Une Démarche Conjointe de Fragmentation et de Placement dans le Cadre des Entrepôts de Données Parallèles

Soumia Benkrid *, Ladjel Bellatreche **

* Ecole Nationale Supérieure en Informatique, Oued-Smar Alger - Algérie
s_benkrid@esi.dz
** LISI/ENSMA - Université de Poiters, Futuroscope 86960 France
bellatreche@ensma.fr

Résumé. Traditionnellement, concevoir un entrepôt de données parallèle consiste d'abord à partitionner son schéma ensuite allouer les fragments générés sur les noeuds d'une machine parallèle. L'inconvénient majeur d'une telle approche est son ignorance de l'interdépendance entre les processus de fragmentation et d'allocation. Une des entrées du problème d'allocation est l'ensemble de fragments générés par la fragmentation. Notons que les deux processus cherchent à optimiser le même ensemble de requêtes. Dans ce papier, nous proposons une approche de conception d'un entrepôt de données relationnel parallèle selon une architecture distribuée (shared nothing) intégrant les processus de fragmentation et d'allocation. Ensuite, une méthode de répartition de charges sur les noeuds de la machine parallèle est proposée. Finalement, une validation de nos propositions en utilisant le banc d'essai APB-1 release II est présentée.

1 Introduction

Le processus d'aide à la décision est souvent mené par l'intermédiaire de requêtes complexes caractérisées par des jointures, sélections et agrégations exécutées sur des entrepôts de données très volumineux. Au cours de la dernière décennie, la taille des entrepôts de données a augmenté de 5 à 100 téra octets (DeWitt et al.). Pour optimiser les requêtes décisionnelles sur des entrepôts de données de telle taille, les techniques d'optimisation classiques telles que les *vues matérialisées*, les *index avancés*, la *fragmentation* ne sont pas suffisantes. Le traitement parallèle devient alors une solution incontournable pour réduire les coûts de requêtes complexes. Ce dernier n'a pas eu la même attention de la part de la communauté des entrepôts de données contrairement aux techniques classiques ; à l'exception des travaux de (Stöhr et al., 2000; Stöhr et Rahm, 2001) et de (Furtado, 2004). Paradoxalement, les éditeurs de bases de données proposent des solutions parallèles comme InfoSphere d'IBM et Oracle.

La conception d'un entrepôt de données parallèle passe par cinq phases principales : (i) le choix de l'architecture matérielle, (ii) la fragmentation de l'entrepôt de données, (iii) l'allocation (ou le placement) de fragments générés par le processus de la fragmentation, (vi) la répartition des charges et (v) le traitement des requêtes.

Trois architectures principales existent pour supporter une base de données (ou un entrepôt de données) parallèle : (1) l'architecture à mémoire partagée (shared-memory), (2) l'architecture à disques partagés (shared disk) et (3) l'architecture distribuée (shared nothing).

Dans le *shared memory*, chaque processeur peut accéder à n'importe quel module de la mémoire ou à n'importe quelle unité du disque à travers un réseau d'interconnexion. Cette architecture permet un accès rapide aux données et assure un bon équilibrage de charges mais elle présente les inconvénients suivants : (a) la complexité du réseau d'interconnexion, due à la nécessité de relier chaque processeur à tous les modules, augmente le coût d'une telle architecture et (b) la limite d'extension du nombre de processeurs qui pénalise la disponibilité des données. Cette architecture a été utilisée par le système Volcano (Graefe, 1994). Ce type de système représente cependant un excellent rapport coût/performance pour des applications de bases de données de taille moyenne (Bouganim, 1996).
Dans le *shared disk*, chaque processeur a accès à n'importe quelle unité de disque à travers le réseau d'interconnexion mais n'a accès qu'à sa mémoire locale. Cette architecture assure un bon équilibrage de charges, une bonne possibilité d'extension du nombre de processeurs et une grande disponibilité des données. Mais, elle souffre du problème du goulot d'étranglement potentiel ainsi que le surcoût de maintenance pour assurer la cohérence des copies caches. (Stöhr et al., 2000) a utilisé cette architecture dans le contexte des entrepôts de données parallèles.
Dans le *shared nothing*, excepté le réseau d'interconnexion, rien n'est partagé entre les processeurs. Ainsi, chaque noeud de la machine correspond à un processeur avec ses disques et sa mémoire locale. Les noeuds communiquent entre eux par envois de messages via un réseau d'interconnexion permettant des échanges à haut débit (Bouganim, 1996). Cette architecture assure une bonne disponibilité des données et la possibilité d'une bonne extension. Par contre, elle présente une mauvaise réparation de charges. De nombreux systèmes de gestion de bases de données parallèles exploitent ce type d'architecture comme TANDEM, Bubba, Gamma, Teradata, etc. Dans le contexte des entrepôts de données, cette architecture a été adoptée par (Furtado, 2004). Cette architecture a été recommandée pour les entrepôts de données relationnels modélisés par des schémas en étoile par (DeWitt et al.). En conséquence, nous la considérons dans ce papier.

