Matérialisation partielle et interrogation d'un hypercube de données dynamiques

Usman Ahmed, Anne Tchounikine, Maryvonne Miquel

Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France
firstname.lastname@insa-lyon.fr

Résumé. Les entrepôts de données ont généralement une stratégie de chargement des données par bloc et hors ligne ce qui les rendent peu compatibles avec des applications où les performances en temps sont critiques. Dans cet article, nous présentons un modèle multidimensionnel pour entreposer en temps réel les données d'un espace multidimensionnel hiérarchique. Nous proposons une matérialisation partielle de l'hypercube de données dans une structure d'arbre qui regroupe les données multidimensionnelles dans des partitions non ordonnées appelées Minimum Bounding Spaces (MBS). Nous présentons le principe des algorithmes d'insertion d'un nouveau fait et de requêtage. Nous évaluons la performance de notre solution en utilisant le Star Schema Benchmark. L'étude expérimentale montre que notre proposition est particulièrement performante à la fois en temps d'insertion et pour le traitement des requêtes.

1 Introduction

De plus en plus d'applications telles que la gestion des risques environnementaux, la finance ou le suivi de déplacement requièrent des analyses décisionnelles en temps réel. Or les modèles d'entrepôts de données classiques, avec leurs structures de données et leurs stratégies de mise à jour, ne sont plus efficaces dans des environnements dynamiques et temps réel. En effet :

- Du fait des politiques de mises à jour hors-ligne, les données les plus récentes sont intégrées dans l'entrepôt avec un délai qui peut nuire à la prise de décision. Elles ne sont pas prises en compte tant que l'opération de mise à jour (quotidienne, hebdomadaire, mensuelle, ...) n'est pas effectuée.
- Dans les entrepôts de données classiques, l'insertion des faits par blocs (bulk insertion) est privilégiée. Or, pour les applications décisionnelles où il est nécessaire de disposer des données les plus récentes, l'insertion de données dans l'entrepôt devrait être effectuée à la volée, tuple par tuple. Mais l'insertion d'un nouveau fait implique la mise à jour des agrégats, qui est une opération complexe et coûteuse en temps.
- Les applications visées nécessitent d'insérer non seulement de nouveaux faits, mais également de nouveaux membres de dimensions afin de tenir compte de l'évolution de l'environnement, comme par exemple l'installation ou le déplacement d'un capteur.

– Les solutions visant à optimiser les processus de maintenance des agrégats supposent souvent que les données sont décrites dans un espace ordonné. Si un ordre naturel n'existe pas, alors les membres de dimensions doivent être ordonnés soit en utilisant un attribut descriptif (par exemple l'ordre alphabétique d'un label), soit par l'utilisation de identifiant numérique. L'insertion de nouveaux membres dans les tables de dimensions nécessite alors une ré-organisation de l'ordre induit, inacceptable dans un contexte d'évolution dynamique.

Face à ce constat, il est nécessaire de définir de nouvelles stratégies de stockage et d'accès aux données adaptées aux environnements d'entreposage de données dynamiques. Notre contribution à cette problématique porte sur les propositions suivantes :

– Un modèle multidimensionnel capable d'intégrer à la fois des dimensions hiérarchiques ordonnées et non ordonnées. A cet effet, nous définissons un espace de données multidimensionnel hybride et hiérarchique. Les points de cet espace représentent les données détaillées et/ou agrégées de l'hypercube de données. Nous proposons un ensemble de relations et d'opérateurs dans cet espace.
– Un modèle de partitionnement dynamique de cet espace de données multidimensionnel hybride et hiérarchique. Pour cela, nous définissons un ensemble de métriques qui permet de distribuer et de regrouper dynamiquement les données de l'espace et d'obtenir des partitions de données non ordonnées appelées *Minimum Bounding Spaces* (MBS).
– Une structure en arbre de données, le DyTree, qui permet de stocker et d'indexer les partitions définies par les MBS et les mesures agrégées associées, constituant ainsi une matérialisation partielle de l'hypercube. Le partitionnement et la construction de l'arbre sont dynamiques et supportent l'insertion atomique d'un fait. Nous proposons les algorithmes qui permettent d'insérer et d'indexer de nouvelles données dans l'espace multidimensionnel, et d'accéder aux données.
– Une étude expérimentale pour évaluer la performance de notre solution. Nous comparons les performances du DyTree avec celles du DC-Tree (Ester et al. (2000)) en utilisant les données du Star Schema Benchmark (O'Neil et al. (2009)). Nous évaluons ces performances lors de la construction de l'arbre et lors du requêtage des données.

Le reste de l'article est organisé comme suit. La section 2 présente l'état de l'art. Dans la section 3, nous définissons notre modèle multidimensionnel, l'espace multidimensionnel et les métriques utilisées pour construire les nœuds de l'arbre DyTree. La section 4 énonce les principes des algorithmes de construction et de requêtage de l'arbre. Dans la section 5, la performance de notre solution est évaluée. Nous concluons notre travail par des perspectives dans la section 6.

2 Etat de l'art

Ces dernières années, une nouvelle attention a été portée sur l'entreposage de données temps réel et plus particulièrement sur l'amélioration des performances dans ce contexte. La prise en compte de ces problématiques dans les technologies OLAP et les entrepôts de données a donné lieu à de nouvelles propositions connues sous les termes de (*near*) *real time* (Golab et al. (2009)) *ou active* (Polyzotis et al. (2007)) *data warehouses*. Bateni et al. (2009) et Thiele et al. (2009) proposent des stratégies de planification pour prendre en compte les mises à jour, et Athanassoulis et al. (2011) s'orientent plutôt vers des stratégies de cache pour des mises

à jour différentielles. Dans Ester et al. (2000), les auteurs proposent une structure d'indexation appelée DC-Tree fondée sur l'ordonnancement partiel des dimensions. Cette proposition est une base de départ intéressante pour un modèle d'entrepôt de données dynamique. Cependant, à notre connaissance, ce travail préliminaire n'a pas été poursuivi. Dans Ahmed et al. (2010), nous proposons de construire un arbre d'indexation qui partitionne l'espace selon la dimension temporelle, regroupant ainsi dynamiquement les faits temporellement proches, même dans le cas où les données sont retardées.

