Chapitre 6

Sélection simultanée d'index et de vues matérialisées

6.1 Introduction

Les vues matérialisées et les index sont des structures physiques qui partagent la même ressource d'espace de stockage. Leur sélection doit donc être faite simultanément en prenant en compte les interactions existant entre eux. En effet, la sélection de vues matérialisées et d'index effectuées de façon isolée ou séquentielle ne permet pas d'améliorer les performances de façon optimale, car au moment où sont construits les index sur les vues pour accélérer les requêtes, tout le bénéfice de la matérialisation des vues réside dans la collaboration de celle-ci avec ses propres index pour l'amélioration des performances.

Dans ce chapitre, nous modélisons les relations existant entre les vues candidates, les index candidats et les requêtes de la charge à l'aide de trois matrices : requêtes-index, requêtes-vues et vues-index. Les vues candidates et les index candidats sont fournis respectivement par notre stratégie de sélection de vues matérialisées et d'index. Nous introduisons la notion de bénéfice de matérialisation et d'indexation. Le bénéfice d'indexation est défini comme l'amélioration du coût que peut apporter un index donnée aux requêtes s'il est crée. Le bénéfice de matérialisation est défini quant à lui comme l'amélioration du coût apporté à ces requêtes si une vue donnée est matérialisée. Ces bénéfices sont calculés à l'aide des matrices requêtes-index, requêtes-vues et vues-index pour prendre en compte les interactions

entre les vues et les index. Nous développons également un algorithme glouton qui effectue la sélection simultanée des vues matérialisées et des index. Cet algorithme exploite des modèles de coût qui permettent d'estimer les coûts d'exploitation d'index et de vues, ainsi que leurs coûts de stockage et de maintenance.

Ce chapitre est organisé comme suit. Nous commençons par présenter à la Section 6.2 notre stratégie de sélection simultanées de vues matérialisées et d'index. Nous détaillons à la Section 6.3 les modèles de coût et à la Section 6.4 le calcul du bénéfice d'indexation et de matérialisation. Nous présentons ensuite à la Section 6.5 notre algorithme de sélection simultanée de vues matérialisées et d'index. Pour valider notre approche, nous présentons des expérimentations à la Section 6.6. Nous concluons à la Section 6.7.

6.2 Principe de notre stratégie de sélection simultanée d'index et de vues matérialisées

Le principe général de notre stratégie de sélection simultanée d'index et de vues matérialisées est représenté à la Figure 6.1. À partir de la charge extraite de l'entrepôt de données, nous sélectionnons un ensemble de vues matérialisées et d'index sur ces vues et sur les tables de base. Rappelons qu'une vue matérialisée est une requête nommée dont les données sont stockées sur disque sous la forme d'une table. La sélection d'index peut donc se faire également sur les vues matérialisées.

L'application isolée des algorithmes de sélection d'index et de vues nous fournit un ensemble de vues candidates et d'index candidats. Les index peuvent être définis soit sur les tables de base, soit sur les vues matérialisées candidates. Nous appliquons tout d'abord notre stratégie de sélection de vues matérialisées afin de construire un ensemble de vues candidates pertinent pour la charge (cf. Chapitre 5). Nous appliquons ensuite notre stratégie de sélection d'index pour construire un ensemble d'index pertinent pour les requêtes de la charge et les vues matérialisées candidates générées à l'étape précédente (cf. Chapitre 4).

Nous procédons comme suit pour proposer une configuration d'index et de vues matérialisées :