Une fois l'architecture matérielle choisie, le concepteur fragmente son entrepôt de données en un ensemble de partitions. La fragmentation est une pré-condition dans la conception des entrepôts parallèles. Cette fragmentation peut être horizontale (selon ses instances) ou verticale (selon ses attributs). La fragmentation horizontale est souvent utilisée pour concevoir des bases de données parallèles (Stöhr et al., 2000) et (Furtado, 2004).

Le placement des données est le processus affectant les fragments générés par la fragmentation sur les noeuds d'une machine parallèle. L'allocation peut être soit redondante (les fragments sont répliqués sur les sites/noeuds) ou non redondante (chaque fragment réside dans un et un seul noeud/site).

Une fois l'allocation réalisée, les requêtes globales seront réécrites en fonction des fragments. Lors du traitement de requêtes, nous devons vérifier si les noeuds de la machine parallèle sont uniformément chargés.

En nous penchant sur la littérature sur les travaux concernant la conception des bases de données parallèles en général et les entrepôts de données parallèles, en particulier, nous constatons que les deux processus de fragmentation et d'allocation se font d'une manière indépendante et séquentielle : le concepteur partitionne d'abord son entrepôt en utilisant son algorithme

favori de fragmentation (le schéma de fragmentation doit optimiser un ensemble de requêtes) ensuite il alloue les fragments sur des noeuds en utilisant un algorithme favori d'allocation (le schéma d'allocation généré doit optimiser le même ensemble de requêtes exécutées sur les différents noeuds). Cette indépendance a fait naître deux communautés de recherche ; l'une travaille que sur la sélection de schéma de fragmentation (Özsu et Valduriez, 1999), (Bellatreche et al., 2006), (Chakravarthy et al., 1994), (Mahboubi et Darmont, 2008), (Navathe et Ra, 1989) et l'autre travaille sur le problème de placement (Apers, 1988), (Karlapalem et Pun, 1997), (Mehta et DeWitt, 1997), (Menon, 2005). Cette communauté ne se soucie pas de la génération des fragments, elle suppose leur existence. L'inconvénient majeur de cette conception est son ignorance de l'interdépendance entre la fragmentation et l'allocation. Étant donné que les fragments générés par le processus de fragmentation sont l'une des entrées du problème d'allocation et que la fragmentation et l'allocation cherchent souvent à optimiser un ensemble de requêtes, nous proposons dans cet article une approche traitant conjointement le problème de fragmentation et de placement. Un algorithme d'équilibrage de charge entre les noeuds est proposé. A notre connaissance, ce travail est le premier qui offre une solution complète de conception d'un entrepôt de données parallèles : fragmentation, allocation et équilibrage de charge.

Ce papier est divisé en six sections : la section 2 résume l'approche itérative de conception d'un entrepôt de données parallèle, plus précisément le problème de fragmentation horizontale dans un entrepôt de données centralisé et le problème d'allocation. La section 3 décrit notre démarche de conception conjointe d'un entrepôt de données parallèle, où un algorithme génétique est décrit. La section 4 décrit l'algorithme de traitement de requêtes sur les fragments alloués sur une machine parallèle de type Shared Nothing qui équilibre la charge. La section 5 montre les résultats expérimentaux obtenus en utilisant le banc d'essai APB-1 release II. La section 6 conclut le papier en récapitulant les résultats principaux et en suggérant des travaux futurs.

2 État de l'Art

Dans cette section, nous décrivons les principaux travaux sur la fragmentation et l'allocation proposés dans le cadre des entrepôts de données parallèles (Furtado, 2004), (Stöhr et al., 2000) et (Stöhr et Rahm, 2001).

Dans (Stöhr et al., 2000), les auteurs proposent une approche de construction et d'exploitation d'un entrepôt de données sur une machine parallèle de type shared disk ayant K disques. L'entrepôt considéré est modélisé par un schéma en étoile caractérisé par une table de faits volumineuse et un nombre de tables de dimension de petite taille. Ils ont proposé une approche de fragmentation qui décompose la table des faits en utilisant une méthode de partitionnement appelée *Fragmentation Hiérarchique Multidimensionnelle*. Elle consiste à fragmenter la table de faits en utilisant plusieurs attributs de tables de dimension. Chaque table de dimension est fragmentée virtuellement en utilisant le mode *intervalle* (Range partitioning) sur des attributs appartenant à des *niveaux plus bas de la hiérarchie*. Les tables dimensions et leurs index (B*-arbre) sont répliquées sur chaque disque de la machine parallèle. Pour accélérer les requêtes, des index de jointure en étoile entre les tables de dimension et la table des faits sur des attributs appartenant à des *niveaux plus hauts de la hiérarchie* sont définis.