Parallèlement, et en dehors du contexte temps réel ou dynamique, de nombreux travaux s'intéressent à l'optimisation des stratégies de matérialisation et de maintenance des agrégats. Sismanis et al. (2002) propose une structure de données compressées, appelée Dwarf, pour stocker et requêter efficacement un cube de données. Par la suite, dans Sismanis et al. (2003), les auteurs intègrent des dimensions hiérarchiques, améliorant ainsi l'accès aux agrégats. Lakshmanan et al. (2002) proposent une structure appelée Quotient Cube pour le résumé sémantique de cubes de données utilisant un schéma de partitionnement du cube. La réduction de la taille du cube de données est également abordée dans Wang et al. (2002). Li et al. (2004) s'intéressent plus particulièrement aux cubes de données impliquant de grands ensembles de dimensions. Leur proposition est un partitionnement vertical des ensembles de données multidimensionnelles en sous ensembles de données et la construction de cubes de données locaux. Toutes ces approches reposent sur la compression des données ou la diminution de la taille des cubes. Le problème de l'insertion atomique de données n'y est pas abordé.

Par ailleurs, certains auteurs étudient la prise en compte d'espace de données partiellement ou non ordonnées. Le cadre proposé par Sacharidis et al. (2009) prend en compte des dimensions partiellement ordonnées pour le requêtage de type skyline et Kaser et Lemire (2003) proposent un mécanisme pour un stockage OLAP optimisé par la réorganisation des valeurs d'attributs dans le cube de données. Dans un contexte non OLAP, Chen et al. (2009) proposent une méthode pour une indexation vectorielle dans un espace qui mixe des dimensions continues et des dimensions discrètes plates (c'est-à-dire sans hiérarchie). Des hyper-rectangles définis par des produits cartésiens permettent de regrouper les données continues et discrètes.

3 Concepts et notations

Dans cette section, nous présentons un modèle multidimensionnel qui intègre des dimensions ordonnées et des dimensions non ordonnées. Nous commençons par rappeler les concepts et les notations du modèle multidimensionnel (section 3.1). Ensuite, nous définissons un *espace de données multidimensionnel hiérarchique* et des concepts géométriques pour représenter et manipuler les différents objets de cet espace de données (section 3.2). Un ensemble de métriques sont introduites, et seront utilisées par la suite pour construire un arbre permettant de matérialiser partiellement l'hypercube de données.

A titre d'illustration, nous nous appuyons sur l'application décisionnelle classique de l'analyse des Ventes avec deux dimensions Time et Location (voir figure 3.1). A l'évidence, ce schéma très simplifié n'est pas une "vraie" application temps réel, qui devrait plutôt comporter des données estampillées à la seconde ou à la milliseconde. Cependant, ce schéma est suffisant pour illustrer nos concepts et nos définitions. Dans la partie 5, nous évaluerons notre solution avec des jeux de données plus réalistes.

3.1 Modèle de données multidimensionnel

Les premiers éléments de notre modèle reprennent les concepts traditionnels des modèles OLAP. Nous en rappelons ici les principales définitions et notations.

L'espace multidimensionnel est composé de *dimensions* définies par des ensembles de *membres*. Une dimension est dite *hiérarchique* si son domaine est composé de plusieurs sous-ensembles définissant des niveaux de détail ; les membres des différents niveaux ont entre eux des liens parent-enfant.

Nous notons l_i^j le $j^{ième}$ niveau de la dimension hiérarchique D_i et ALL_i le niveau le plus haut de D_i. L'ensemble des membres du niveau ALL_i est le singleton $\{all_i\}$. Une *dimension hiérarchique* est un graphe direct acyclique $\mathcal{H}$ dont les nœuds sont des éléments de L_i=$\{l_i^1, l_i^2, ..., l_i^{k-1}, ALL_i\}$ de la dimension D_i avec k niveaux de hiérarchie et ALL_i est la racine. Pour $l_i^u, l_i^v \in L_i$, on note $l_i^u \uparrow l_i^v$ s'il existe un chemin de l_i^v à l_i^u dans le graphe $\mathcal{H}$, où $\uparrow$ est une relation réflexive, transitive et asymétrique.

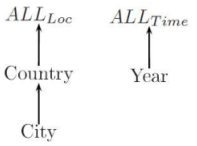


FIG. 3.1: Dimensions Location et Time

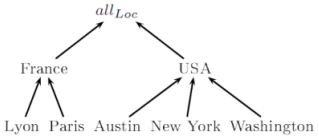


FIG. 3.2: Instances de la dimension Location

S_{D_i}, l'*espace de dimension* associé à la dimension D_i ayant k niveaux, est défini par l'union des ensembles de membres de tous les niveaux de la hiérarchie de dimensions, soit :

$$S_{D_i} = \bigcup_{j=1}^{k} domain(l_i^j)$$

Soit $level$ la fonction qui permet d'associer à un membre $m \in domain(l_i^j)$ son niveau d'appartenance, on a alors : $level(m) = l_i^j$. Par exemple, dans l'exemple de l'application Ventes, $level(Paris) = City$.

Une instance de la dimension hiérarchique D_i est un graphe direct acyclique $\mathcal{I}$ dont les nœuds sont des éléments de S_{D_i} et all_i est le sommet. Pour $m, n \in S_{D_i}$, on dit que "*m* est l'ancêtre de *n*" et on note $m \Uparrow n$ s'il existe un chemin de *n* à *m* dans le graphe $\mathcal{I}$. $\Uparrow$ est une relation réflexive, transitive et asymétrique.

Afin de naviguer dans les différents niveaux de hiérarchies des dimensions, nous définissons les opérateurs suivants.

Definition 1. (Drill-Up) : Soient $l_i^u, l_i^v \in L_i$ et $l_i^u \uparrow l_i^v$. Une opération de drill-up est une fonction notée $\sigma_{l_i^u}$ telle que :

$$\begin{array}{rcccl} \sigma_{l_i^u} & : & domain(l_i^v) & \longrightarrow & domain(l_i^u) \\ & & x & \longmapsto & \sigma_{l_i^u}(x) = y \quad tel\ que\ y \Uparrow x. \end{array}$$

Definition 2. (Drill-Down) : Soient $l_i^u, l_i^v \in L_i$ et $l_i^u \uparrow l_i^v$. Une opération de drill-down est une fonction notée $\varrho_{l_i^v}$ telle que :

$$\begin{aligned} \varrho_{l_i^v} : domain(l_i^u) &\rightarrow \mathcal{P}(domain(l_i^v)) \\ x &\mapsto \varrho_{l_i^v}(x) = \{y \mid \forall y,\ x \Uparrow y\} \end{aligned}$$

Exemples. $\sigma_{Country}(Paris) = France$ et $\varrho_{City}(USA)$ = *{New York, Austin, Washington}*

Dans le modèle multidimensionnel Ventes, toutes les dimensions sont hiérarchiques, avec un ordre partiel noté $\Uparrow$ entre les membres de différents niveaux de la hiérarchie de dimension. Cependant, les membres d'un même niveau hiérarchique n'ont pas nécessairement une relation d'ordre entre eux. De ce point de vue, nous distinguons deux types de dimension : les dimensions dites *non ordonnées*, pour lesquelles les membres d'un même niveau sont non ordonnés, par exemple pour la dimension Location, et les dimensions dites *ordonnées* pour lesquelles les membres d'un même niveau sont naturellement ordonnés, par exemple la dimension Time.