– extraction de la charge des requêtes,

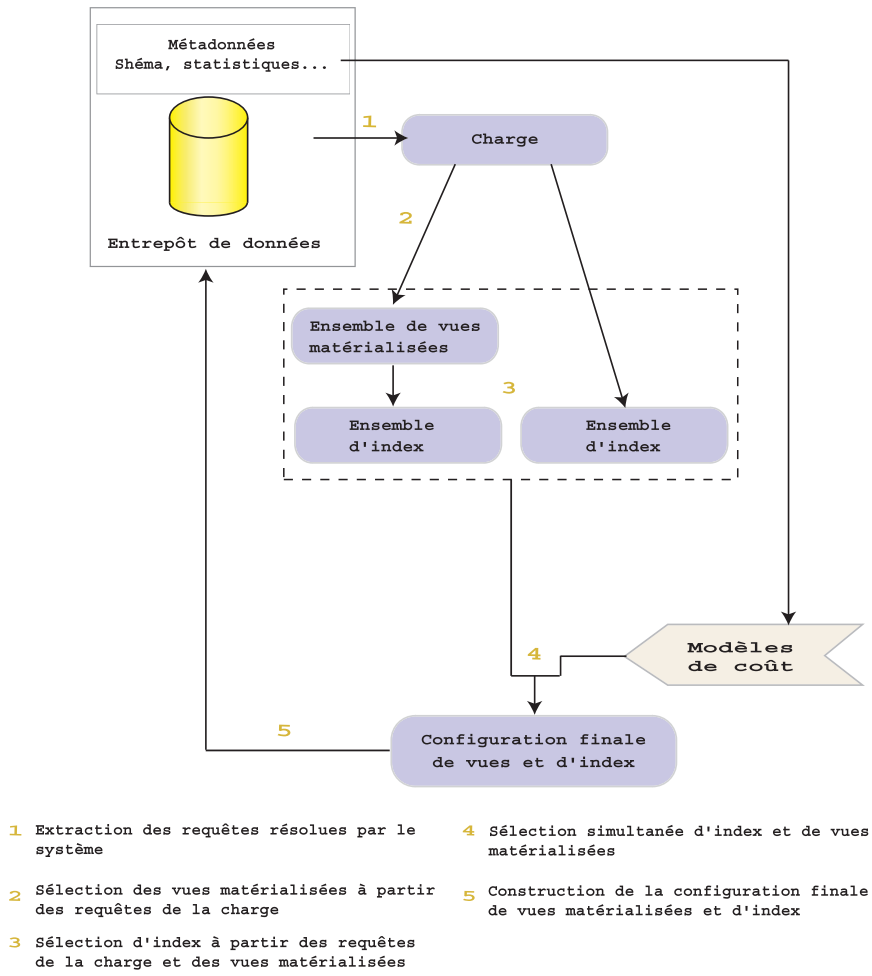

FIG. 6.1 – Architecture de notre stratégie de sélection simultanée d'index et de vues matérialisées

- construction de l'ensemble des vues matérialisées candidates à partir de la charge,
- construction de l'ensemble d'index candidats à partir de la charge et des vues matérialisées,
- sélection simultanée d'index et de vues matérialisées,
- construction de la configuration finale d'index et de vues matérialisées.

Nous ne détaillons pas ici les trois premiers points, car ils sont décrits aux Chapitres 3 et 4. En revanche, nous décrivons en détail dans les sections suivantes notre stratégie de sélection simultanée d'index et de vues matérialisées. Nous illustrons également notre stratégie à l'aide des requêtes de la charge de la Figure 6.2, des vues candidates de la Figure 6.3 et des index

de la Figure 6.4.

q_1 **select** sales.time_id, **sum**(amount_sold)
from sales, times
where sales.time_id = times.time_id
and times.time_fiscal_year = 2000
group by sales.time_id

q_2 **select** sales.prod_id,
sum(amount_sold)
from sales, products, promotions
where sales.prod_id = products.prod_id
and sales.promo_id = promotions.promo_id
and promotions.promo_category = 'news paper'
group by sales.prod_id

q_3 **select** customers.cust_gender, **sum**(amount_sold)
from sales, customers, products,
where sales.cust_id = customers.cust_id
and sales.prod_id = products.prod_id
and customers.cust_marital_status ='single'
and products.prod_category = 'women'
group by customers.cust_gender

q_4 **select** products.prod_name, sum(amount_sold)
from sales, products, promotions
where sales.prod_id = products.prod_id
and sales.promo_id = promotions.prom_id
and promotions.promo_category = 'TV'
group by products.prod_name

q_5 **select** promotions.promo_name,
sum(amount_sold)
from sales, promotions
where sales.promo_id = promotions.promo_id
and promotions.promo_begin_date='30/01/2000'
and promotions.promo_end_date='30/03/2000'
group by promotions.promo_name

q_6 **select** customers.cust_marital_status,
sum(quantity_sold)
from sales, customers, products
where sales.cust_id = customers.cust_id
and sales.prod_id = products.prod_id
and customers.cust_gender = 'woman'
and products.prod_name = 'shampooing'
group by customers.cust_marital_status

q_7 **select** products.prod_name, **sum**(amount_sold)
from sales, products, promotions
where sales.prod_id = products. prod_id
and sales.promo_id =promotions.promo_id
and products.prod_category='tee shirt'
and promotions.promo_end_date='30/04/2000'
group by products.prod_name

q_8 **select** sales.promo_name, sum(quantity_sold)
from sales, promotions
where sales.promo_id = promotions.promo_id
and promotions.promo_begin_date = '30/01/2000'
and promotions.promo_end_date = '30/03/2000'
group by promotions.promo_name

