Une répartition statique et dynamique de l'espace entre les vues matérialisées et les index dans les entrepôts de données

Ladjel Bellatreche
LISI/ENSMA
86960 Futuroscope FRANCE
bellatreche@ensma.fr

Kamel Boukhalfa
Laboratoire Génie des Procédés
Université Laghouat - Algérie
k.boukhalfa@mail.lagh-univ.dz

Résumé : Dans les entrepôts de données, deux structures d'optimisation de requêtes sont diponibles: (1) des structures redondantes : les vues matérialisées et les index et (2) des structures non redondantes : la fragmentation horizontale, la fragmentation verticale, et le parallélisme. Généralement, la sélection des vues et des index se fait séquentiellement; l'administrateur d'un entrepôt de données sélectionne d'abord les vues à matérialiser (ou les index), et ensuite les index (ou les vues) en respectant la contrainte de stockage allouée aux deux structures. L'attribution de l'espace réalisée d'une manière uniforme ou aléatoire pourrait affecter la conception physique de l'entrepôt compte tenu des fortes interactions entre les deux structures. Dans ce papier, nous considérons le problème de répartition de l'espace entre les vues matérialisées et les index, dans l'objectif de réduire le coût d'exécution de requêtes et le coût de maintenance dans le contexte statique et dynamique. Nous présentons une approche itérative qui distribue automatiquement l'espace entre les vues et les index. Finalement, cette approche est évaluée en utilisant un exemple emprunté du papier d'Informix.
Mots-Clés : la sélection des vues matérialisées, la sélection des index, la conception physique des entrepôts.

1 Introduction

Les entrepôts de données collectent les informations de plusieurs sources afin de former une seule base de données. En conséquence, ces derniers peuvent nécessiter de grandes capacités de stockage. Le processus d'analyse de données s'appuie souvent sur des requêtes qui groupent et filtrent des données de différentes formes. L'interrogation d'un entrepôt de données est complexe, et le traitement de ces requêtes peut prendre des heures, voire des jours vu le nombre de jointures et d'aggrégations. Pour accélérer l'exécution de ces requêtes, de nombreuses structures ont été proposées. Ces structures peuvent être divisées en deux catégories : les structures redondantes [1] (comme les vues matérialisées et les index) et les structures non redondantes (comme la fragmentation horizontale, verticale [2, 1, 11], et le traitement parallèle [13]). Dans ce papier nous nous intéressons aux structures redondantes. Rappelons qu'une vue matérialisée est une table contenant les résultats d'une requête. Certaines requêtes nécessitent seulement l'accès à la vue matérialisée [7] et sont ainsi exécutées plus rapidement.

Nombre de travaux récents concernant la sélection des vues matérialisées et la sélection des index ont été réalisés dans les entrepôts de données de type ROLAP (relational On-Line Analytical Processing). De nombreux systèmes commerciaux proposent la création et l'utilisation des vues matérialisées et des index [4, 5, 3, 15]. On peut citer Oracle10g, SQL server, et DB2, etc.

Conceptuellement, les vues et les index présentent certaines similitudes. En effet, il s'agit de structures *redondantes* qui se *partagent* la même ressource (*l'espace disque*). Toutes deux peuvent entraîner des *surcharges* causées par des opérations de mises à jour, et demandent un *temps de calcul élevé*. Une vue matérialisée est stockée sous forme d'une table relationnelle dans le contexte ROLAP, et son indexation peut toujours améliorer la performance des requêtes accédant à cette vue. La présence d'index peut rendre les vues matérialisées plus attractive et vice versa. Cependant, un outil de conception physique d'un entrepôt de données doit prendre en considération *l'interaction* entre les vues matérialisées et les index.

Dans la littérature, la sélection des vues et la sélection des index sont deux tâches *séquentielles* [15, 3] : *la sélection des vues passe avant la sélection des index* (appelée VIEWFIRST dans [12]) ou vice versa

[1] ces structures nécessitent un quota de stockage.

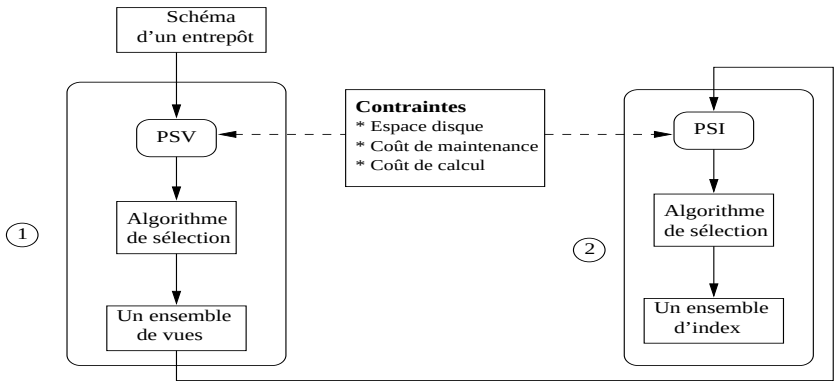

Figure 1: Le processus de sélection des vues et des index

(INDEXFIRST) (Figure 1). La sélection séquentielle présente deux inconvénients majeurs :

- *L'impossibilité de considérer plusieurs algorithmes de sélection :* Nous avons vu que les algorithmes de sélection des vues à matérialiser et des index sont contraints par une contrainte donnée (parmi les trois à savoir : l'espace disque, le coût de maintenance, le coût de calcul). Rappelons que le problème de sélection des vues matérialisées (PSV) et problème de sélection des index (PSI) sont formulés de la manière suivante : Le PSV consiste à sélectionner un ensemble de vues $\{V_1, V_2, ..., V_k\}$ minimisant/maximisant une fonction "objectif" et satisfaisant une contrainte A. Le PSI sélectionne un ensemble d'index $\{I_1, I_2, ..., I_{k'}\}$ minimisant/maximisant une fonction "objectif" et satisfaisant une contrainte B.
  Étant donné que les deux contraintes A et B peuvent être semblables ou différentes, la sélection séquentielle peut entraîner la considération de 9 problèmes de sélection correspondant à la combinaison de trois contraintes. Nous pouvons trouver, par exemple, un algorithme de sélection de vues contraint par l'espace disque, et un algorithme de sélection des index contraint par le coût de maintenance, et ainsi de suite.

- *La non prise en considération de l'interaction entre les vues et les index :* dans le cas où les deux problèmes ont la même contrainte (c'est-à-dire, $A = B$), trois formulations séquentielles de PSV et PVI sont possibles. Ignorer l'interaction entre les vues et les index entraîne *un problème de gestion des ressources (représentant les contraintes) entre les vues et les index.* Pour éclairer ce point, supposons que l'administrateur de l'entrepôt de données dispose d'une capacité de stockage S destinée aux vues et aux index. Étant donné que les index et les vues sont deux structures en compétition sur la même ressource (l'espace disque), la question qui se pose est de savoir *comment cette capacité est distribuée d'une manière automatique entre eux afin de garantir une meilleure performance pour les requêtes ?.*

  Une mauvaise distribution peut compromettre la qualité des solutions obtenues par les algorithmes de sélection des vues et des index.