Le processus d'allocation concerne alors les fragments de la table des faits et les index de jointure définis. Notons que le nombre de fragments générés N peut être largement supérieur au nombre de disques K. Pour soutenir un degré de parallélisme élevé et un bon équilibrage de la charge, une allocation circulaire des fragments de la table des faits sur les disques est utilisée. Les index de jointure binaires définis sur le même fragment sont placés consécutivement sur les disques afin de permettre un parallélisme intra-requêtes. Par exemple, si le fragment $frag_i$ de la table des faits est placé sur le disque j, tous les k index de jointure qui lui sont associés sont placés sur les disques $j, j+1, ..., j+k-1\ modulo\ d$. A la fin de leur papier, certaines directives pour les concepteurs d'entrepôts de données parallèles ont été recommandées : exclure tous les schémas de fragmentation qui ne respectent pas les contraintes suivantes : (i) le nombre des index binaires autorisés que l'administrateur souhaite matérialiser, (ii) le nombre de fragments des tables des faits générés que l'administrateur souhaite avoir, (iii) la taille de disque de chaque noeud, (vi) éviter que le nombre de fragments générés soit inférieur au nombre de disques. Malheureusement, ces recommandations n'ont pas toutes été prises en considération dans leurs algorithmes de fragmentation et d'allocation. Dans le Warlock (Stöhr et Rahm, 2001), certaines recommandations, comme le nombre des fragments de la table des faits ont été prises en compte.

Dans (Furtado, 2004) ; une stratégie de placement de données consistant à partitionner horizontalement la table des faits et les grosses tables de dimension ensuite les allouer sur les noeuds en utilisant une distribution circulaire ou aléatoire est proposée. Les tables de petite taille sont répliquées sur tous les noeuds. Les tables de dimension plus importantes sont fragmentées par hachage selon leurs clés primaires. La table des faits est à son tour partitionnée par hachage en utilisant les clés étrangères. Cette fragmentation ne prend pas en considération des exigences de requêtes de jointure en étoile ((i) des sélections sur les tables de dimension et (ii) des jointures entre la table des faits et les tables de dimension). (Mahboubi et Darmont, 2008) proposent une approche de fragmentation des entrepôts de données XML. Les fragments générés sont ensuite alloués sur des grilles de calcul. (Lima et al., 2009) proposent une approche de conception d'un entrepôt relationnel sur clusters de données. Un cluster de base de données est un ensemble de machines chacune a son propre SGBD, inter connectées par une couche middleware. Elles gèrent une base de données et exécutent des requêtes. Le problème traité par (Lima et al., 2009) est comment allouer d'une manière efficace les données sur les différentes machines. Deux solutions peuvent exister : répliquer totalement la base de données sur toutes les machines, ou partitionner les données sur les machines. Leur contribution consiste à combiner la fragmentation et la réplication. La table des faits est partitionnée et les tables de dimensions sont totalement répliquées. Les auteurs ne contrôlent pas le nombre de fragments générés comme dans (Stöhr et al., 2000).

Dans les travaux cités, les processus de partitionnement et d'allocation se font d'une manière séquentielle (itérative).

3 Les Étapes de l'Approche Itérative

Dans cette section, nous décrivons brièvement les principes de base des deux processus de la conception itérative ; à savoir la fragmentation et l'allocation.

La fragmentation horizontale constitue le coeur de la conception d'un entrepôt de données parallèle. Dans le contexte des entrepôts de données relationnels, elle consiste à dé-