FIG. 6.2 – Requêtes de la charge

v_1 **create materialized view** v_1 **as**
select sales.time_id, times.time_fiscal_year,
sum(amount_sold)
from sales, times
where sales.time_id = times.time_id
group by sales.time_id, times.times_fiscal_year

v_2 **create materialized view** v_2 **as**
select sales.prod_id, sales.cust_id, channels.channel_desc,
sum(quantity_sold)
from sales, channels, products, customers
where sales.prod_id = products.prod_id
and sales.channel_id = channels.channel_id
and sales.cust_id = customers.cust_id
group by sales.prod_id, sales.cust_id, channels.channel_desc

v_3 **create materialized view** v_3 **as**
select customers.cust_first_name, products.prod_name,
products.prod_category, customers.cust_gender,
customers.cust_marital_status, **sum**(sales.quantity_sold)
from sales, customers, products
where sales.cust_id = customers.cust_id
and sales.prod_id = products.prod_id
group by customers.cust_first_name, products.prod_name,
products.prod_category, customers.cust_gender,
customers.cust_marital_status

v_4 **create materialized view** v_4 **as**
select products.prod_name, products.prod_category,
promotions.promo_category, **sum**(amount_sold)
from sales, products, promotions
where sales.prod_id = products.prod_id
and sales.promo_id = promotions.promo_id
group by products.prod_name, products.prod_category,
promotions.promo_category

v_5 **create materialized view** v_5 **as**
select sales.prod_id, products.prod_category,
promotions.promo_category, **sum**(amount_sold)
from sales, products, promotions
where sales.prod_id = = products.prod_id
and sales.promo_id = promotions.promo_id
group by sales.prod_id, products.prod_category,
promotions.promo_category

v_6 **create materialized view** v_6 **as**
select channels.channel_class, products.prod_name, channels.channel_desc,
products.prod_category, **sum**(sales.quantity_sold), **sum**(sales.amount_sold)
from sales, channels, products
where sales.prod_id = products.prod_id
and sales.channel_id = channels.channel_id
group by channels.channel_class, products.prod_name,
products.prod_category, channels.channel_desc

v_7 **create materialized view** v_7 **as**
select sales.prod_id, products.prod_category, channels.channel_desc,
promotions.promo_name, promotions.promo_begin_date,
promotions.promo_end_date, products.prod_name, **sum**(sales.quantity_sold),
sum(sales.amount_sold)
from sales, products, promotions
where sales.prod_id = products.prod_id
and sales.promo_id = promotions.promo_id
and sales.channel_id = channels.channel_id
group by sales.prod_id, products.prod_category, channels.channel_desc,
promotions.promo_name, promotions.promo_begin_date,
promotions.promo_end_date, products.prod_name

FIG. 6.3 – Vues matérialisées candidates

index	attributs indexés
i_1	promotions.promo_category
i_2	channels.channel_desc
i_3	channels.channel_class
i_4	customers.cust_marital_status
i_5	customers.cust_gender
i_6	times.time_begin_date
i_7	times.time_end_date
i_8	times.fiscal_year
i_9	products.prod_name
i_{10}	products.prod_category
i_{11}	promotions.promo_name
i_{12}	customers.cust_fisrst_name

FIG. 6.4 – Index candidats

Nous modélisons les relations existantes entre les requêtes, vues matérialisées candidates et index candidats à l'aide de trois matrices : requêtes-vues, requêtes-index et vues-index. Nous décrivons dans les sections suivantes ces trois matrices.

6.2.1 Matrice requêtes-vues

La matrice requêtes-vues modélise les relations existantes entre les requêtes de la charge et les vues matérialisées extraites de cette charge, c'est-à-dire, les vues exploitées par une requête de la charge. Cette matrice peut être vue comme le résultat de la réécriture des requêtes de la charge en fonction des vues matérialisées. Les lignes et les colonnes de cette matrice sont les requêtes de la charge et les vues matérialisées recommandées par notre stratégie de sélection de vues, respectivement. Le terme général de la matrice est égal à un si une requête donnée exploite une vue et est égal à zéro sinon.

Le Tableau 6.1 illustre un exemple de matrice requêtes-vues composée de huit requêtes et de neuf vues matérialisées recommandées pour ces requêtes.

6.2.2 Matrice requêtes-index

La matrice requêtes-index permet d'indiquer les index construits sur les tables de base. La matrice requêtes-index peut être vue comme la réécriture des requêtes de la charge en fonction des index recommandés par notre algorithme de sélection d'index. Les lignes et les colonnes de cette matrice sont respectivement les requêtes de la charge et les index

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
q_1	1	0	0	0	0	0	0
q_2	0	0	0	0	1	0	0
q_3	0	0	1	0	0	0	0
q_4	0	0	0	1	0	0	0
q_5	0	0	0	0	0	0	1
q_6	0	0	1	0	0	0	0
q_7	0	0	1	0	0	0	1
q_8	0	0	0	0	0	0	1

TAB. 6.1 – Matrice requêtes-vues

sélectionnés à partir de cette charge. Le terme général de la matrice est égal à un si une requête donnée exploite un index et est à zéro sinon.