Dans ce papier, nous nous attachons au problème de la gestion de la ressource représentant l'espace disque. Autrementdit le problème de distribution de l'espace disque entre deux structures redondantes, à savoir les vues matérialisées et les index, avec pour objectif l'amélioration de la performance des requêtes.

2 Motivations

Le problème de distribution de l'espace entre les vues et les index représente un enjeu important pour les chercheurs et les industriels. En effet, il peut avoir un effet crucial sur la performance des requêtes et poser des problèmes énormes à l'administrateur dans les cas statique et dynamique, comme nous allons le voir dans les sections qui suivent.

2.1 Le cas statique

Pour illustrer ces difficultés, considérons le scénario suivant : soit S l'espace disque global que l'administrateur possède pour sélectionner un ensemble de vues V et un ensemble d'index I. Pour répartir S entre V et I, l'administrateur a trois possibilités:

1. Attribuer la totalité de l'espace disque S aux vues matérialisées seules. Toutes les requêtes décisionnelles sont alors exprimées en fonction des vues et des relations de base. Cette attribution n'est pas efficace. En effet, plusieurs travaux ont démontré que des vues matérialisées seules sont insuffisantes pour accélérer toutes les requêtes décisionnelles [2, 10, 12]. Cette possibilité est de ce fait écartée.

2. Attribuer la totalité de S aux index seuls. Toutes les requêtes décisionnelles sont maintenant traitées en fonction des index et des relations de base. Or, interroger un entrepôt de données sans vues matérialisées ne garantit aucune performance [9, 8, 14]. En conséquence, les index seuls sont insuffisants pour garantir une bonne performance des requêtes décisionnelles. Cette possibilité est donc également écartée.

3. La dernière possibilité consiste en un compromis entre les deux premières possibilités. Ce compromis devrait garantir une bonne performance et de ce fait est l'objet de notre étude.

   Plus formellement, l'administrateur doit déterminer une fraction f ($0 \leq f \leq 1$) tel que le quota d'espace définie par $f \times S$ soit réservé aux vues matérialisées (ou aux index) et le quota d'espace défini par $(1 - f) \times S$ soit réservé aux index (ou aux vues).
   Trois possibilités de distribution sont possibles: (i) une distribution *uniforme* qui présente une répartition équitable de S entre les vues et les index ($f = 0.5$), (ii) une distribution *aléatoire* qui présente une répartition quelconque de S et (iii) une distribution *calculée* qui consiste en l'attribution de l'espace aux vues et aux index, en respectant certains critères de performance.

   Certes, les distributions uniforme et aléatoire sont faciles à obtenir, mais elles présentent plusieurs inconvénients : elles ne prennent pas en compte l'interdépendance mutuelle entre les vues et les index (Figure 1). La distribution aléatoire peut allouer plus d'espace que nécessaire aux vues (ou aux index). Cette allocation se fait au détriment de l'espace des index et vice versa. Notons que pour une exécution efficace des requêtes, il est parfois préférable d'avoir plus d'espace pour les vues que pour les index (ou vice-versa).

4. Ces deux distributions ne possèdent aucune métrique garantissant la réduction du coût de l'ensemble de requêtes.

2.2 Le cas dynamique

Étant donné que l'entrepôt de données est un environnement dynamique, le problème de redistribution d'espace entre les vues et les index doit être considéré après les opérations de mise à jour. Ces dernières interviennent au niveau des tables des sources de l'entrepôt, et les changements correspondants doivent être répercutés sur les vues matérialisées et les index. Ainsi les tailles des vues et des index peuvent augmenter ou diminuer. Il est donc nécessaire de revoir la distribution initiale de l'espace global entre les vues et les index afin de garantir une meilleure performance. Un autre problème pouvant entraîner une redistribution de l'espace est le changement des requêtes de départ (ajout/suppression de requêtes, changement de fréquence d'accès des requêtes, etc). Cela est dû au fait que la plupart des algorithmes de sélection des vues et des index sont basés sur des requêtes connues a priori.

Dans ce papier, nous proposons des solutions au problème de distribution de l'espace entre les vues et les index dans les contextes *statique* et *dynamique*. Les contributions principales de ce papier sont : (1) la présentation d'une méthodologie itérative de distribution efficace de l'espace entre les vues et les index, (2) l'évaluation de cette méthodologie et l'illustration de son utilité.

3 L'intuition de notre approche de distribution

L'objectif principal à satisfaire pour la sélection des vues matérialisées et la sélection d'index est *la minimisation du coût total d'évaluation des requêtes décisionnelles* [1, 14, 8, 3]. Notre solution de distribution de l'espace entre les vues et les index *doit également satisfaire cet objectif.*

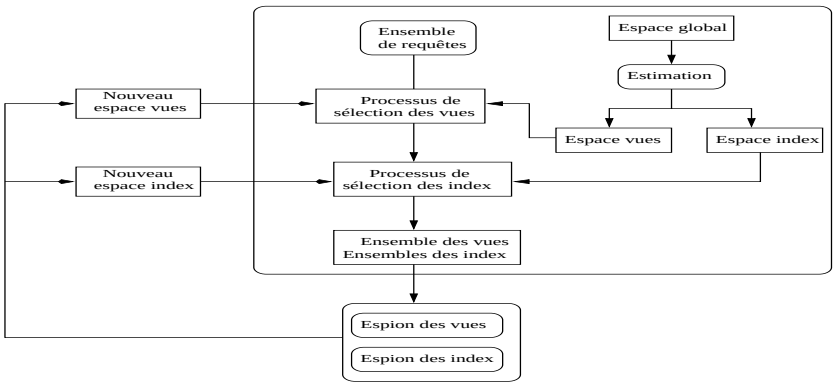

Figure 2: L'intuition de notre approche itérative

L'intuition sous-jacente de notre approche est la suivante : initialement, l'administrateur estime les deux quotas d'espace S^V et S^I pour les vues matérialisées et les index respectivement, ($S = S^V + S^I$). Cette estimation peut être établie en utilisant une distribution uniforme ou aléatoire. En fonction du quota réservé aux vues S^V, l'administrateur sélectionne un ensemble de vues $V = \{V_1, V_2, ..., V_s\}$ en utilisant l'un des algorithmes de sélection de vues ayant le capacité de stockage comme contrainte [14].

Une fois les vues sélectionnées, un ensemble d'index I est construit $I = \{I_1, ..., I_r\}$ en utilisant le quota réservé aux index (S^I). En conséquence, nous obtenons une solution initiale pour le problème de distribution statique, qui consiste en un ensemble de vues V et un ensemble d'index I. Toutes les requêtes de départ sont donc exécutées en utilisant ces deux ensembles. À l'aide d'un modèle de coût, le coût d'évaluation de l'ensemble des requêtes est calculé.