Dans notre proposition, nous prendrons en compte les dimensions ordonnées et les dimensions non ordonnées. Nous n'introduirons pas d'ordre artificiel pour les membres non ordonnés. Lors de la construction dynamique du cube, le traitement des dimensions ordonnées sera différencié par l'utilisation de métriques spécifiques exploitant l'ordre naturel.

3.2 Espace de données multidimensionnel hiérarchique

Nous présentons ici une nouvelle approche pour définir l'espace de données multidimensionnel. Notre proposition repose sur une représentation de l'espace des données et un modèle de regroupement et de partitionnement des points de cet espace.

Un espace de données multidimensionnel hiérarchique S est défini par le produit cartésien des espaces de dimensions associés à toutes les dimensions intervenant dans le modèle multidimensionnel. Pour un modèle multidimensionnel de n dimensions, $S = S_{D_1} \times S_{D_2} \times ... \times S_{D_n}$. Un point *p* de l'espace S est représenté par un n-tuple ordonné $(x_1, x_2, ..., x_n)$ où $\forall i\ (1 \leq i \leq n)$, $x_i \in S_{D_i}$ est la $i^{ième}$ coordonnée de *p*.

La Figure 3.3 montre une représentation possible de l'espace de données multidimensionnel hiérarchique associé au modèle multidimensionnel dont les instances de dimensions sont décrites par la figure 3.2. La représentation d'un espace de données multidimensionnel hiérarchique n'est pas unique, comme le montre la figure 3.4. On notera dans ces deux représentations que, si les membres d'une même dimension sont bien représentés sur le même axe, l'ordre sur cet axe est indifférent. Cet espace de données multidimensionnel hiérarchique est composé à la fois de dimensions hiérarchiques ordonnées et non ordonnées, c'est pourquoi nous nommons également cet espace HHMDS (*Hierarchical Hybrid Multidimensional Data Space*).

Lorsqu'un fait est inséré, un nouveau point de l'espace de données est créé. Pour chaque membre de dimension utilisé par ce nouveau point (c'est-à-dire pour chaque coordonnée) qui n'est pas encore présent dans l'espace, le membre est ajouté à l'axe de la dimension correspondante. Ainsi, les axes "grandissent" lorsque cela est nécessaire, sans prendre en compte un ordre entre ses membres.

A partir de l'ordre partiel induit par la hiérarchie de dimension, nous pouvons définir une relation de couverture entre différents points comme suit :

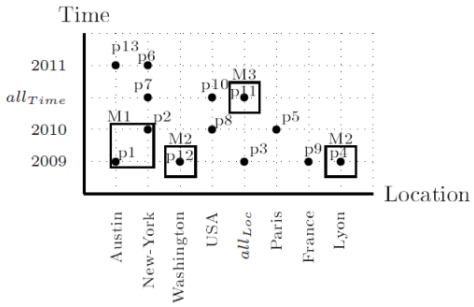


FIG. 3.3: Une représentation de l'espace de données multidimensionnel hiérarchique S

Definition 3. (Cover) : Soient deux points de l'espace n-dimensionnel $p(x_1, x_2, ..., x_n)$ et $p'(x'_1, x'_2, ..., x'_n)$, on dit que p' *covers* p, et on note $p' \circledcirc p$, si $\forall i\ (1 \leq i \leq n),\ x'_i \Uparrow x_i$.

Autrement dit, un point *p'* *covers* p si toutes les coordonnées de *p'* peuvent être obtenues par une opération de drill-up appliquée aux coordonnées de *p* (rappelons que l'opération drill-up est réflexive).

Exemples. Sur la figure 3.3, on a les relations suivantes :

$p_7 \circledcirc p_6$, $p_7 \circledcirc p_2$, $p_8 \circledcirc p_2$, mais $\neg(p_8 \circledcirc p_5)$, $p_{10} \circledcirc p_1,\ p_{10} \circledcirc p_2,\ p_{10} \circledcirc p_6$, $p_{10} \circledcirc p_7$, et enfin $\forall p \in S,\ p_{11} \circledcirc p$.

Definition 4. (hyper-plan multidimensionnel) : Un hyper-plan multidimensionnel $\mathcal{P}$ (nommé plus simplement un *hyper-plan* par la suite) avec n dimensions est défini par $\mathcal{P} = domain(l_1^{h_1}) \times domain(l_2^{h_2}) \times ... \times domain(l_n^{h_n})$ où $\forall i\ (1 \leq i \leq n)$, $1 \leq h_i \leq k_i$ et k_i est le nombre de niveaux de la dimension D_i.

Il y a $k_1 * k_2 * ... * k_n$ hyper-plans différents dans un espace de données multidimensionnel hiérarchique de n dimensions. Les hyper-plans sont des sous-espaces disjoints d'un HHMDS. Chaque hyper-plan est identifié par un n-tuple ordonné $< l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$. Un point $p(x_1, x_2, ..., x_n)$ appartient à l'hyper-plan $< level(x_1), level(x_2), ..., level(x_n) >$. Tous les hyper-plans de S sont organisés dans une hiérarchie avec un ordre partiel $\geq_p$.

Soient $\mathcal{P} < l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$ et $\mathcal{P}' < l_1^{h'_1}, l_2^{h'_2}, ..., l_n^{h'_n} >$ deux hyper-plans multidimensionnels, on note $\mathcal{P} \geq_p \mathcal{P}'$ ssi $\forall i\ (1 \leq i \leq n),\ l_i^{h_i} \uparrow l_i^{h'_i}$. Sur la figure 3.4, on a $< City, ALL_{Time} > \geq_p < City, Year >$

Definition 5. (Minimum Bounding Space) : Soit $\Delta = \{(x_{1,1},\ x_{1,2}, ...,\ x_{1,n}), (x_{2,1},\ x_{2,2}, ..., x_{2,n}), ...,\ (x_{m,1},\ x_{m,2}, ...,\ x_{m,n})\}$ un ensemble de m points multidimensionnels du même hyper-plan, c'est-à-dire tels que $x_{j,i} \in domain(l_i^{k_i})$ pour $1 \leq i \leq n$, $1 \leq j \leq m$, avec k_i le niveau de la hiérarchie de dimension D_i. Un espace minimal englobant (*Minimum Bounding Space*, noté MBS) construit à partir de Δ, et noté $\mathcal{M}_\Delta$, est défini par :

$$\mathcal{M}_\Delta = \bigcup_{j=1}^{m} x_{j,1} \times \bigcup_{j=1}^{m} x_{j,2} \times ... \times \bigcup_{j=1}^{m} x_{j,n}$$