Le Tableau 6.2 illustre un exemple de matrice requêtes-index composée de huit requêtes et de douze index recommandés pour ces requêtes.

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}
q_1	0	0	0	1	0	1	0	0	0	0	0	0
q_2	0	1	0	0	0	0	1	0	0	0	0	0
q_3	0	0	0	1	1	0	0	0	0	1	0	0
q_4	1	0	0	0	0	0	0	0	9	0	0	0
q_5	0	0	0	0	0	1	1	0	0	0	1	0
q_6	0	0	0	1	1	0	0	0	9	0	0	0
q_7	0	0	0	0	0	0	1	0	0	1	0	0
q_8	0	0	0	0	0	1	1	0	0	1	0	0

TAB. 6.2 – Matrice requêtes-index

6.2.3 Matrice vues-index

La matrice vues-index identifie les index construits sur les vues matérialisées recommandées par l'algorithme de sélection de vues. Les lignes et les colonnes de cette matrice sont respectivement les vues matérialisées candidates préalablement sélectionnées et les index candidats recommandés pour ces vues. Le terme général de la matrice est égal à un si une vue donnée exploite un index et à zéro sinon.

Le Tableau 6.3 illustre un exemple d'une matrice vues-index composée de sept vues et de douze index recommandés pour ces vues.

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}
v_1	0	0	0	0	0	0	0	1	0	0	0	0
v_2	0	1	0	0	0	0	0	0	0	0	0	0
v_3	0	0	0	1	1	0	0	0	0	0	0	1
v_4	1	0	0	0	0	0	0	0	1	1	0	0
v_5	1	0	0	0	0	0	0	0	0	1	0	0
v_6	0	1	1	0	0	0	0	0	1	1	0	0
v_7	0	1	0	0	0	0	1	0	1	1	1	0

TAB. 6.3 – Matrice vues-index

6.3 Modèles de coût

Généralement, le nombre d'index candidats et de vues candidates est d'autant plus important que la charge en entrée est volumineuse. La création de tous ces index et vues peut ne pas être réalisable en pratique à cause de la contrainte définie sur l'espace de stockage alloué aux index et vues. Pour pallier ces limitations, nous exploitons des modèles de coût permettant de ne conserver que les index et les vues les plus avantageux. Ces modèles estiment l'espace en octets occupé par les index et les vues, les coûts d'accès aux données à travers ces index et/ou ces vues et le coût de leur maintenance en terme de nombre d'entrées/sorties.

Dans le Chapitre 4, Section 4.4, nous avons développé des modèles qui estiment le coût d'accès aux données à travers des index *bitmaps* de jointure, ainsi que les coûts de maintenance et de stockage de ces index. Nous avons également présenté dans le Chapitre 5, Section 5.4, des modèles qui estiment le coût d'accès aux données à travers des vues matérialisées, ainsi que les coûts de maintenance et de stockage de ces vues. Dans la suite de chapitre, nous ne développons donc que les modèles de coût relatifs aux index en B-arbre. Le Tableau 6.4 résume les notations utilisées pour élaborer ces modèles de coût.

6.3.1 Coût d'accès aux données à travers un index en B-arbre

Pour une valeur donnée $a_i \in domaine(a)$, l'utilisation d'un index en B-arbre construit sur l'attribut a produit une liste d'identifiants de n-uplets, dénotés TID[1]. Ces identifiants permettent un accès direct aux n-uplets possédant la valeur a_i pour l'attribut a. Le coût

[1]TID signifie *Tuple IDentifier*.

Symbole	Description
$\mid X \mid$	Nombre de n-uplets de la table X ou cardinalité de l'attribut X
S_p	Taille en octets d'une page disque
p_X	Nombre de pages nécessaires pour stocker la table X
BF	Facteur de bloc *(bloc factor)*
SF	Facteur de sélectivité *(selectivity factor)*
SNA_q	Les attributs présents dans la clause `Where` de la requête q
SNA_{update}	Les attributs mis à jour dans la requête de mise à jour *update*

TAB. 6.4 – Paramètres des modèles de coût

d'exploitation d'un index en B-arbre est composé de deux parties : le coût d'accès aux n-uplets pertinents pour une requête donnée, dénoté $C_{selection}$ et le coût de maintenance de l'index, dénoté $C_{maintenance}$. Dans les sections suivantes, nous développons le calcul de ces coûts.