Le principe de notre approche est de *reconsidérer itérativement* cette solution initiale dans le but de réduire le plus possible le coût d'évaluation des requêtes. Elle s'appuie sur deux agents, *l'espion des index* et *l'espion des vues* dont les rôles sont les suivants :

- l'espion des index a pour tâche de voler de l'espace réservé aux vues. L'espace ainsi récupéré sera utilisé pour créer d'autres index et les vues correspondantes seront supprimées. L'opération est validée si le coût total d'exécution des requêtes diminue.
- d'une manière similaire, l'espion des vues a pour tâche de dérober de l'espace réservé aux index afin de créer d'autre vues dans le but de minimiser le coût total d'exécution des requêtes.

Notre algorithme organise donc une sorte de "combat" entre ces deux espions. Ce combat est pour la bonne cause: *la meilleure utilisation de l'espace disque disponible dans l'objectif de minimiser le coût d'évaluation des requêtes.*

L'algorithme s'achève lorsque les deux espions ne peuvent plus réduire le coût total d'exécution des requêtes. À la fin de cet algorithme, nous obtenons deux résultats principaux : (1) Un ensemble de vues matérialisées $V' = \{V'_1, V'_2, ..., V'_{s'}\}$ et un ensemble d'index $I' = \{I'_1, I'_2, ..., I'_{r'}\}$ garantissant un coût minimal d'exécution des requêtes. Ces deux ensembles peuvent être différents des deux ensembles de départ, et (2) de nouveaux quotas d'espace pour les vues matérialisées $S^{V'}$ et les index $S^{I'}$ tels que : $S^{V'} \neq S^V$ et $S^{I'} \neq S^I$.

3.1 Un exemple d'illustration

Nous illustrons maintenant notre algorithme par un exemple. Prenons le schéma en étoile de la Figure 3 et supposons que nous ayons cinq requêtes fréquentes dont les graphes sont représentés en Figure 4. Les cinq graphes de requêtes représentant les cinq requêtes sont fusionnés en un seul graphe appelé plan multiple d'exécution des vues (PMEV) [14]. Supposons que l'administrateur dispose de 800 MB et ait affecté **600 MB** aux vues matérialisées et **200 MB** aux index.

Premièrement, nous exécutons l'algorithme *sélect_vue* présenté dans [14, 2] pour sélectionner des vues en utilisant les 600 MB comme capacité de stockage. Les vues sélectionnées sont V_1, V_2 et V_3 comme le montre la Table 1. Le coût total d'évaluation de l'ensemble de requêtes (en nombre des entrées-sorties) est égal à **4054825**.

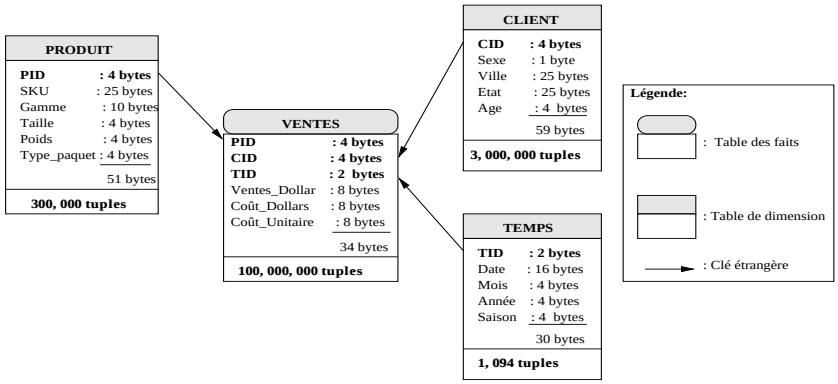

Figure 3: Un exemple d'un schéma en étoile

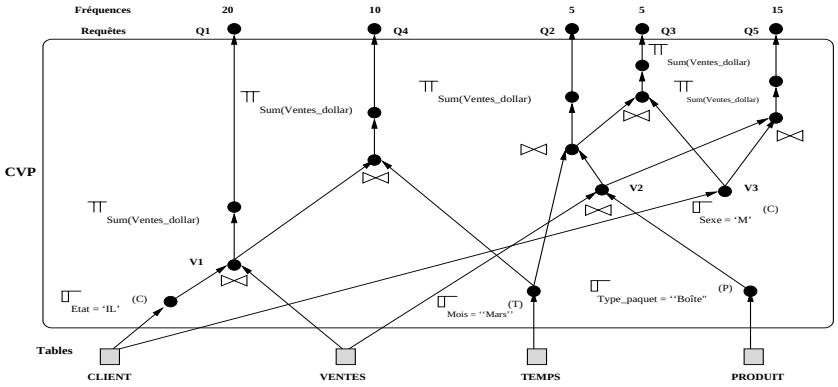

Figure 4: Le PMEV pour cinq requêtes

Nous construisons ensuite les index (en utilisant les 200 MB) afin de réduire encore le coût obtenu par la présence des trois vues matérialisées. Les cinq requêtes sont réécrites en fonction de V_1, V_2, V_3, et des deux tables de dimensions CLIENT et TEMPS. En exécutant l'algorithme de sélection des index proposé dans [2], un index de jointure entre la vue V_2 et la table de dimension CLIENT noté par ($V_2 \sim CLIENT$) est sélectionné. La quantité d'espace nécessaire pour cet index est 181821440 bytes. La présence de cet index réduit le coût total d'évaluation des requêtes de **4054825** à **3263405**.

Maintenant, l'espion des index tente de dérober de l'espace attribué aux vues matérialisées afin d'ajouter des index. Puisque la vue V_3 est de petite taille et qu'elle est moins utilisée que les autres vues (V_1 et V_2), son espace (89047040 bytes) est convoité par l'espion des index. L'espace attribué aux index sera donc de 289047040 bytes au total. Après l'élimination de la vue V_3, il ne reste plus que V_1, V_2, CLIENT et TEMPS pour exécuter les cinq requêtes. En utilisant un nouvel espace, l'espion des index sélectionne un nouvel index de jointure ($V_1 \sim V_2 \sim TEMPS \sim CLIENT$). Cet index réduit le coût total d'exécution des requêtes de **3263405** à **1955305**.

Finalement, aucun des deux espions ne peut plus obtenir de réduction, d'où la fin de l'algorithme.

Vues	**Définition de vues**	**Tailles (bytes)**
V_1	VENTES $\bowtie \sigma_{Etat='IL'}$ (CLIENT)	178 094 080
V_2	VENTES $\bowtie \sigma_{Type\_paquet='Boite'}$ (PRODUIT)	324 435 968
V_3	$\sigma_{Sexe='M'}$ (CLIENT)	89 047 040
Total		591 577 088

Table 1: Les vues matérialisées sélectionnées et leurs tailles

L'ensemble des vues matérialisées et l'ensemble des index deviennent respectivement $V = \{V_1, V_2\}$ et $I = \{(V_1 \sim V_2 \sim TEMPS \sim CLIENT)\}$. 500 MB sont attribués aux vues matérialisées et 300 MB aux index. Le coût total d'exécution de l'ensemble de requêtes est de 1955305 opérations d'entrées-sorties. Le déroulement de cet algorithme est résumé par la Table 2.