composer réellement ou virtuellement les tables de dimension en utilisant la fragmentation primaire (en exploitant les prédicats de sélection définis dans les requêtes) ensuite utiliser leurs schémas de fragmentation pour décomposer la table des faits (Bellatreche et al., 2008; Stöhr et al., 2000). Cette méthodologie de fragmentation peut générer un nombre important de fragments de la table des faits (Bellatreche et al., 2008). Pour éviter l'explosion de ce nombre, nous offrons à l'Administrateur de l'Entrepôt de Données (DBA), la possibilité de choisir un nombre des fragments maximal (seuil W) qu'il souhaite avoir (pour plus de détails sur cette méthodologie, consulter (Boukhalfa et al., 2008)). Pour illustrer cette démarche de fragmentation, nous considérons un schéma en étoile avec trois tables de dimension : Client, Temps et Produit et une table de fait Ventes. Supposons que la table de dimension Client est fragmentée en trois fragments $Client_1$, $Client_2$ et $Client_3$ définis par les clauses suivantes : $Client_1 = \sigma_{Ville=''Alger''}(Client)$, $Client_2 = \sigma_{Ville=''Paris''}(Client)$ et $Client_3 = \sigma_{Ville=''Montpellier''}(Client)$. En conséquence, la table peut être fragmentée en trois fragments $Vente_1$, $Vente_2$ et $Vente_3$ définis comme suit : $Vente_i = Vente \ltimes Client_i$ $(1 \leq i \leq 3)$ ($\ltimes$ représente l'opération de semi jointure). Cette méthodologie de fragmentation a été proposée dans le contexte centralisé, où l'objectif principal est d'optimiser une charge de requêtes $Q = \{Q_1, Q_2, ..., Q_n\}$ (chaque requête Q_i a une fréquence d'accès f_i). Plusieurs algorithmes de fragmentation horizontale ont été proposés dans la littérature (Bellatreche et al., 2006, 2008; Mahboubi et Darmont, 2008; Stöhr et al., 2000; Furtado, 2004). Le problème de fragmentation dans les entrepôts de données est NP-complet (Bellatreche et al., 2008). Soit $F = \{F_1, ..., F_K\}$ l'ensemble des fragments horizontaux générés par le processus de fragmentation.

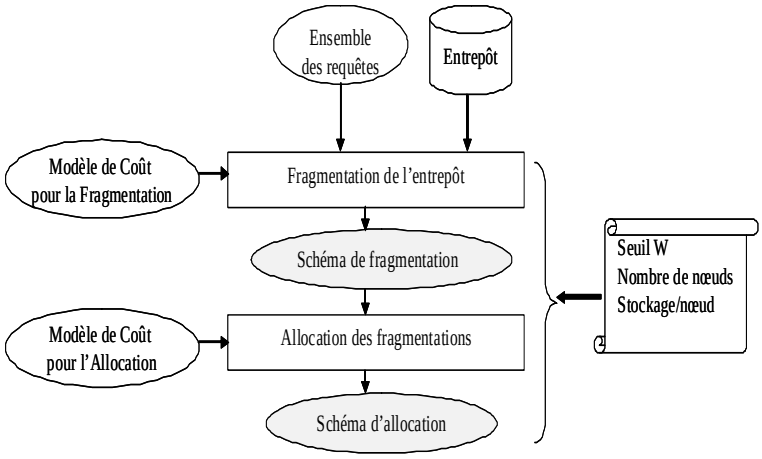

FIG. 1 – *Les étapes de l'approche de conception itérative*

Le processus de placement quant à lui consiste à trouver une distribution optimale de F sur l'ensemble de noeuds N de la machine parallèle afin d'améliorer la performance de l'ensemble de requêtes. Ce problème d'allocation est NP-complet (Saccà et Wiederhold, 1985; Karlapalem et Pun, 1997). Cependant, la plupart des approches existantes d'allocation dans le contexte des bases de données parallèles alloue des fragments en utilisant des techniques comme le

round robin (placement circulaire) ou le placement par hachage (Furtado, 2004). La Figure 1 résume l'approche itérative de la conception des entrepôts de données parallèles. Une des conséquences de cette conception itérative est l'utilisation de deux modèles de coût ; un utilisé pour assurer la qualité du schéma de fragmentation et le deuxième assurant un placement optimal (ou quasi optimal) des fragments sur les noeuds.

4 Notre Approche de Conception d'un Entrepôt Parallèle

Dans cette section, nous présentons une formalisation de notre approche de conception des entrepôts de données parallèles, où la fragmentation et l'allocation se font simultanément comme un problème d'optimisation à contrainte ensuite un algorithme de fragmentation et d'allocation pour le résoudre.

4.1 Formalisation de Notre Problème

Étant donné :

- Un entrepôt de données composé de d tables de dimension $\{D_1, ..., D_d\}$ et d'une table de faits F. Comme dans (Furtado, 2004; Lima et al., 2009), nous supposons que toutes les tables de dimension sont répliquées sur tous les noeuds et résident dans leurs mémoires centrales.
- Un ensemble de charge de requêtes $Q = \{Q_1, Q_2, ..., Q_n\}$ exécuté sur une machine parallèle Shared Nothing à M noeuds $N = \{N_1, N_2, ..., N_M\}$. Chaque noeud N_i $(1 \leq i \leq M)$ possède une capacité de stockage ST_i.
- Une contrainte de maintenance W représentant le nombre de fragments de l'entrepôt que le DBA considère pertinents pour le processus d'allocation. Ce nombre doit être largement supérieur au nombre de noeuds (voir les recommandations de (Stöhr et al., 2000) dans Section 2).