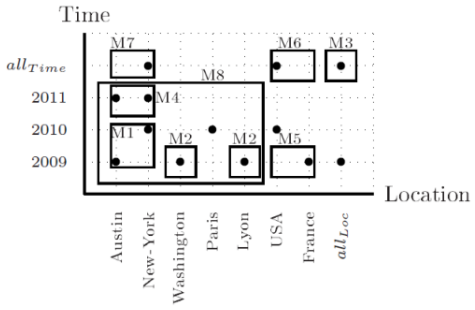


FIG. 3.4: Espace de données multidimensionnel hiérarchique avec des MBSs

Un MBS permet le regroupement des données dans l'espace. L'ensemble $E_i = \bigcup_{j=1}^{m} x_{j,i}$ représente la $i^{ième}$arête du MBS. Le MBS $\mathcal{M} = E_1 \times E_2 \times ... \times E_n$, appartient à l'hyper-plan $< level(e_1), level(e_2), ..., level(e_n) >$ avec $e_i \in E_i$.

Exemples. Dans la figure 3.3, on peut représenter les MBS suivants :

$M1$= $M_{\{(Austin,2009),(New\,York,2010)\}}$ = $\{Austin, New\ York\}\times\{2009, 2010\}$ est un MBS qui regroupe deux points,

$M2$= $M_{\{(Washington,2009),(Lyon,2009)\}}$ = $\{Washington, Lyon\}\times\{2009\}$ est un MBS qui regroupe deux points qui ne sont pas contigus dans la représentation de l'espace de données,

$M3$= $M_{\{(all_{Loc},all_{Time})\}} = \{all_{Loc}\} \times \{all_{Time}\}$ un MBS qui regroupe un point qui couvre tous les points de l'espace HHMDS (l'apex).

Definition 6. (Translate-Up) : Soient $\mathcal{P} < l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$ et $\mathcal{P}' < l_1^{h_1'}, l_2^{h_2'}, ..., l_n^{h_n'} >$ deux hyper-plans de S tels que $\mathcal{P}' \geq_p \mathcal{P}$ et $\mathcal{M} = E_1 \times E_2 \times ... \times E_n$ un MBS de $\mathcal{P}$. Une opération translate-up $\lceil_{\mathcal{P}'}$ sur un MBS est définie par :

$$\begin{array}{rcl} \lceil_{\mathcal{P}'} : \mathcal{P} & \rightarrow & \mathcal{P}' \\ \mathcal{M} & \mapsto & \mathcal{M}' = \bigcup\limits_{e_1 \in E_1} \sigma_{l_1^{h_1'}}(e_1) \times \bigcup\limits_{e_2 \in E_2} \sigma_{l_2^{h_2'}}(e_2) \times ... \times \bigcup\limits_{e_n \in E_n} \sigma_{l_n^{h_n'}}(e_n) \end{array}$$

Une opération translate-up transpose (ou « projette ») un MBS depuis son hyper-plan vers un hyper-plan de niveau plus élevé dans S.

Definition 7. (Translate-Down) **:** Soient $\mathcal{P} < l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$ et $\mathcal{P}' < l_1^{h_1'}, l_2^{h_2'}, ..., l_n^{h_n'} >$ deux hyper-plans de S tels que $\mathcal{P}' \geq_p \mathcal{P}$ et soit $\mathcal{M}' = E_1' \times E_2' \times ... \times E_n'$ un MBS de $\mathcal{P}'$. Une opération translate-down $\lfloor_{\mathcal{P}}$ sur un MBS est définie par :

$$\begin{array}{rcl} \lfloor_{\mathcal{P}} : \mathcal{P}' & \rightarrow & \mathcal{P} \\ \mathcal{M}' & \mapsto & \mathcal{M} = \bigcup\limits_{e_1 \in E_1'} \varrho_{l_1^{h_1}}(e_1) \times \bigcup\limits_{e_2 \in E_2'} \varrho_{l_2^{h_2}}(e_2) \times ... \times \bigcup\limits_{e_n \in E_n'} \varrho_{l_n^{h_n}}(e_n) \end{array}$$

Une opération translate-down transpose un MBS depuis son hyper-plan vers un hyper-plan de niveau plus bas dans S.

Exemples. A partir de la figure 3.4, on a :
$\lceil_{<City,\ ALL_{Time}>}(M1) = \{Austin,\ New\,York\} \times \{all_{Time}\}$ = $M7$,
$\lceil_{<Country,\ Year>}(M2)$ =$\{France, USA\}\times\{2009\}$ = $M5$,
$\lfloor_{<City,\ Year>}(M3)$=$\{Austin,\ New\ York\}\times\{2009,\ 2010,\ 2011\} = M4 \cup M1$,
$\lfloor_{<City,\ Year>}(M6)$=$\{Austin,\ New\ York,\ Washington,\ Paris,\ Lyon\}\times\{2009,\ 2010,\ 2011\}$ = $M8$

On note que les opérateurs translate-up et translate-down peuvent créer un MBS qui contient des points qui auparavant n'existaient pas dans le HHMDS (par exemple dans la figure 3.4 M5, M6 et M7 et M8). On note également que $\lfloor_{\mathcal{P}}(\lceil_{\mathcal{P}'}(\mathcal{M})) \neq \mathcal{M}$ alors que $\lceil_{\mathcal{P}'}(\lfloor_{\mathcal{P}}(\mathcal{M}^{'})) = \mathcal{M}^{'}$.

3.3 Métriques

Les métriques définies dans cette section vont permettre de décrire la position relative de deux MBS et de calculer la différence entre deux MBS. Ces métriques seront utilisées pour construire un arbre de matérialisation partielle de l'hypercube.