Etape 1	*Etape 2*	*Etape 3*	*Etape 4*	*Etape 5*
Sans vues & index	Avec vues	Avec vues et index	Espion des index	Espion des vues
68514928	4054825	3263405	1955305	1955305 Algorithme terminé

Table 2: Réduction du coût dû à l'espion des index

4 Le problème de la distribution statique

Dans cette section, nous considérons le problème de distribution d'espace dans le cas statique. Tous les paramètres de l'entrepôt (les vues, les index, les tables de bases, etc.) sont connus a priori et restent inchangés. Pour décrire notre approche, nous considérons un entrepôt modélisé par un schéma en étoile.

4.1 La formulation du problème

Le problème de distribution de l'espace entre les vues matérialisées et les index est formulé de la manière suivante : soient un entrepôt de données modélisé par un schéma en étoile, $Q = \{Q_1, Q_2, ..., Q_n\}$ un ensemble de requêtes d'interrogation (les plus fréquentes) avec leurs fréquences d'accès $F = \{f_1, f_2, ..., f_n\}$ et S l'espace disque destiné à stocker les vues matérialisées V et les index I, pour supporter l'ensemble des requêtes Q. L'objectif du problème de distribution de l'espace est de répartir l'espace global S entre les vues (V) et les index (I) afin de minimiser le coût d'évaluation de l'ensemble des requêtes.

4.2 Architecture de notre approche

L'architecture de notre approche de distribution de l'espace est présentée dans la Figure 2. Les composantes de cette architecture sont: (1) un espion des vues, (2) un espion des index, (3) un contrôle d'admission de vol, (4) un contrôleur d'espion, (5) un algorithme de sélection des vues, (6) un algorithme de sélection des index, et (7) un modèle de coût. Chaque composante a sa propre tâche :

1. L'espion des vues et l'espion des index : Comme nous l'avons déjà mentionné, les espions des index et des vues ont pour tâche de dérober l'espace réservé aux vues matérialisées (aux index) afin de créer d'autre(s) index (vues) (voir section 3).
2. Le contrôle d'admission de vol : Il vérifie que chaque tentative de vol d'un espion (des index ou des vues) contribue à la réduction du coût des requêtes. Cette décision est basée sur un modèle de coût. Pour clarifier le rôle de contrôle d'admission de vol, soient V et I les ensembles des vues et des index déjà sélectionnés. Soit CT_p le coût d'évaluation de l'ensemble de requêtes en présence de $\mathcal{V}$ et de $\mathcal{I}$ (avant la tentative de vol). Supposons que l'espion des vues tente l'opération de vol et que d'autres vues soient créées. Soient $\mathcal{V}'$ le nouvel ensemble de vues et $\mathcal{I}'$ le nouvel ensemble des index. Le contrôle d'admission de vol évalue alors le nouveau coût d'exécution des requêtes en présence de $\mathcal{V}'$ et de $\mathcal{I}'$ qui est noté par CT_c. Si ce dernier est inférieur à CT_p, la tentative de vol est validée par le contrôle d'admission de vol, sinon elle ne l'est pas.
3. Le contrôleur d'espion : Son rôle est d'assurer qu'à chaque moment de l'algorithme, un seul espion est actif.
4. Un algorithme de sélection des vues : Il représente l'algorithme de sélection des vues matérialisées utilisé. Dans cette étude, nous utilisons l'algorithme *sélect_vue* présenté dans [14, 2].
5. Un algorithme de sélection des index : Il représente l'algorithme de sélection des index utilisé. Dans cette étude, nous utilisons les techniques d'indexation présentées dans [2].
6. Le modèle de coût : Il constitue le coeur de notre approche. Il est utilisé par chacune des composantes et quantifie la solution obtenue. Le modèle de coût utilisé calcule le nombre d'entrées sorties. Par faute d'espace ce modèle ne sera pas présenté (pour plus de détails, voir [2]).

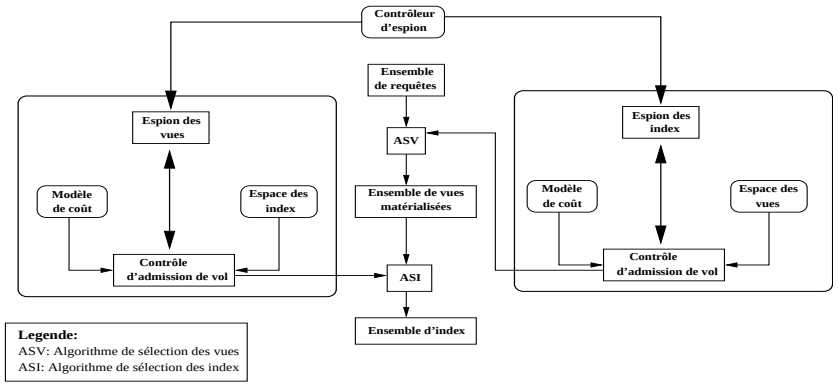

Figure 5: Les éléments de notre algorithme statique et leurs interactions

4.3 Les étapes de notre algorithme

Rappelons que l'algorithme de distribution est basé sur une approche itérative dont les étapes principales sont les suivantes: (1) la détermination d'une solution initiale, (2) le mécanisme d'amélioration de la solution initiale et (3) le test d'arrêt. Dans les sections suivantes, nous décrivons plus en détail chaque étape.

4.4 La solution initiale

Pour amorcer notre algorithme, nous suivons la sélection séquentielle des vues et des index. Nous estimons (d'une manière uniforme ou aléatoire) les deux quotas de l'espace des vues matérialisées S^V et l'espace des index S^I. Tout d'abord, nous exécutons un algorithme de sélection des vues à matérialiser dirigé par l'espace S^V [14]. Comme résultat, nous obtenons un ensemble de vues $V = \{V_1, V_2, ..., V_m\}$. Ensuite, en utilisant le quota de l'espace réservé aux index, nous sélectionnons un ensemble d'index $I = \{I_1, I_2, ..., I_l\}$. Finalement, nous calculons le coût d'évaluation de l'ensemble des requêtes en présence des ensembles de vues (V) et des index (I) noté par $CTER_i$.

4.5 Le processus d'amélioration de la solution initiale

Pour réaliser cette amélioration, l'algorithme utilise les deux espions décrits précédemment. La tâche des ces espions est contrôlée par les règles suivantes : Règle 1 : un espion doit être capable d'utiliser l'espace volé, c'est-à-dire qu'il doit trouver une vue (pour l'espion d'index) ou un index (pour l'espion des vues) à ranger dans l'espace volé, Règle 2 : l'espion ne peut voler que si le coût global d'exécution des requêtes est réduit après l'utilisation de l'espace volé et la Règle 3 : un seul espion est actif à la fois.

4.5.1 Les structures utilisées dans notre algorithme

Les structures de données nécessaires à l'algorithme itératif sont deux matrices appelées: *la matrice de réécriture* et *la matrice requête-index*.

