pp. 479–488.

Bellatreche, L., R. Missaoui, H. Necir, et H. Drias (2008). A data mining approach for selecting bitmap join indices. *Journal of Computing Science and Engineering 2*(1), 206–223.

Canahuate, G., T. Apaydin, A. Sacan, et H. Ferhatosmanoglu (2009). Secondary bitmap indexes with vertical and horizontal partitioning. In *12th International Conference on Extending Database Technology (EDBT'09)*, pp. 600–611.

Chan, C. Y. et Y. E. Ioannidis (1998). Bitmap index design and evaluation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 355–366.

Chaudhuri, S. (2004). Index selection for databases : A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering 16*(11), 1313–1323.

Chaudhuri, S. et V. Narasayya (1997). An efficient cost-driven index selection tool for microsoft sql server. *Proceedings of the International Conference on Very Large Databases*, 146–155.

Chaudhuri, S. et V. Narasayya (2007). Self-tuning database systems : A decade of progress. In *Proceedings of the International Conference on Very Large Databases*, pp. 3–14.

Chmiel, J., T. Morzy, et R. Wrembel (2009). Hobi : Hierarchically organized bitmap index for indexing dimensional data. In *11th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, pp. 87–98.

Comer, D. (1979). The ubiquitous b-tree. *ACM Comput. Surv. 11*(2), 121–137.

Informix-Corporation (1997). Informix-online extended parallel server and informix-universal server : A new generation of decision-support indexing for enterprise data warehouses. *White Paper*.

Kimball, R. et K. Strehlo (1995). Why decision support fails and how to fix it. *SIGMOD Record 24*(3), 92–97.

Labio, W., D. Quass, et B. Adelberg (1997). Physical database design for data warehouses. *Proceedings of the International Conference on Data Engineering (ICDE)*.

Lemire, D., O. Kaser, et K. Aouiche (2010). Sorting improves word-aligned bitmap indexes. *Data & Knowledge Engineering 69*(1), 3 – 28.

OLAP-Council (1998). Apb-1 olap benchmark, release ii. *http ://www.olapcouncil.org/research/bmarkly.htm*.

O'Neil, P. et G. Graefe (1995). Multi-table joins through bitmapped join indices. *SIGMOD Record 24*(3), 8–11.

O'Neil, P. et D. Quass (1997). Improved query performance with variant indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 38–49.

Pasquier, N., Y. Bastide, R. Taouil, et L. Lakhal (1999). Discovering frequent closed itemsets. *ICDT*, 398–416.

Stöhr, T., H. Märtens, et E. Rahm (2000). Multi-dimensional database allocation for parallel data warehouses. In *Proceedings of the International Conference on Very Large Databases*, pp. 273–284.

Valduriez, P. (1987). Join indices. *ACM Transactions on Database Systems 12*(2), 218–246.

Valentin, G., M. Zuliani, D. C. Zilio, G. M. Lohman, et A. Skelley (2000). Db2 advisor : An optimizer smart enough to recommend its own indexes. In *ICDE'00*, pp. 101–110.

Wu, K., A. Shoshani, et K. Stockinger (2010). Analyses of multi-level and multi-component compressed bitmap indexes. *ACM Transactions on Database Systems 35*(1).

Sélection des Index de Jointure Binaires

Summary

Physical design of relational data warehouses is mainly based on the selection of a set of indexes to reduce complex OLAP queries execution cost. These data warehouses are generally modeled by a star schema consisting of a large fact table and a set of dimension tables linked to the fact table by their foreign keys. The queries defined on this schema (called *star join queries*) have multiple joins between the fact and dimension tables. These joints are very costly due to the large size of the referenced tables. Bitmap join indexes are well adapted to reduce the cost these joints. they are defined on the fact table using one or more dimension attributes. Select a configuration of indexes reducing the cost of workload with space constraint is known NP-complete problem. In this paper, We first introduce the selection problem of bitmap join indexes and some related works. We present then our approach and selection algorithms that we propose. We conduct experiments to compare different selection strategies. Finally, we make a real validation of different algorithms under Oracle using dataset from APB1 benchmark.