Le problème de conception d'un entrepôt de données parallèle consiste à fragmenter la table des faits en N fragments et à les allouer simultanément afin de réduire le coût d'exécution de requêtes sur les noeuds de la machine parallèle. Figure 2 illustre les étapes de notre approche. Nous constatons l'existence d'un seul modèle de coût contrairement à l'approche itérative.

Pour répondre à ce problème, nous proposons dans la section suivante un algorithme de sélection des schémas de fragmentation et d'allocation.

4.2 Algorithme de Conception

Étant donné que notre démarche partitionne l'entrepôt de données et en même temps, elle alloue les fragments, il nous faut une procédure de fragmentation et une autre d'allocation. Pour la fragmentation, nous avons adopté notre algorithme génétique pour fragmenter l'entrepôt de données (Bellatreche et Boukhalfa, 2005). Les grandes étapes de cet algorithme sont décrites dans la Figure 3.

Au démarrage de l'algorithme génétique, nous générons une population initiale d'une manière aléatoire. Pour chaque individu généré [1], nous vérifions s'il satisfait la contrainte de

[1] Un individu représente un schéma de fragmentation de la table des faits et les tables de dimension

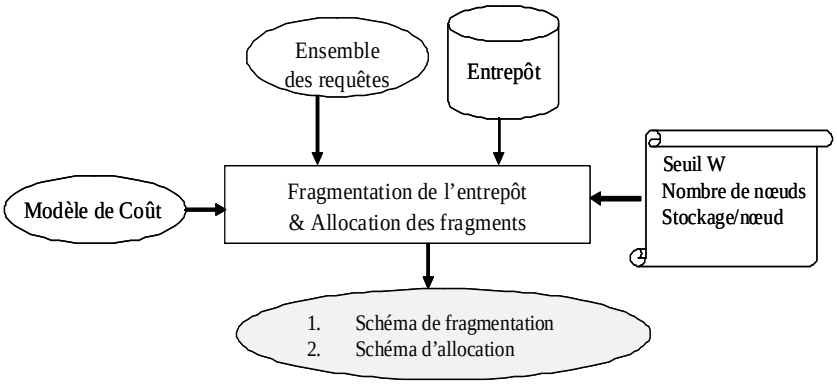

FIG. 2 – *Les étapes de l'approche de conception conjointe*

maintenance (W). Si c'est le cas, il sera gardé dans la population. Dans le cas contraire, des opérations de fusions sont effectuées afin de réduire son nombre de fragments. Une fois la population initiale créée, notre algorithme effectue des opérations génétiques (croisement et mutation) afin d'améliorer la population. Il est à noter que nous gardons le même principe de sélection, croisement et mutation que dans (Bellatreche et Boukhalfa, 2005). L'application de ces opérations est contrôlée par une fonction d'évaluation. Dans (Bellatreche et Boukhalfa, 2005), cette dernière calcule le coût d'exécution de la charge de requêtes en utilisant le schéma de fragmentation proposé par chaque individu. Dans notre cas, cette fonction alloue les fragments de chaque individu sur les différents noeuds en calculant le coût d'exécution de la charge de requêtes sur les fragments placés. A la fin, nous considérons l'individu générant un bon schéma de placement. Dans la section suivante, nous décrivons la procédure d'allocation utilisée pour placer chaque chromosome.

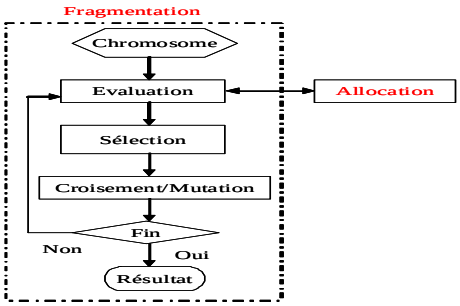

FIG. 3 – *Étapes de base d'un Algorithme Génétique*

4.3 Procédure d'allocation des fragments

Lors de l'évaluation de chaque solution candidate de la fragmentation, nous allouons ses fragments sur les différents noeuds. Cette allocation est basée sur les affinités entre les fragments. Nous avons emprunté cette idée de travaux de (Navathe et Ra, 1989) consacrés à la génération des fragments verticaux des tables traditionnelles [2]. Contrairement à (Navathe et Ra, 1989) qui regroupent des attributs ayant des grandes affinités dans le même fragment vertical, nous regroupons les fragments de la table des faits ayant une faible affinité. Ce type de regroupement augmente le parallélisme entre les noeuds. Les étapes de notre algorithme d'allocation sont :