Le coût de sélection des n-uplets répondant à la requête q se décompose à son tour en deux parties. La première partie, dénotée $C_{lecture}$, est le coût requis pour former les listes de TID qui peuvent être recherchées en utilisant chaque index de l'ensemble I. La deuxième partie, dénotée $C_{recherche}$, est le coût requis pour rechercher les n-uplets restants après l'intersection de ces listes de TID.

Le nombre de pages accessibles pour former les listes de TID lorsque la requête q requiert une ou plusieurs sélections sur la table T est :

$$C_{lecture}(\{q\}, I) = \sum_{a \in I \cap SNA_q} C_{acces\_index}(a)$$

où SNA_q désigne l'ensemble des attributs des prédicat de sélection présents dans la clause `Where` de la requête q et $C_{acces\_index}(a)$ représente le coût d'accès à un index construit sur l'attribut a. Lorsque cet index est organisé en B-arbre, $C_{acces\_index}(a)$ s'exprime comme suit :

$$C_{acces\_index}(a) = \lceil log_{BF_a} |T| \rceil + \left\lceil \frac{SF_a |T|}{BF_a} \right\rceil - 1$$

où BF_a désigne le facteur de bloc de l'index construit sur l'attribut a, c'est-à-dire, le nombre

moyen de couples $\{valeur, listeTID\}$ par page disque, où $valeur$ et $listeTID$ sont respectivement une valeur de clé d'entrée de l'index et la liste des TID correspondant à cette valeur. SF_a désigne le facteur de sélectivité de l'attribut a. Le premier et le deuxième terme de C_{acces_index} représentent respectivement la hauteur du B-arbre et le nombre de blocs de nœuds feuilles auxquels il faut accéder.

Le nombre de n-uplets $T_q(I)$ à rechercher pour q après l'intersection des listes de TID est :

$$T_q(I) = |T| \prod_{a \in I \cap SNA_q} SF_a$$

À partir du nombre de n-uplets recherchés $T_q(I)$, nous dérivons le nombre de pages disques à rechercher en utilisant la formule de Cardenas [Car75] comme suit :

$$C_{recherhe} = Sp_T \left(1 - (1 - \frac{1}{Sp_T})^{T_q(I)}\right).$$

En résumé, le coût d'exploitation d'un ensemble d'index en B-arbre I par une requête q est égal à :

$$C(\{q\}, I) = |T| \prod_{a \in I \cap SNA_q} SF_a + Sp_T \left(1 - (1 - \frac{1}{Sp_T})^{T_q(I)}\right) + C_{maintenance}(I).$$

6.3.2 Coût de maintenance

La mise à jour des données requiert la maintenance des index construits sur ces données afin de conserver leur cohérence. Ce coût s'exprime en fonction des fréquences d'insertion f_i, de suppression f_d et de modification f_u comme suit :

$$C_{maintenance}(I) = \sum_{i \in insert} f_i \sum_{a \in I} C_{insert}(a) + \sum_{d \in delete} f_d \sum_{a \in I} C_{delete}(a) + \sum_{i \in update} f_u \sum_{a \cap SNA_{update}} C_{insert}(a)$$

où C_{insert}, C_{delete} et C_{update} sont respectivement le coût maintenance dû à une opération d'insertion $insert$, de suppression $delete$ et de modification $update$. SNA_{update} et a désignent respectivement les attributs à mettre à jour et l'attribut indexé.

Les coûts C_{insert}, C_{delete} et C_{update} d'un index construits sur un attribut a sont donnés respectivement par les formules suivantes [KY87] :

$$C_{insert} = C_{delete} = \lceil log_{BF_a}|T| \rceil$$
$$C_{update} = \lceil log_{BF_a}|T| \rceil + \left\lceil 0,5 \times SF_a \frac{|T|}{BF_a} \right\rceil - 1$$

6.3.3 Coût de stockage d'un index en B-arbre

La taille de stockage, en octets, d'un index structuré en B-arbre est donnée par la formule [Wu99] :

$$taille_{B-arbre} = \frac{1,44 S_p |T|}{m}$$

où S_p est la taille d'une page disque, m l'ordre du B-arbre et |T| le nombre de n-uplets de la table indexée.

6.4 Calcul du bénéfice de matérialisation et d'indexation

Le bénéfice apporté par la sélection d'un objet (index, vue matérialisée ou vue matérialisée avec ses index) est défini comme la différence entre le coût des requêtes de la charge à un moment donné et le coût de ces mêmes requêtes suite à l'ajout de cet objet.

Soient Q une charge de requêtes, $Config$ une configuration composée de vues matérialisées et d'index construits sur les tables de base ou les vues, QI, QV et VI sont respectivement les matrices requêtes-index, requêtes-vues et vues-index. Nous développons dans la suite le calcul du bénéfice pour un index ou une vue matérialisée.