Nous avons vu que les arêtes d'un MBS sont représentées par un ensemble de membres. Cependant, pour une dimension ordonnée, telle que la dimension Time, l'arête peut également être décrite par un intervalle. Soit D_t une dimension ordonnée de notre schéma multidimensionnel, et E_t l'arête correspondante d'un MBS $\mathcal{M}$. L'intervalle de temps couvert par E_t est $Interval(E_t) = [\min_{m_i \epsilon E_t}(m_i), \max_{m_i \epsilon E_t}(m_i)]$. Cette définition spécifique aux dimensions naturellement ordonnées est utilisée pour différentier le mode de calcul des métriques : pour les dimensions non ordonnées, le calcul des métriques fera intervenir la cardinalité de l'ensemble des membres de l'arête ; pour les dimensions ordonnées, le calcul utilisera la longueur de l'intervalle représentant l'arête. Nous supposerons par la suite que seule la dimension Time est ordonnée, mais notre proposition pourrait être généralisée à plusieurs dimensions ordonnées.

Definition 8. (Contains) : Soient $\mathcal{M} = E_1 \times E_2 \times ... \times E_n$ et $\mathcal{N} = E_1^{'} \times E_2^{'} \times ... \times E_n^{'}$ deux MBS de S.

$$(\mathcal{M}\ contains\ \mathcal{N})\ si \begin{cases} \forall p \in \mathcal{N},\ \exists q \in \mathcal{M} \mid q \circledcirc p \\ ou \\ (\forall i \neq t,\ E_i^{'} \subseteq E_i)\ et\ pour\ (i = t,\ Interval(E_t^{'}) \subseteq Interval(E_t)) \end{cases}$$

Contains est une relation réflexive, transitive et asymétrique entre MBS.

Definition 9. (Overlap) : Soient $\mathcal{M} = E_1 \times E_2 \times ... \times E_n$ et $\mathcal{N} = E_1^{'} \times E_2^{'} \times ... \times E_n^{'}$ deux MBSs d'un même hyper-plan. On a

$$(\mathcal{M}\ overlaps\ \mathcal{N})\ si :\ \exists p \in \mathcal{N},\ \exists q \in \mathcal{M} \mid q = p$$

On peut quantifier la relation *overlap* entre $\mathcal{M}$ et $\mathcal{N}$ par :

$$Overlap(\mathcal{M},\mathcal{N}) = \begin{cases} 0 \quad \text{si } Interval(E_t) \cap Interval(E_t^{'}) = \emptyset \\ \prod_{i=1, i\neq t}^{n} |E_i \cap E_i^{'}| * |Interval(E_t) \cap Interval(E_t^{'})| \quad sinon \end{cases}$$

Definition 10. (Extension) : Soient $\mathcal{M} = E_1 \times E_2 \times ... \times E_n$ et $\mathcal{N} = E_1^{'} \times E_2^{'} \times ... \times E_n^{'}$ deux MBS d'un même hyper-plan. L'extension de $\mathcal{M}$ par rapport à $\mathcal{N}$, notée $Extension(\mathcal{M}|\mathcal{N})$ est calculée par :

$$Extension(\mathcal{M}|\mathcal{N}) = \sum_{i=1, i \neq t}^{n} |E_i^{'} - E_i| + g(d_l) + g(d_u)$$

avec

$$\begin{cases} d_l = \min_{m_i \in E_t}(m_i) - \min_{n_i \in E_t^{'}}(n_i) \\ d_u = \max_{n_i \in E_t^{'}}(n_i) - \max_{m_i \in E_t}(m_i) \end{cases} \quad et \;\; g(x) = \begin{cases} 0 & si\, x < 0 \\ x & sinon \end{cases}$$

Exemples. Figure 3.4, on note que :
($M7\ contains\ M1$), ($M5\ contains\ M2$), ($M8\ contains\ M1$) etc.
$Overlap(M1,\ M4) = 0$, $Overlap(M8,\ M4) = 2$, $Overlap(M8,\ M4 \cup M2) = 12$ etc.
$Extension(M1|M2) = 2$, $Extension(M2|M1) = 3$, $Extension(M1|M4) = 1$ etc.

4 Matérialisation de l'hypercube dans le DyTree

Les métriques définies précédemment sont utilisées dans l'algorithme d'insertion pour construire des MBS optimaux. Le mode de calcul des métriques pour les dimensions ordonnées favorise le regroupement de données proches au sens de l'ordre de la dimension (par exemple, le regroupement de données temporellement proches pour la dimension Time). Ainsi, ce rapprochement rendra les requêtes de type Range Query plus efficaces en temps de réponse. Dans cette section, nous définissons la structure d'arbre, nommé DyTree, qui va permettre de stocker les MBS et nous présentons le principe des algorithmes d'insertion et de requêtage.

4.1 Eléments du DyTree

A chaque point $p(x_1, x_2, ..., x_n)$ du HHMDS, on associe au moins une valeur numérique notée $m(p)$ et appelée *mesure*. Le point p associé à $m(p)$ est un *fait* si *p* appartient à l'hyper-plan $< l_1^1, l_2^1, ..., l_n^1 >$. La valeur numérique associée à un MBS $\mathcal{M}$ de k points, notée $a(\mathcal{M})$, est un *agrégat.* La valeur de $a(\mathcal{M})$ est calculée par une fonction d'agrégation f (SUM, MAX, MIN etc.) sur les valeurs des mesures associées aux points $p_i \in \mathcal{M}$, $1 \leq i \leq k$: $\mathtt{a}(\mathcal{M}) = f(m(p_1),\ m(p_2), ...,\ m(p_k))$.

Nous définissons une structure d'arbre, le DyTree, qui stocke les points multidimensionnels, les MBS, et leurs mesures ou valeurs d'agrégats associés. Le DyTree est un arbre balancé qui indexe et stocke les MBS dans ses nœuds internes, et les faits dans ses feuilles. Les nœuds du DyTree sont créés et mis à jour en intégrant les nouvelles données et en utilisant les métriques dédiées. L'ordre d'insertion des faits dans le DyTree est indifférent, ce qui facilite leur insertion tuple par tuple. Ainsi, le DyTree matérialise dynamiquement et partiellement l'hypercube de données et offre un compromis acceptable entre l'espace mémoire utilisé et les temps de réponse aux requêtes. Un DyTree présente trois types de nœuds :

- Un nœud de type Directory Node est un nœud de capacité fixe et limitée. Il représente un MBS, la ou les valeurs agrégées associées au MBS (dans le cas de plusieurs mesures) et les pointeurs vers les nœuds qu'il indexe (appelés "entrées").
- Un nœud de type Super Node est un nœud dont la capacité est variable et illimitée. Il représente un MBS, la ou les valeurs agrégées au MBS et les entrées dont le nombre peut être variable et illimité.

– Un nœud de type Data Node est une feuille et représente un fait. C'est également un MBS qui n'englobe qu'un seul point. Ses arêtes sont des singletons représentant les coordonnées du point dans l'espace multidimensionnel.