1. *Construction de la Matrice d'Usage des Fragments (MUF)* : elle décrit l'utilisation des fragments de la table des faits par les requêtes. Les lignes et les colonnes de cette matrice représentent respectivement les requêtes et les fragments. La valeur de l'élément de la ligne i et la colonne j a pour valeur 1 si le fragment j est accédé par la requête i, sinon celle valeur est nulle. Cette matrice est complétée par une valeur de fréquence d'accès pour chacune des requêtes présentée dans une colonne supplémentaire.

2. *Construction de la Matrice d'Affinité des Fragments (MAF)* : elle a N lignes et N colonnes correspondant aux fragments de la table des faits. La valeur de l'élément de la ligne i et la colonne j reporte les valeurs d'affinité définis entre ces fragments. Cette affinité correspond à la somme des fréquences d'accès des requêtes accédant simultanément aux deux fragments. Cette matrice est symétrique.

3. *Regroupement des fragments* : pour regrouper des fragments, nous générons un graphe complet et étiqueté, appelé "graphe d'affinité des fragments". Les noeuds de ce graphe représentent les fragments. Une arête entre deux fragments représente la valeur d'affinité. Nous parcourons ce graphe pour former des cycles, où chacun contient des fragments ayant une faible affinité. Chaque cycle devient l'unité d'allocation.

4. *Construction de la Matrice d'Allocation des Fragments (MALF)* : elle représente la présence des fragments sur les noeuds. Les lignes et les colonnes de cette matrice sont associées aux N fragments obtenus par le schéma de fragmentation en cours d'évaluation et les M noeuds. L'algorithme ci-dessous remplit la MALF.

A partir de l'ensemble de cycles $C = \{C_1, ..., C_k\}$, la procédure d'allocation place les cycles d'une manière circulaire sur les noeuds. Chaque fragment appartient à un seul cycle et le cycle est alloué à un seul noeud. Notre unité de placement circulaire est le cycle (qui contient un ensemble de fragments), contrairement aux travaux existants (Stöhr et al., 2000; Furtado, 2004), où l'unité de placement est un fragment.

Une fois la table d'allocation des fragments construite, nous calculons le coût d'exécution de la charge Q sur les M noeuds. Nous ne prenons pas en compte le temps CPU et le coût de communication que nous supposons négligeables par rapport au temps nécessaire pour effectuer les entrées sorties (E/S). En conséquence, notre modèle de coût estime le nombre d'E/S nécessaires (exprimé en nombre de pages chargées) pour exécuter la charge de requêtes exécutée sur la machine parallèle. Ce coût est donné par :

$$\sum_{k=1}^{n} \sum_{j=1}^{M} \sum_{i=1}^{N} MUF_{ki} \times MALF_{ij} \times |Fi| \quad (1)$$

[2] Un fragment vertical est constitue d'un ensemble d'attributs de la table à fragmenter

Entrée : $C = \{C_1, ..C_k\}$ ensemble de cycles, M
Sortie : MALF
$indicenode := 0$;
pour *chaque* $C_i \in C$ **faire**
  pour *chaque* $e \in F$ **faire**
    si $e \in C_i$ **alors**
      $MALF[e][indicenode] := 1$
    sinon
      $MALF[e][indicenode] := 0$
    fin
  fin
  $indicenoeud := indicenoeud + 1$;
  si $indicenoeud = M$ **alors**
    $indicenoeud := 0$
  fin
fin

Algorithme 1: Algorithme d'allocation des cycles

Avec n, M, N et $|Fi|$ représentent respectivement le nombre de requêtes, le nombre de noeuds, le nombre des fragments et taille du fragment F_i. Ceci revient à minimiser :

$$\sum_{k=1}^{n} max(\sum_{j=1}^{M}\sum_{i=1}^{N} MUF_{ki} \times MALF_{ij} \times |F_i|) \quad (2)$$