6.4.1 Bénéfice apporté par un index

L'ajout d'un index à la configuration $Config$ peut conduire aux alternatives résumées au Tableau 6.5. En effet, l'ajout d'un index donné à la configuration $Config$ peut améliorer de façon directe le coût des requêtes de la charge ou indirectement à travers des vues auxquelles cet index est associé.

	$VI[v,i]=1$	$VI[v,i]=0$
$v \in Config$	min (bénéfice de matérialisation, bénéfice d'indexation de v)	bénéfice d'indexation
$v \notin Config$	—	bénéfice d'indexation

TAB. 6.5 – Bénéfice apporté par l'ajout d'index

6.4.2 Bénéfice apporté par une vue matérialisée

L'ajout d'une vue matérialisée à la configuration $Config$ peut conduire aux alternatives résumées au Tableau 6.6. En effet, l'ajout d'une vue donnée à la configuration $Config$ peut améliorer de façon directe le coût des requêtes de la charge ou de façon collaborative avec les index associés à cette vue.

	$VI[v,i]=1$	$VI[v,i]=0$
$i \in Config$	—	bénéfice d'indexation
$i \notin Config$	min (bénéfice d'indexation, bénéfice de matérialisation)	bénéfice de matérialisation

TAB. 6.6 – Bénéfice apporté par l'ajout de vue

Le bénéfice d'indexation apporté par l'ajout d'un index i est calculé comme suit :

$$benefice(Q, Config \cup \{i\}) = \begin{cases} \frac{C(Q,Config) - C(Q,Config \cup \{i\})}{taille(i)} & \text{si } \forall v \in V, VI[v,i] = 0, \\ \frac{C(Q,Config) - C(Q,Config \cup \{i\} \cup V')}{taille(\{i\}) + \sum_{v' \in V'} taille(\{v'\})} & \text{si } V' = \{v \in Config, VI[v,i] = 1\} \neq \emptyset \\ 0 & \text{sinon.} \end{cases}$$

Le bénéfice de matérialisation apporté par la vue v est calculé comme suit :

$$benefice(Q, Config \cup \{v\}) = \begin{cases} \frac{C(Q,Config) - C(Q,Config \cup \{v\})}{taille(v)} & \text{si } \forall i \in I, VI[v,i] = 0, \\ \frac{C(Q,Config) - C(Q,Config \cup \{v\} \cup I')}{taille(\{v\}) + \sum_{i' \in I'} taille(\{i'\})} & \text{si } I' = \{i \in Config, VI[v,i] = 1\} \neq \emptyset. \\ 0 & \text{sinon.} \end{cases}$$

6.5 Algorithme de sélection simultanée d'index et de vues matérialisées

Notre algorithme de sélection d'index et de vues matérialisée (Algorithme 16) est basé sur une recherche gloutonne dans l'ensemble des objets obtenus en réalisant l'union de l'ensemble

d'index candidats I et l'ensemble de vues candidates V ($O = I \cup V$). Rappelons que les ensembles I et V sont respectivement fournis par notre stratégie de sélection d'index et notre stratégie de sélection de vues matérialisées ou par toute autre stratégie. La fonction objectif F est définie comme suit :

$$F_{/Config}(\{o_i\}) = \text{bénéfice}(Q, Config) - \beta C_{maitenance}(\{o_i\})$$

où o_i peut être un index ou une vue et $\beta = |Q|\, p(o_i)$ estime le nombre de mises à jour de o_i. La probabilité de mise à jour $p(o_i)$ est égale à $\frac{1}{\text{nombre d'éléments de } O} \frac{\%\text{rafraîchissement}}{\%\text{interrogation}}$, où le ratio $\frac{\%\text{rafraîchissement}}{\%\text{interrogation}}$ représente la proportion de rafraîchissement par rapport à la proportion d'interrogation de l'entrepôt de données.

Soit S l'espace disque alloué par l'administrateur de l'entrepôts de données pour stocker les vues matérialisées et les index. À la première itération de l'algorithme de sélection simultanée d'index et de vues, les valeurs de la fonction objectif sont calculées pour chaque index ou vue de l'ensemble O. Le coût d'exécution de toutes les requêtes de la charge Q est égal au coût d'exécution de ces requêtes à partir des tables de base sans indexation ni vue. L'ensemble o_{max} de vues et/ou d'index qui maximise F, s'il existe ($F_{Config}(\{o_{max}\} > 0$), est alors ajouté à l'ensemble $Config$ si l'espace de stockage S n'est pas atteint. Si l'ajout est fructueux, l'espace S est diminué de l'espace de stockage occupé par o_{max}. L'espace occupé par o_{max} dépend de son contenu (index et/ou vue). Nous utilisons les fonctions booléennes $index(o_{max})$ et $vue(o_{max})$ qui renvoient vrai si o_{max} est un index ou une vue (lignes 15 à 25 de l'Algorithme 16), respectivement.