La racine du DyTree est toujours un Directory Node avec un MBS au niveau ALL_i pour chaque dimension D_i de l'espace, c'est-à-dire qu'elle représente toujours le MBS $all_1 \times all_2 \times ... \times all_n$, qui est l'apex de l'hypercube.

Nous présentons maintenant les principes de nos algorithmes permettant d'insérer un nouveau fait dans le DyTree et de traiter les requêtes de type Range Query sur le DyTree. Un algorithme de traitement des requêtes de type Point Query a également été développé et implémenté dans notre prototype, il n'est pas présenté ici par manque de place. Un requête de type Group-by peut être considérer comme un ensemble des requêtes agrégées de type Point Query, alors on ne l'implémente pas dans notre prototype.

4.2 Principe de l'algorithme d'insertion

Pour insérer un nouveau fait, c'est-à-dire un nouveau point de l'espace HHMDS, il faut : (1) Déterminer le Directory Node de plus bas niveau où le placer, c'est-à-dire le Directory Node dont les entrées sont des Data Nodes. (2) Si le nœud de plus bas niveau sélectionné est un Directory Node qui n'a pas atteint sa capacité maximum ou si c'est un super nœud, alors un Data Node fils est créé et la valeur de l'agrégat est mise à jour. Dans le cas d'un nœud plein, l'algorithme de Split est appelé puis le fait est inséré.

Pour déterminer le Directory Node où placer le nouveau fait, on parcourt l'arbre depuis la racine en sélectionnant les sous arbres correspondant à des MBS pouvant contenir le point à insérer (c'est-à-dire le fait), au sens de la métrique *contains*. A chaque choix d'un sous arbre, l'agrégat correspondant au Directory Node est mis à jour. S'il n'y a pas de MBS contenant le point, il est nécessaire d'effectuer une extension minimale, c'est-à-dire d'étendre le MBS dont l'extension avec le point est la plus petite (au sens de la métrique *extension*). Après cette extension, l'algorithme d'insertion est appelé récursivement pour insérer le nouveau fait.

4.3 Principe de l'algorithme de split

Soit le MBS $\mathcal{M} = E_1 \times E_2 \times ... \times E_k \times ... \times E_n$ dans l'hyper-plan $\mathcal{P} < l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$, une opération de split selon la $k^{ième}$ dimension est une opération qui produit deux nouveaux MBS $\mathcal{M}' = E_1' \times E_2' \times ... E_k' \times ... \times E_n'$ et $\mathcal{M}'' = E_1'' \times E_2'' \times ... E_k'' \times ... \times E_n''$ dans l'hyper-plan $\mathcal{P}' < l_1^{h_1'}, l_2^{h_2'}, ..., l_n^{h_n'} >$ tels que :

$$\begin{cases} \exists k \; tel \; que \; (l_k^{h_k} \uparrow l_k^{h_k'}) \; et \; (\forall i \neq k, \; h_i = h_i') \\ ou \\ \exists k \; tel \; que \; (E_k = E_k' \bigcup E_k'') \; et \; (\forall i, \; h_i = h_i') \end{cases}$$

On notera qu'ainsi défini, un split peut produire deux MBS qui sont dans le même hyper-plan que le MBS d'origine (quand $E_k = E_k' \cup E_k''$) ou dans un hyper-plan inférieur (quand $l_k^{h_k} \uparrow l_k^{h_k'}$). De plus, un split de MBS peut produire deux MBS avec ou sans overlap.

L'algorithme de Split est appelé par l'algorithme d'insertion dans le cas où un Directory Node a atteint sa capacité. Le principe de l'algorithme est de : (1) Sélectionner la dimension et son niveau où sera effectué le split du MBS correspondant au nœud à diviser. Cela revient

à rechercher l'hyper-plan dans lequel sera représenté le résultat du split. On commence par privilégier la dimension de plus forte cardinalité, c'est à dire celle dont l'arête a le plus de membres. Dans ce cas, le niveau de split (et donc l'hyper-plan de split) est le même que celui du MBS à diviser. Sinon, la dimension de plus haut niveau est choisie comme dimension de split et le niveau de split est le niveau inférieur de la hiérarchie de dimension (au sens de l'opération *translate-down*). Lorsque la dimension de split et le niveau sont sélectionnés, l'hyper-plan de split est connu : les niveaux des dimensions, sauf celle de la dimension où s'effectue le split, sont les mêmes que celles du MBS d'origine. (2) Une fois déterminé l'hyper-plan de split, les MBS de toutes les entrées du nœud à diviser et celui qui a provoqué le split sont projetés (au sens de l'opération *translate-up*) sur l'hyper-plan de split, ce sont les nœuds à distribuer. Deux nouveaux Directory Nodes sont créés avec des MBS dans l'hyper-plan de split, ce sont les nœuds candidats résultant au split. Les deux nœuds à distribuer les plus distants, c'est-à-dire ceux qui nécessitent une extension maximale (au sens de la métrique *extension*) sont placés dans chaque nœud divisé. Ensuite, chaque nœud à distribuer est attribué à un nœud divisé en utilisant, dans l'ordre, les critères suivants : extension minimale, overlap minimal, nombre d'entrées disponibles. Après avoir affecté tous les nœuds à distribuer entre les deux nœuds résultant du split, si l'overlap entre ces deux nœuds divisés est plus grand qu'une limite prédéfinie, alors une nouvelle dimension de split et un nouveau niveau doivent être choisis.

Ce processus se poursuit jusqu'à trouver un split acceptable. En cas de succès, les nœuds divisés remplacent le nœud d'origine dans les liens de ses parents, ce qui peut déclencher un processus récursif de split. Si le nœud à diviser est la racine, alors une nouvelle racine est créée et l'arbre croit en profondeur. Si aucune solution de split n'est trouvée, alors le nœud de départ est transformé en Super Node.