Les lignes et les colonnes de la matrice de réécriture ($W = w_{ij}$) représentent les requêtes et les vues (et tables), respectivement. Ses valeurs w_{ij} sont définies comme suit: $\mathbf{w_{ij}} = 1$ si la requête Q_i utilise la vue (table) j, 0 sinon. Nous ajoutons une autre colonne à cette matrice représentant la fréquence de chaque requête (voir Figure 6). Cette matrice sert à identifier les relations de base ou les vues utilisées pour une requête donnée. Nous attribuons à chaque vue V_i une *fréquence d'accès* dénotée par f_{V_i} obtenue par la somme des fréquences des requêtes accédant à cette vue : $f_{V_i} = \sum_{j=1}^{n} w_{ij} * f_j$. Les victimes potentielles de l'espion des index sont les vues figurant dans cette matrice.

La deuxième matrice, que nous appelons requête-index $RI = \{ri_{lk} \mid 1 \leq l \leq n, 1 \leq k \leq m\}$, est telle que : $\mathbf{ri_{lk}} = 1$ si la requête Q_l utilise l'index I_k, 0 sinon. D'une manière similaire, nous ajoutons une nouvelle colonne pour les fréquences et nous attribuons à chaque index I_k une *fréquence d'accès* dénotée par f_{I_k}, obtenue par la sommation des fréquences des requêtes : $f_{I_k} = \sum_{j=1}^{n} ri_{kj} * f_j$. Les victimes potentielles de l'espion des vues sont les index figurant dans cette matrice.

	V_1	V_2	V_3	T	FQ
Q_1	1	0	0	0	20
Q_2	0	1	0	1	5
Q_3	0	1	1	1	5
Q_4	1	0	0	1	10
Q_5	0	1	1	0	15

Figure 6: La matrice de réécriture initiale

	V_1	V_2	C	T	FQ
Q_1	1	0	0	0	20
Q_2	0	1	0	1	5
Q_3	0	1	1	1	5
Q_4	1	0	0	1	10
Q_5	0	1	1	0	15

Figure 7: La matrice de réécriture après l'exécution de l'espion des index

Notre algorithme utilise deux procédures: espion-index-vol-espace-vue et espion-vue-vol-espace-index, que nous décrivons dans les sections qui suivent.

Procédure espion-index-vol-espace-vue Comme son nom l'indique, cette procédure décrit la tâche de l'espion des index. Elle présente en entrée les éléments suivants : (i) un ensemble de requêtes d'interrogation, (ii) un ensemble de vues matérialisées $V = \{V_1, V_2, ..., V_l\}$ et un ensemble d'index $I = \{I_1, I_2, ..., I_m\}$, obtenus par la solution initiale; et (iii) les deux quotas de stockage S^V (pour les vues) et S^I (pour les index). Comme sortie, cette procédure génère un ensemble d'index $I' = \{I'_1, I'_2, ..., I'_{m'}\}$ qui peut être soit identique à l'ensemble initial I soit différent et plus intéressant en termes de réduction du coût des requêtes.

Pour identifier les victimes des vues matérialisées, l'espion des index doit avoir une politique de sélection de vue à éliminer pour créer de nouveaux index en garantissant une bonne performance. Nous suggérons deux politiques : *la Vue la Moins Fréquemment Utilisée (VMFU)* et *la Vue de plus Petite Taille (VPPT)*. Dans *La VMFU:* , l'espion des index doit sélectionner la vue la moins fréquemment utilisée. Cette vue est celle qui contribue le moins à l'amélioration du coût d'exécution des requêtes. Dans la *VPPT*, l'espion des index doit sélectionner la vue ayant la plus petite taille. En effet, la taille des vues étant en moyenne plus grande que la taille des index, une petite vue pourra être remplacée par un index.
Une fois la vue victime éliminée, les requêtes de départ doivent être réécrites en fonction des vues restantes. D'où la mise à jour de la matrice de réécriture.

Exemple 1 *Cet exemple illustre le fonctionnement de l'espion des index. Supposons que nous ayons trois vues matérialisées V_1, V_2 et V_3 comme dans l'exemple d'illustration occupant 600 MB et un index ($V_2 \sim CLIENT$). À partir de ces vues et index, nous construisons la matrice de réécriture W (voir Figure 6). La fréquence d'accès de chaque vue V_i $(1 \leq i \leq 3)$ est calculée: $f_{V_1} = 30$, $f_{V_2} = 25$ et $f_{V_3} = 20$.*

Étant donné que l'espion des index élimine la vue V_3 qui est la moins fréquente, la matrice de réécriture doit être modifiée de la manière suivante: la vue V_3 est remplacée par la table de dimension CLIENT comme le montre la Figure 7.

4.5.2 Algorithme de l'espion des index

La description détaillée de la tâche de l'espion des index est donnée dans l'algorithme 1.

Procédure espion-vues-vol-espace-index Cette procédure fonctionne comme celle de l'espion des index. Elle possède les mêmes entrées que la procédure espion-index-vol-espace-vue. Comme sortie, elle génère un ensemble de vues $\mathcal{V}' = \{V'_1, V'_2, ..., V'_{l'}\}$ qui peut être soit identique à l'ensemble initial V soit plus intéressant par rapport à V en termes de réduction de coût des requêtes.

Pour sélectionner les victimes, l'espion des vues possède deux stratégies: *l'Index le Moins Fréquemment Utilisé (IMFU)* et *le Plus Grand Index (PGI)*. Dans l'IMFU, l'espion des vues doit sélectionner l'index le moins fréquemment utilisé. Cet index est celui qui contribue le moins à la réduction du coût d'exécution des requêtes. Dans la PGI, le plus grand index est sélectionné parce que la taille des vues est en moyenne plus grande que la taille des index. En effet, un index de petite taille ne présente pas en général suffisamment d'espace pour stocker une nouvelle vue.

Algorithme 1 : La procédure espion-index-vol-espace-vue

Calculer le coût global G_i d'évaluation des requêtes en utilisant les vues et les index;
Sélectionner une vue V_i en utilisant une stratégie parmi VMFU ou VPT ;
$S^I = S^I + espace(V_i)$ + l'espace inutilisé ;
$S^V = S^V - espace(V_i)$;
Éliminer la vue V_i;
Mettre à jour la matrice de réécriture W;
Exécuter ASI (on obtient un nouvel ensemble d'index) ;
Calculer le coût global G_c d'évaluation des requêtes en utilisant le nouvel ensemble des index et les vues matérialisées restantes;
si $(G_c < G_i)$ **alors**
  Mettre à jour la matrice requête-index RI
sinon
  Restaurer la vue éliminée;
  Rétablir S^V, S^I et G_c à leur valeurs précédentes;
  Rétablir la matrice W à son état précédent

4.5.3 Algorithme de l'espion des vues

Les étapes principales de l'espion des vues sont illustrées par l'algorithme 2.

Algorithme 2 : La procédure espion-vues-vol-espace-index

Calculer le coût global G_i d'évaluation des requêtes en utilisant les vues et les index Sélectionner un index I_i en utilisant une stratégie parmi IMFU ou PGI ;
$S^V = S^V + espace(I_i)$ + espace inutilisé ;
$S^I = S^I - space(I_i)$;
Éliminer I_i;
Mettre à jour la matrice RI ;
Exécuter Select_Vues (il fournit un nouvel ensemble de vues matérialisées);
Exécuter l'algorithme ASI;
Calculer le coût global G_c d'évaluation des requêtes en utilisant le nouvel ensemble des index et les vues matérialisées;
si $(G_c < G_i)$ **alors**
  Mettre à jour les deux matrices W et RI
sinon
  Restaurer l'index éliminé;
  Rétablir S^V, S^I et G_c à leur valeurs précédentes;
  Rétablir la matrice RI à son état précédent ;

4.6 Le test d'arrêt

Nous exécutons ces deux procédures jusqu'à ce qu'aucune réduction du coût global d'évaluation de l'ensemble de requêtes ne soit constatée.