5 Traitement des requêtes et équilibrage de charges

Dans cette section, nous nous intéressons au problème d'équilibrage de charge entre les différents noeuds de la machine parallèle. Nous supposons que la fragmentation et l'allocation soient établies. Pour exécuter une requête donnée, on doit d'abord identifier les partitions valides ensuite leurs localisations. L'architecture orchestrant le processus d'exécution de requêtes est décrite dans la Figure 4. Cette architecture contient un noeud de soumission, également appelé coordinateur dont le rôle est la vérification si la charge de requêtes est distribuée d'une manière équitable sur l'ensemble des noeuds. Si c'est le cas, il décompose la requête en sous requêtes, et il les envoie sur leurs noeuds d'exécution. Dans le cas contraire, il redistribue la charge et il envoie les sous requêtes. Une fois que le traitement s'achève au niveau d'un noeud de traitement, ce dernier transmet son résultat au noeud de soumission (le résultat transmis représente un résultat partiel pour la requête). Le noeud de soumission s'occupe de la collecte des résultats ainsi que leur fusion pour avoir le résultat final de l'exécution de la requête.

L'équilibrage de charge représente un enjeu important pour atteindre une haute performance d'un entrepôt de données parallèle. Si ce dernier n'est pas équilibré, nous devons redistribuer les fragments des noeuds surchargés dénotés par $NSUR$ sur les noeuds sous chargés dénotés par $NSOUS$. Cette distribution peut être formalisée comme suit :
Étant donné la matrice d'allocation des fragments $MALF$, l'ensemble de M noeuds, une matrice de communication ayant M lignes et M colonnes correspondant aux noeuds. La valeur

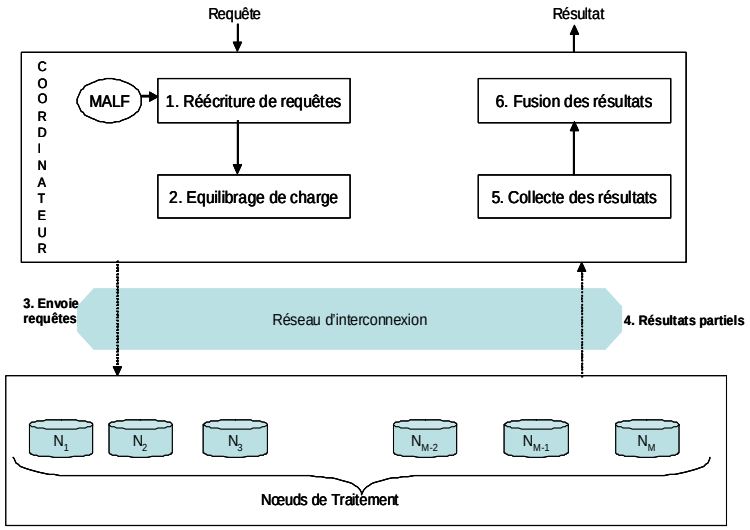

FIG. 4 – *Traitement de Requêtes et Répartition des Charges*

de l'élément de la ligne i et la colonne j reporte les valeurs le coût de transfert entre les deux noeuds. Le problème de la répartition de la charge consiste à offrir un système équilibré assurant un coût de communication minimal. Pour résoudre ce problème, quelques définitions s'imposent.

Définition 1 *Un noeud est dit surchargé si son niveau de chargement*[3] *est supérieur à la moyenne de chargement* [4].

Définition 2 *Un noeud est dit normalement chargé si son niveau de chargement est égal à la moyenne de chargement.*

Définition 3 *Un noeud est dit sous chargé si son niveau de chargement est inférieur à la moyenne de chargement.*

Ces définitions nous permettent de classer les noeuds sur chargés et sous chargés. Une fois classés, nous effectuons la migration de fragments de $NSUR$ vers $NSOUS$ à l'aide d'un algorithme baptisé *algo_migration_dynam*. Chaque noeud de $NSUR$ N_i est associé à un poids $PSUR(N_i)$ représentant le nombre de requêtes qui le rend surchargé. $P(N_i)$ calculée comme suit : $PSUR(N_i)$ = niveau de chargement(N_i)- moyenne de chargement. D'une manière identique, chaque noeud de $NSOUS$ est associé à poids $PSOUS(N_j)$ représentant le nombre de requêtes qu'il peuvent encore recevoir. $PSOUS(N_j)$ calculée comme suit : $PSOUS(N_i)$ = moyenne de chargement - niveau de chargement(N_j). Maintenant nous avons tous les ingrédients pour proposer notre algorithme équilibrant les charges des noeuds.

[3]Niveau de chargement d'un noeud N = nombre de fragments participants à l'exécution d'une requête Q/ nombre de noeuds participants.