4.4 Principe de l'algorithme pour les Range Query

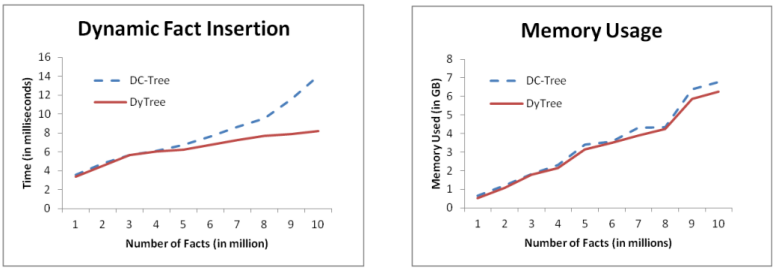
La requête est d'abord réécrite sous la forme d'un MBS et l'exploration de l'arbre commence à la racine. Pour un nœud donné du DyTree, le principe de l'algorithme est de comparer son MBS avec le MBS représentant la requête. Si le MBS de la requête contient le MBS du nœud (au sens de la métrique *contains*), la valeur d'agrégat du nœud est agrégée au résultat et le sous arbre correspondant n'est plus exploré. Dans le cas contraire, les deux MBS sont rapportés au même hyper-plan (avec l'opération *translate-up* sur le MBS de plus bas niveau) afin de pouvoir calculer l'overlap. S'il y a un chevauchement entre les deux MBS, l'algorithme est appliqué récursivement pour chaque entrée du nœud courant, l'objectif étant de trouver la ou les parties qui participeront au résultat de la requête. S'il n'y a pas chevauchement, alors ce nœud et son sous arbre est ignoré. Le résultat de la requête est ainsi construit récursivement par une exploration maitrisée par les métriques *contains* et *overlap*.

5 Evaluation des performances

Le prototype a été réalisé en Visual C#. Nous comparons les performances de notre solution avec celles du DC-Tree (Ester et al. (2000)). Pour les 2 solutions, les nœuds de type Data Node sont stockés sur disque alors que les nœuds de type Directory Node ou Super Node sont en mémoire centrale. Les tests sont tous effectués sur un ordinateur avec un processeur 2.67 GHz Intel Core 2 Duo, une RAM de 14 Go et un disque 500 Go.

Comme jeux de données, nous utilisons le schéma et les données du Benchmark SSB (Star Schema Benchmark) (O'Neil et al. (2009)), qui est une version OLAP du benchmark TPC-H. Dans le jeux de données SSB, nous considérons que la dimension Time est ordonnée alors que les autres dimensions (Customer, Supplier, Product) sont non ordonnées. Les tables de dimensions et de faits sont en schéma en étoile et la mesure est une valeur numérique dont la fonction d'agrégation est SUM. Les jeux de données générés comprennent entre 1,000,000 et 10,000,000 tuples. Les jeux de requêtes tests sont composés de 50 requêtes de type Range Query ou Point Query. Pour chaque Range Query, le niveau hiérarchique de chaque dimension et l'intervalle de valeurs requêté sont fixés aléatoirement. Pour un Point Query, un point multidimensionnel est aléatoirement sélectionné dans l'espace de données. Ce point n'est pas forcement situé au niveau le plus bas du HHMDS, mais peut appartenir à un hyper plan de plus haut niveau et sa mesure correspond alors à un agrégat.

Nous avons retenu les indicateurs de performance suivants : la taille mémoire utilisée pour stocker les nœuds internes de l'arbre, le temps d'insertion et le temps de réponse à une requête. Les temps d'insertion sont calculés pour l'insertion d'un tuple et non d'un bloc.



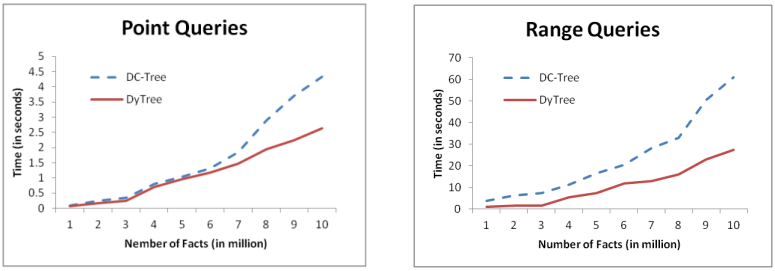
(a) (b)

FIG. 5.1: Insertion d'un tuple et mémoire utilisée avec SSB

Les tests comparatifs sur le temps d'insertion et la mémoire utilisée sont présentés figure 5.1. La figure 5.1a permet de comparer le temps d'insertion dynamique d'un fait dans le DyTree et dans le DC-Tree. Ce temps correspond à la moyenne des temps de 100 insertions atomiques dans un DyTree et DC-Tree stockant 1, 2 .., jusqu'à 10 millions de faits. Les résultats montrent que pour de petits jeux de données, le DC-Tree et le DyTree ont des performances similaires. Mais, avec la croissance du volume du jeu de données, le temps d'insertion dans le DC-Tree devient considérablement plus important que pour le DyTree. Une analyse détaillée des résultats nous a permis de constater que ce comportement était lié à l'augmentation du nombre de nœuds de type Super Node. Ces Super Nodes étant des nœuds à capacité variable plus grande, il peuvent indexer plus de données qu'un Directory Node, pénalisant ainsi la recherche préalable à l'insertion.

Les résultats sur la mémoire utilisée montrent peu de différences entre les deux solutions (figure 5.1b) même si notre solution est légèrement moins consommatrice. La différence peut s'expliquer par la présence accrue de Super Nodes dans le DC-Tree.

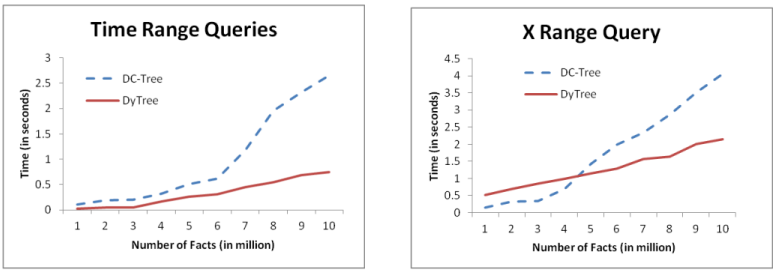
La figure 5.2 permet de comparer les temps de réponses pour des requêtes de type Range Query et Point Query. Puisque notre solution favorise le regroupement de valeurs temporellement proches dans le même MBS, il est normal que le temps de réponse des Range Queries soit nettement meilleurs (figure 5.2b). Le DyTree présente également de meilleures performances



(a) (b)

FIG. 5.2: Temps de réponse à des requêtes avec SSB

dans le cas de requêtes de type Point Query. Ceci est dû principalement au nombre plus faible de Super Nodes dans le DyTree et au regroupement de données temporellement proches. En effet, comme le Point Query peut porter sur le résultat d'agrégat, le regroupement de données temporellement proches améliore le requêtage pour ce type de requêtes.