4.7 La détermination de l'espion privilégié

L'objectif principal de notre algorithme est d'améliorer la solution initiale obtenue par l'estimation de l'espace. Notons que nous possédons deux espions, chacun ayant le droit de commencer son vol. Cependant, ils ne peuvent pas commencer simultanément (voir la règle 3). Un espion doit donc recevoir le privilège de commencer le vol. Reste à identifier cet espion, car cette identification peut avoir des effets importants sur la qualité de la solution obtenue par notre algorithme.

Pour réaliser la tâche d'identification, nous suggérons un critère de sélection basé sur la notion de *contribution de chaque espion* définie par la suite dans cette section.

Soient CT_s, CT_v et CT_i le coût total d'évaluation de l'ensemble des requêtes de départ sans les vues et les index, en présence seulement des vues, ou en présence seulement des index.
Nous définissons la **contribution de l'espion des vues** $cont(V)$ par $cont(V) = CT_s - CT_v$. Elle nous donne la réduction du coût d'évaluation des requêtes obtenue par la sélection des vues matérialisées. D'une manière

similaire, nous définissons **la contribution de l'espion des index** $cont(I)$ par $cont(I) = CT_s - CT_i$. Cette contribution nous offre une réduction du coût d'évaluation des requêtes obtenue par la sélection des index.

En utilisant ces deux contributions, nous définissons l'espion à privilégier comme l'espion ayant la plus grande contribution. Plus formellement, il est défini de la manière suivante:

$$\text{l'espion à privilégier} = \begin{cases} \text{l'espion des vues si } cont(V) < cont(I) \\ \text{l'espion des index, sinon} \end{cases}$$

En cas d'égalité des contributions, nous sélectionnons un espion au hasard.

4.8 Description de l'algorithme

Maintenant, nous avons tous les ingrédients pour définir l'algorithme itératif distribuant l'espace entre les vues et les index. Il commence par la recherche d'une solution initiale. Une fois cette solution trouvée, il construit les deux matrices W et RI. En fonction de la contribution de chaque espion, l'algorithme active l'espion privilégié suivi par l'autre espion. Le processus d'activation des espions est répété jusqu'à ce qu'aucune réduction au niveau du coût total d'évaluation des requêtes ne soit plus constatée.

Les étapes principales de notre approche sont décrites dans l'algorithme 3.

Algorithme 3 : Algorithme statique

Initialisation: les deux matrice (W et RI) sont initialisées à zéro ;
Exécuter Select_Vues en respectant la contrainte d'espace S^V;
Construire la matrice W en fonction de l'ensemble des vues sélectionnées ;
Exécuter ASI en respectant la contrainte d'espace S^I;
Construire la matrice RI en fonction de l'ensemble des index sélectionnés ;
Activer espion-index-vol-espace-vues et espion-vues-vol-espace-index (le choix dépend de leurs contributions respectives);
Répéter l'étape 6 jusqu'à ce qu'aucune réduction du coût total d'évaluation des requêtes ne soit constatée;

Notre algorithme se termine parce qu'il est contrôlé par un modèle de coût.

5 La problème de la distribution dynamique

Dans la section précédente, nous avons étudié le problème de distribution de l'espace dans le contexte statique. Dans cette section, nous traitons ce problème dans le cas dynamique en adaptant l'algorithme statique.

5.1 Position du problème

Rappelons que les entrepôts sont construits en collectant les informations de plusieurs sources et en les intégrant dans une seule base de données. Les entrepôts sont volumineux pour des applications fonctionnant dans des environnements de grande échelle composées de multiples sources hétérogènes, tel que le Web. De tels environnements sont souvent marqués par des changements dynamiques au niveau des sources d'informations, modifiant non seulement les contenus des données, mais également leurs schémas et leurs capacités de requêtes. Ces changements peuvent provoquer des modifications des vues et des index définis sur l'entrepôt.

Quelle que soit la nature du changement, il peut avoir une forte incidence sur l'ensemble des vues et des index. Une vue après la maintenance peut occuper plus ou moins d'espace par rapport à son espace initial. Ceci est valable également pour les index. La contrainte d'espace risque de ne plus être respectée. Si l'espace total occupé par les vues et les index est plus grand (respectivement plus petit) que l'espace demandé, il est nécessaire de *déterminer* quelle vue ou quel index doit être éliminé (ou ajouté) [2]. Notre approche itérative peut facilement être adaptée pour traiter ce problème en ajustant les espaces réservés aux vues et aux index. Dans la section suivante, nous décrivons les étapes de l'algorithme que nous appelons *répartition_dynamique*.

[2]Notre approche ne prend pas en considération le coût de création et de suppression d'une vue et d'un index. Car notre objectif principal est de satisfaire la contrainte de l'espace. Il serait cependant intéressant de combiner la contrainte de l'espace et la contrainte du coût de calcul pour le problème de redistribution de l'espace entre les vues et les index.

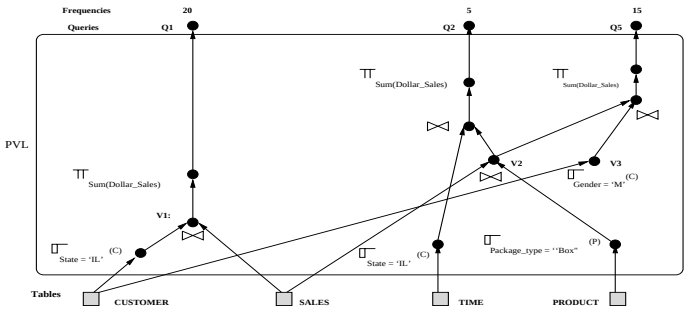

Figure 8: Le PMEV pour les requêtes: Q_1, Q_2 et Q_3

5.2 L'algorithme répartition_dynamique

Soient S_i^V et S_i^I les espaces occupés par les vues et les index avant les opérations de mise à jour. Après ces dernières, l'ensemble des vues matérialisées V et l'ensemble des index I occupent de nouveaux quotas d'espace nommés S_c^V et S_c^I, respectivement. Soit S_i $(= S_i^V + S_i^I)$ et S_c $(= S_c^V + S_c^I)$ les espaces totaux avant et après les opérations de mises à jour. Notre algorithme exécute les étapes suivantes pour redistribuer et gérer les nouveaux quotas d'espace :