Les valeurs de la fonction objectif F sont ensuite recalculées pour chaque élément restant dans l'ensemble $O - Config$, car elles dépendent des vues et des index sélectionnés présents dans $Config$. Cela aide à prendre en compte les interactions qui peuvent exister entre les index et les vues. Rappelons que cette interaction est prise en compte dans la manière dont le bénéfice est calculé, qui exploite les matrices requêtes-index QI, requêtes-vues QV et vues-index VI.

Nous répétons ces itérations jusqu'à ce qu'il n'y ait plus d'amélioration de la fonction objectif ($F_{/Config_V}(o_{max}) \leq 0$), que tous les index et vues aient été sélectionnés ($V - Config_V = \emptyset$) ou que limite d'espace de stockage soit atteinte ($S \leq 0$).

Algorithme 16 Sélection simultanée de vues matérialisées et d'index

1: $Config \leftarrow \emptyset$
2: $O \leftarrow I \cup V$
3: **répéter**
4: $o_{max} \leftarrow \emptyset$
5: $taille(o_{max}) \leftarrow 0$
6: $benefice_{max} \leftarrow 0$
7: **pour tout** $o_i \subseteq O - Config$ **faire**
8: **si** $F_{Config}(\{o_i\}) > benefice_{max}$ **alors**
9: $benefice_{max} \leftarrow F_{Config}(\{o_i\})$
10: $o_{max} \leftarrow \arg\max(F_{Config}(\{o_i\})$
11: {l'ensemble o_{max} peut contenir des index et des vues matérialisées}
12: **fin si**
13: **fin pour**
14: **si** $F_{max}(o_{max}) > 0$ **alors**
15: **si** $index(o_{max}) = vrai$ **alors**
16: $taille(o_{max}) \leftarrow taille_{index}(o_{max})$
17: **sinon**
18: **si** $vue(o_{max}) = vrai$ **alors**
19: $taille(o_{max}) \leftarrow taille_{vue}(o_{max})$
20: **sinon**
21: $o_{max} = i_{max} \cup v_{max}$
22: $taille(o_{max}) \leftarrow taille(o_{max}) + taille_{index}(i_{max})$
23: $taille(o_{max}) \leftarrow taille(o_{max}) + taille_{vue}(v_{max})$
24: **fin si**
25: **fin si**
26: **fin si**
27: $S \leftarrow S - taille(o_{max})$
28: **jusqu'à** $(F_{Config}(o_{max}) \leq 0$ ou $O - Config \leftarrow \emptyset$ ou $S \leq 0)$

6.6 Expérimentation

Afin de valider notre stratégie de sélection simultanée d'index et de vues matérialisées, nous l'avons expérimentée sur notre entrepôt de données test (cf. Chapitre 4, Section 4.6).

Nous avons mesuré le temps d'exécution des requêtes de cette charge dans les cas suivants : sans index ni vues matérialisées, avec vues matérialisées, et avec index et vues matérialisées. La Figure 6.5 représente la variation de ce temps en fonction du pourcentage de l'espace de stockage. Ce pourcentage est calculé par rapport à l'espace total occupé par les index et les vues sélectionnés obtenus en appliquant notre stratégie de sélection d'index