(a) (b)

FIG. 5.3: Temps de réponse sur le SSB pour des Range Query sur (a) la dimension ordonnée Time (b) une dimension non ordonnée quelconque

Afin de mieux comprendre la différence entre les temps de réponse des deux solutions, la figure 5.3 présente les résultats obtenus pour des Range Query portant uniquement sur une dimension. La figure 5.3a donne les temps de réponse lorsque la requête porte uniquement sur un intervalle de temps, dont la dimension est ordonnée. Dans la figure 5.3b, les temps de réponse sont donnés pour des requêtes où l'intervalle d'intérêt porte sur une dimension quelconque non ordonnée. On constate que la différence entre les temps de réponse est plus importante dans la figure 5.3a que dans 5.3b. Cette différence est bien sûr due à notre stratégie d'appliquer un traitement spécial aux dimensions ordonnées et d'utiliser l'ordre pour regrouper les données proches. Cependant, on constate que, même sans stratégie particulière pour les dimensions ordonnées, les performances du DyTree sont meilleures dans 5.3b. L'ensemble des résultats obtenus permet de conclure que le DyTree est meilleur que le DC-Tree à la fois en insertion dynamique et en temps de réponse pour les requêtes.

6 Conclusions et perspectives

Notre travail porte sur la prise en compte de données multidimensionnelles hiérarchiques dans un espace de données hybride. Le contexte d'étude est l'OLAP dynamique et l'entreposage de données lorsque les systèmes opérationnels et l'entrepôt sont tous les deux en ligne et que l'analyse doit porter sur des données agrégées à différents niveaux hiérarchiques des dimensions. Nous définissons un modèle d'espace de données multidimensionnel hiérarchique et un ensemble de métriques permettant de regrouper et de partitionner les données de cet espace. Nous proposons une technique de matérialisation partielle de l'hypercube à l'aide d'un arbre, le DyTree, qui supporte des dimensions ordonnées ou non ordonnées. Nous avons implémenté le DyTree ainsi que le DC-Tree, et nous avons mené une étude comparative en utilisant le benchmark SSB. Cet étude montre que le DyTree a de meilleures performances que le DC-Tree et qu'il est une solution particulièrement intéressante quand le volume des données augmente.

Pour encore améliorer ces performances, nous étudions de nouvelles métriques pour construire le DyTree afin d'orienter plus efficacement l'insertion de nouveaux faits dans les nœuds de l'arbre. L'optimisation de la construction des MBS devrait également permettre d'améliorer les temps de réponse aux requêtes. Nous nous intéressons également à la recherche de valeurs optimales des paramètres tels que la capacité des nœuds de type Directory Node ou la limite d'overlap toléré, en construisant de nouveaux indicateurs d'évaluation des arbres construits et de caractérisation des jeux de données.

Références

Ahmed, U., A. Tchounikine, M. Miquel, et S. Servigne (2010). Real-time temporal data warehouse cubing. In *Proceedings of the 21st international conference on Database and expert systems applications : Part II*, DEXA'10, pp. 159–167.

Athanassoulis, M., S. Chen, A. Ailamaki, P. B. Gibbons, et R. Stoica (2011). MaSM : efficient online updates in data warehouses. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pp. 865–876.

Bateni, M. H., L. Golab, M. T. Hajiaghayi, et H. Karloff (2009). Scheduling to minimize staleness and stretch in real-time data warehouses. In *Proceedings of the 21st annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '09, pp. 29–38.

Chen, C., S. Pramanik, Q. Zhu, W. Alok, et G. Qian (2009). The C-ND tree : a multidimensional index for hybrid continuous and non-ordered discrete data spaces. In *Proceedings of the 12th International Conference on Extending Database Technology*, EDBT '09, pp. 462–471.

Ester, M., J. Kohlhammer, et H.-P. Kriegel (2000). The DC-tree : A Fully Dynamic Index Structure for Data Warehouses. In *Proceedings of the 16th International Conference on Data Engineering*, ICDE'00, pp. 379–388.

Golab, L., T. Johnson, J. S. Seidel, et V. Shkapenyuk (2009). Stream warehousing with DataDepot. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pp. 847–854.

Kaser, O. et D. Lemire (2003). Attribute value reordering for efficient hybrid OLAP. In *Proceedings of the 6th ACM international workshop on Data warehousing and OLAP*, DOLAP '03, pp. 1–8.

Lakshmanan, L. V. S., J. Pei, et J. Han (2002). Quotient Cube : how to summarize the semantics of a data cube. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB'02.

Li, X., J. Han, et H. Gonzalez (2004). High-dimensional OLAP : a minimal cubing approach. In *Proceedings of the 30th international conference on Very Large Data Bases - Volume 30*, VLDB '04, pp. 528–539.

O'Neil, P., E. O'Neil, X. Chen, et S. Revilak (2009). In R. Nambiar et M. Poess (Eds.), *Performance Evaluation and Benchmarking*, Chapter The Star Schema Benchmark and Augmented Fact Table Indexing, pp. 237–252.

Polyzotis, N., S. Skiadopoulos, P. V. andAlkis Simitsis, et N.-E. Frantzell (2007). Supporting Streaming Updates in an Active Data Warehouse. In *Proceesdings of 23rd International Conference on Data Engineering*, ICDE'07, pp. 476 – 485.

Sacharidis, D., S. Papadopoulos, et D. Papadias (2009). Topologically sorted skylines for partially ordered domains. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE'09, pp. 1072–1083.

Sismanis, Y., A. Deligiannakis, Y. Kotidis, et N. Roussopoulos (2003). Hierarchical dwarfs for the rollup cube. In *Proceedings of the 6th ACM international workshop on Data warehousing and OLAP*, DOLAP '03, pp. 17–24.

Sismanis, Y., A. Deligiannakis, et Y. Roussopoulos, Nickand Kotidis (2002). Dwarf : shrinking the PetaCube. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of data*, SIGMOD '02, New York, NY, USA, pp. 464–475. ACM.

Thiele, M., U. Fischer, et W. Lehner (2009). Partition-based workload scheduling in living data warehouse environments. *Information Systems 34*, 382–399.

Wang, W., H. Lu, J. Feng, et J. X. Yu (2002). Condensed Cube : An Efficient Approach to Reducing Data Cube Size. In *Proceedings of International Conference on Data Engineering*, ICDE '02, Los Alamitos, CA, USA, pp. 155–165. IEEE Computer Society.

Summary

The scheduled offline batch update strategy used in traditional data warehouses is no more suitable for time-critical applications. To address the issue, we present a multidimensional model for real time data warehousing in hierarchical multidimensional data space. We propose a dynamic partial cube materialization and a tree storage structure that groups the multidimensional data in non-ordered data partitions called minimum bounding spaces. Algorithms for integrating a new fact and for querying the so-constructed cube are provided. We use Star Schema Benchmark to evaluate the performance of our solution. Experimental study shows performance improvement in both insertion time and queries over the data space.