1. Calculer les nouveaux quotas de l'espace disque occupé par les vues (S_c^V) et les index (S_c^I)
2. Si l'espace total S_c diminue (en cas d'opération de suppression par exemple), nous aurons de l'espace non utilisé qui pourrait être utilisé par les vues, les index ou les deux. Dans le cas où cet espace est utilisé pour la sélection d'autres vues, l'algorithme cherche des vues appartenant à la couche des vues potentielles (CVP) (voir Figure 4). Dans le cas où cet espace est utilisé pour sélectionner d'autres index, l'algorithme cherche à les sélectionner d'une manière incrémentale en utilisant l'algorithme présenté dans [2].
3. Si l'espace global S_c augmente (en cas de création par exemple) et devient plus grand que l'espace initial S_i, il faut libérer de l'espace. On peut considérer différents cas :
   - **Augmentation de la taille des vues seulement :** Si l'espace des vues S_c^V dépasse son quota initial (S_i^V), nous devons éliminer certaine(s) vues afin de satisfaire la contrainte d'espace. Mais avant d'entamer le processus d'élimination, nous devons tester si l'espion des vues peut voler de l'espace réservé aux index et garder ces vues (si ces vues contribuent fortement à réduction du coût total des requêtes). Si la tentative de vol de l'espion des vues échoue, nous éliminons certaines vues en utilisant une politique parmi VMFU et VPPT.
   - **Augmentation de la taille des index seulement :** Si l'espace des index S_c^I dépasse son quota initial (S_i^I), comme dans le cas précédent, nous testons si l'espion des index peut voler de l'espace réservé aux vues et garder ces index. Si cette tentative échoue, nous éliminons les index en utilisant une politique parmi IMFU et PGI.
   - **Augmentation de la taille des vues et les index :** Si les deux espaces sont plus grands que leurs espaces d'origine, nous éliminons des vues et des index en utilisant les politiques des vues et des index jusqu'à ce que la contrainte d'espace soit satisfaite.

6 Evaluation

Notre expérimentation utilise le schéma en étoile de la Figure 3. Nous supposons que la taille de la page est de 8192 bytes et que la taille du pointeur de bloc est de 8 bytes. À partir de ces informations et des cardinalités de chaque table, nous pouvons facilement calculer la taille de chaque table.

Pour évaluer notre algorithme, nous utilisons deux ensembles de requêtes: le premier est l'ensemble des cinq requêtes $\{Q_1, Q_2, Q_3, Q_4, Q_5\}$ (Figure 4) et le deuxième est l'ensemble des trois premières requêtes $\{Q_1, Q_2, Q_3\}$. Les PMEVs, pour le premier et le deuxième ensemble de requêtes sont illustrés par les Figures 4 et 8, respectivement. Pour le premier ensemble, la capacité de stockage $S^V = 600$ MB pour les vues

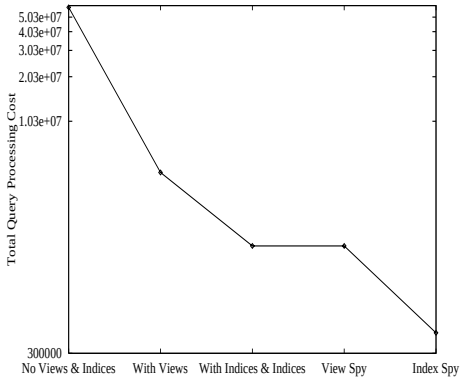

Figure 9: Effets des espions pour les 5 requêtes

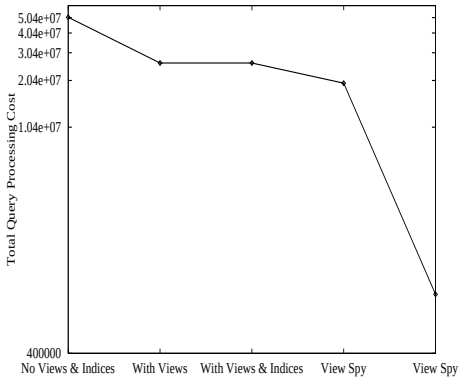

Figure 10: Effets des espions pour les 3 requêtes

matérialisées et $S^I = 200$ MB pour les index. Pour le deuxième ensemble, ces valeurs sont: $S^V = 200$ MB and $S^I = 600$ MB, respectivement.

Solution	**Vue & Index**	**CT**	S^V	S^I
Solution initiale	Sans vue & index	68514928	0	0
Avec vues	V_1: V $\bowtie \sigma_{Etat='IL'}$ C V_2: V $\bowtie \sigma_{Type\_paquet='Boite'}$ P V_3: $\sigma_{Sexe='M'}$ C	4054825	600	0
avec vues et index	V_1, V_2, V_3 & I_1 ($V_2 \sim V_3$)	3263405	600	200
Espion des vues		3263405	600	200
Espion des index	$V_1 \sim V_2 \sim T \sim C$	1955305	500	300

Table 3: La description des solutions pour les cinq requêtes

Solution	**Vues et index**	**CT**	S^V	S^I
Solution initiale	Sans vue et index	37371780	0	0
Avec vue	V_1: V $\bowtie \sigma_{Etat='IL'}$ C	12892060	178	0
Avec vues et index		12892060	178	600
Espion des vues	V_2: V $\bowtie \sigma_{Mois="Mars"}$ T	2017640	345	433
Espion des vues	V_3: V $\bowtie \sigma_{Type\_paquet='Boite'}$ P	1274237	667	109

Table 4: La description des solutions pour les trois requêtes

Pour les deux ensembles de requêtes, le coût d'évaluation des requêtes est d'abord calculé sans considérer les vues et les index (avec la technique de hachage). Nous sélectionnons un ensemble de vues en utilisant l'algorithme *sélect_vue*. Ces vues réduisent d'une manière significative le coût initial obtenu par la technique de hachage (Figure 9). Ensuite, nous sélectionnons des index réduisant le coût. Une fois la solution initiale obtenue, nous activons l'espion des vues car il présente une plus grande contribution que l'espion des index. Cet espion échoue dans la réduction du coût des requêtes (Figure 9). L'espion des index est alors activé et réduit le coût des requêtes. Les vues sélectionnées sont: V_1 et V_2, et l'index sélectionné est ($V_1 \sim V_2 \sim TEMPS \sim CLIENT$) (voir Table 3).

Dans la Table 4, nous présentons les résultats de notre algorithme pour le deuxième ensemble de requêtes. Nous constatons que le quota réservé aux index (600MB) n'est pas suffisant pour sélectionner des index. En conséquence, cet espace est utilisé par l'espion des vues qui sélectionne deux autres vues.