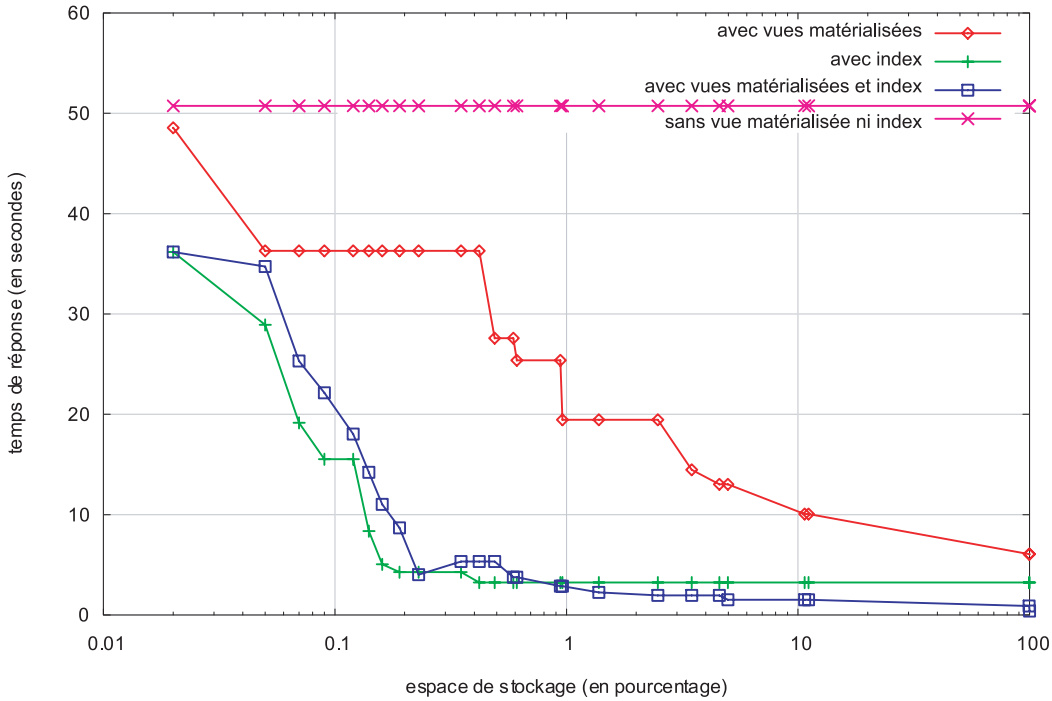

FIG. 6.5 – Résultats de sélection simultanées d'index et de vues matérialisées

et de vues sans appliquer aucune contrainte. Cela permet d'obtenir tout l'espace de vues et d'index pertinents pour la charge selon nos stratégies et, par conséquent, d'avoir un intervalle plus large pour faire varier l'espace de stockage. De plus, pour mieux visualiser les résultats, nous avons utilisé une échelle logarithmique sur l'axe des abscisses.

La Figure 6.5 montre que pour les valeurs élevées de l'espace de stockage, la sélection simultanée d'index et de vues est meilleure que la sélection isolée des index et vues. En revanche, nous constatons que pour les petites valeurs de l'espace de stockage, il peut arriver que la sélection d'index soit plus performante que la sélection simultanée d'index et de vues. Cela peut être expliqué par le fait qu'en général, la taille des index est significativement moins importante que celle des vues. Dans ce cas, on peut mettre dans le même espace (de petite taille) plus d'index que de vues et ainsi améliorer davantage le temps de réponse, car plus on dispose d'index, mieux on améliore les performances.

6.7 Conclusion

Nous avons proposé dans chapitre une nouvelle démarche qui prend en compte réellement l'interaction qui peut exister entre les index et les vues matérialisées en les considérant ensemble comme des structures physiques susceptibles de collaborer afin de réduire le coût d'exécution de requêtes. Cela donne lieu à un partage efficace de l'espace alloué pour le stockage de ces index et vues. En effet, nos résultats expérimentaux montrent que la sélection simultanée de vues et d'index apporte une meilleure amélioration des performances par rapport à la sélection isolée de ces structures.

Bellatreche *et al.* proposent une sorte de compétition entre deux agents qui se disputent l'espace de stockage, tantôt pour les index et tantôt pour les vues matérialisées, en utilisant des politiques de remplacement (cf. Chapitre 3). L'approche de Bellatreche *et al.* part d'une solution initiale composée d'index et de vues sélectionnée d'une manière isolée sous la contrainte d'espace de stockage. Le fait de considérer cette contrainte *a priori* peut éliminer des solutions qui peuvent devenir pertinentes dans les itérations suivantes du processus de sélection simultanée d'index et de vues. Dans notre approche, nous ne spécifions pas la contrainte d'espace de stockage pour le calcul de la solution initiale. Cette contrainte est prise en compte *a posteriori* dans notre algorithme de sélection. De plus, Bellatreche *et al.* ont recours à des politiques de remplacement. Cependant, ces dernières ne reflètent pas en réalité l'interaction vues-index, car les remplacements ne considèrent pas le *bénéfice* apporté aux requêtes par ces structures prises ensemble mais plutôt la *fréquence* de leur usage dans ces requêtes.

D'autre part, Rizzi et Saltarelli proposent une approche orientant l'indexation ou la matérialisation en fonction de la sélectivité des attributs de la clause `Where` des requêtes et de la granularité de leur clause `Group by`. Dans notre approche, ces aspects sont pris en compte implicitement dans le calcul du bénéfice de matérialisation et d'indexation. En effet, les modèles de coût qui permettent de calculer le bénéfice admettent comme paramètres le nombre de dimensions dans les clauses `Group by` et la sélectivité des attributs des clauses `Where`.