Dans cet exemple, nous pouvons faire l'observation suivante: l'approche espion contre espion distribue l'espace avec l'objectif de minimiser le coût d'évaluation des requêtes. Dans les deux cas, nous constatons une réduction significative du coût des requêtes. Nous notons que dans cette évaluation, le coût total d'évaluation

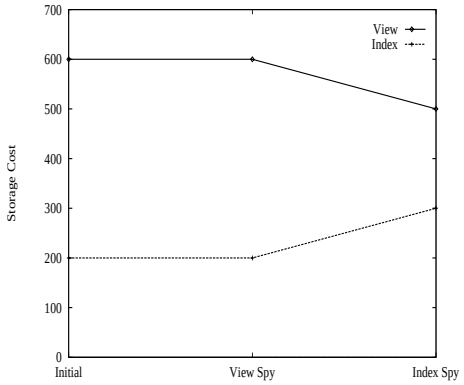

Figure 11: La distribution d'espace pour les 5 requêtes

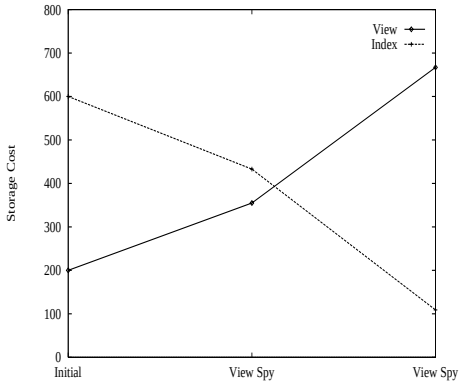

Figure 12: La distribution d'espace pour les 3 requêtes

des requêtes en utilisant la solution initiale (c'est-à-dire, sans l'application de notre approche) est trois fois plus coûteuse dans le cas des cinq requêtes et dix fois plus coûteuse dans le cas des trois requêtes. On constate aussi que pour les cinq requêtes l'espion des index **réduit** davantage le coût que l'espion des vues. Dans ce cas, l'espion des index vole 100 MB sur les 600 alloués aux vues matérialisées (Figure 11). Mais pour les trois requêtes, l'espion des vues **réduit** davantage le coût que l'espion des index. Dans ce cas, l'espion des vues vole 491 MB d'espace des 600 MB alloués aux index (Figure 12). On peut donc considérer que notre approche contribue effectivement à une meilleure distribution de l'espace disponible entre les vues et les index afin de réduire le coût total d'évaluation des requêtes.

Nous évaluons maintenant notre approche itérative dans le cas dynamique. Supposons que 500 000 nouveaux n-uplets soient ajoutés à la table des faits. Ils doivent être propagés dans les vues matérialisées et les index obtenus par l'algorithme statique (Table 3). Nous supposons que 40% de ces n-uplets sont propagés dans la vue V_1 (VENTES $\bowtie \sigma_{Etat='IL'}$ CLIENT) et la vue V_2 (VENTES $\bowtie \sigma_{Type\_paquet='Boite'}$ PRODUIT). Après ces mises à jour, l'espace total occupé par les vues et les index est de 860 MB et la contrainte de stockage est violée. Il est nécessaire de redistribuer l'espace entre les vues et les index. Puisque les deux vues matérialisées contribuent le plus à la réduction du coût d'évaluation des cinq requêtes, l'espion des vues essaye d'abord de voler de l'espace aux index pour garder ces vues. Il n'existe qu'un seul index défini sur les vues V_1 and V_2 et les deux tables de dimension CLIENT et TEMPS. L'espion des vues s'attribue donc l'espace alloué à cet index et sélectionne une nouvelle vue V_3 ($\sigma_{Sexe='M'}$ CLIENT). Ces trois vues V_1, V_2, V_3 occupent maintenant 620 MB et il reste 180 MB. L'espion des index intervient donc en éliminant la vue V_3 et en récupérant l'espace disponible. Il provoque la création de l'index: ($V_2 \sim$ TEMPS $\sim$ CLIENT) qui réduit d'une manière significative le coût total des requêtes.

Solution	**Vues et index**	**CT**	S^V	S^I
Solution statique	V_1, V_2 & ($V_1 \sim V_2 \sim T \sim C$)	1955305	500	300
Après mise à jour	V_1, V_2 & ($V_1 \sim V_2 \sim T \sim C$)	2022680	540	320
Espion des vues	V_1, V_2 and V_3	4848146	800	0
Espion des index	V_1, V_2 & ($V_2 \sim$ T $\sim$ C)	2478150	550	250

Table 5: Les vues sélectionnées et leurs tailles pour les cinq requêtes après mises à jour

7 Conclusion et perspectives

Dans ce papier, nous avons développé une nouvelle méthodologie de la distribution automatique de l'espace entre les vues et les index dans les contextes statique et dynamique en prenant en considération l'interaction entre les vues et les index. Cette méthodologie est basée sur une approche itérative. Elle commence par une sélection initiale des vues et des index, puis essaye d'améliorer cette dernière en utilisant deux algorithmes gloutons (espion des index et espion des vues). Finalement, nous avons évalué notre approche en utilisant un schéma en étoile emprunté à Informix [6]. Les résultats ont montré que notre approche réduit d'une manière significative le coût total d'évaluation des requêtes. Dans certains cas, c'est l'espion des vues qui contribue le plus et dans d'autres, c'est l'espion des index. Cela démontre l'intérêt d'opposer les deux espions. Cependant,

l'administrateur de l'entrepôt peut privilégier les vues matérialisées en exécutant seulement l'espion des vues, ou privilégier les index en exécutant seulement l'espion des index. Ainsi, cette approche peut conduire à plusieurs stratégies de conception d'un entrepôt. Pour identifier les victimes (vues ou index à éliminer) notre algorithme utilise quatre politiques différentes basées sur la fréquence (la vue la moins fréquemment utilisée, l'index le moins fréquemment utilisé) et la taille (la vue de plus petite taille, l'index de plus grande taille). Il serait intéressant de considérer d'autres politiques en se basant sur des critères de coût (en éliminant les vues ou les index qui contribuent le moins dans la réduction du coût d'évaluation des requêtes).

References

[1] L. Bellatreche, M. Schneider, H. Lorinquer, and M. Mohania. Bringing together partitioning, materialized views and indexes to optimize performance of relational data warehouses. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2004)*, pages 15–25, September 2004.

[2] L. Bellatreche, M. Schneider, M. Mohania, and B. K. Bhargava. Partjoin: An efficient storage and query execution for data warehouses. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2002)*, pages 296–306, September 2002.

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

[4] S. Chaudhuri and V. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. *Proceedings of the International Conference on Very Large Databases*, pages 146–155, August 1997.

[5] S. Chaudhuri and V. Narasayya. Autoadmin 'what-if' index analysis utility. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 367–378, June 1998.

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

[7] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering Bulletin*, 18(2):3–18, June 1995.

[8] H. Gupta. Selection and maintenance of views in a data warehouse. Ph.d. thesis, Stanford University, September 1999.

[9] Y. Kotidis and N. Roussopoulos. An alternative storage organization for rolap aggregate views based on cubetrees. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 249–258, June 1998.

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

[11] A. Sanjay, V. R. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–370, June 2004.

[12] A. Sanjay, C. Surajit, and V. R. Narasayya. Automated selection of materialized views and indexes in microsoft sql server. *Proceedings of the International Conference on Very Large Databases*, pages 496–505, September 2000.

[13] T. Stöhr, H. Märtens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. *Proceedings of the International Conference on Very Large Databases*, pages 273–284, 2000.

[14] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. *Proceedings of the International Conference on Very Large Databases*, pages 136–145, August 1997.

[15] D. C. Zilio, J. Rao, S. Lightstone, G. M Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. *Proceedings of the International Conference on Very Large Databases*, pages 1087–1097, August 2004.