[4]Moyenne de chargement= nombre de fragments participants / nombre de noeuds de la machine parallèle

1. Rechercher le noeud $N_j \in NSUR$ ayant la plus grande priorité ;
2. Rechercher le noeud $N_k \in NSOUS$ qui minimise le coût de redistribution CT entre N_k et N_j
3. Transférer le(s) fragment(s) F alloué sur N_j à N_k en mettant à jour la matrice d'allocation des fragments $MALF$
4. Calculer la priorité $PSUR(N_j)$:
si $PSUR(N_j) = 0$ **alors**
  Supprimer N_j de $NSUR$ et N_k de $NSOUS$; aller à 5
sinon
  Supprimer N_k de $NSOUS$ et aller à 2
fin
5. **si** *le système est équilibré*[5] **alors**
  aller à fin
sinon
  aller à 1
fin

Algorithme 2: Répartition de charges

Cet algorithme calcule le coût de transmission (CT) et il l'inclut lors de l'exécution de la requête

$$CT = \sum_{j=1}^{g_1} \sum_{m=1}^{g_2} \sum_{l=1}^{g_3} |F_l| \times CC(j,m) \tag{3}$$

Avec : g_1, g_2, g_3 et CC représentent respectivement le nombre de noeuds de S, le nombre de noeuds de D participants dans l'équilibrage de la charge du noeud j, le nombre de fragments alloués sur le noeud j transféré à m et la matrice de transmission.

6 Étude expérimentale

Notre programme de simulation a été développé dans un environnement Windows XP en Java sur une machine Pentium IV 2.8 GHZ ayant 1 Go de RAM. Nous avons utilisé le banc d'essai APB-1 release II (Council, 1998) qui utilise un schéma en étoile contenant quatre tables de dimension Prodlevel (9 000 tuples), Custlevel(900 tuples), Timelevel (24 tuples), Chanlevel (9 tuples) et une table des faits Actvars (24 786 000 tuples).

Nous avons considéré 55 requêtes de recherche composées d'un seul bloc (pas de requêtes imbriquées) avec 40 prédicats de sélection définis sur 9 attributs (ClassLevel, GroupLevel, FamilyLevel, LineLevel, DivisionLevel, YearLevel, MonthLevel, RetailerLevel, AllLevel). Les domaines de valeurs de ces attributs sont décomposés en 4, 2, 5, 2, 4, 2, 12, 4, et 5 sous domaines. Chaque prédicat possède un facteur de sélectivité calculé sur l'entrepôt réel.

Nous considérons une machine parallèle Shared Nothing de 12 noeuds. La taille de la page système au niveau de chaque noeud est de 65536 Octets. Nous avons utilisé un taux de croisement de 80%, un taux de mutation de 20% pour améliorer 20 chromosomes des 10 générations comme paramètres de l'algorithme générique

La première expérience que nous avons effectuée étudie l'impact du critère de construction de cycle de fragments sur la performance de requêtes. Rappelons que l'unité d'allocation de notre procédure de placement est un cycle de fragments. Pour générer un cycle, notre procédure choisit aléatoirement un fragment ensuite elle essaye d'enrichir le cycle par l'ajout

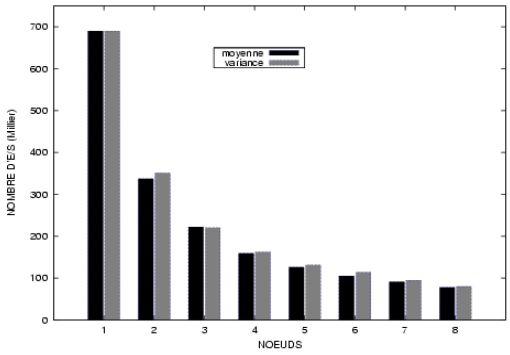

FIG. 5 – *Étude de l'impact du Seuil de Génération des Cycles : Variance vs. Moyenne*

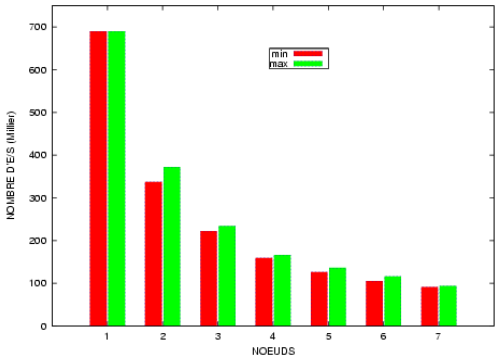

FIG. 6 – *Cycles avec basse affinité vs. Cycles avec haute Affinité*

d'autres fragments. Un nouveau fragment fera partie du cycle initial s'il vérifie un critère d'acceptation. Dans notre travail, nous avons utilisé deux critères d'acceptation : la *moyenne* et la *variance*. Pour accepter un nouveau fragment dans le cycle initial il faut que son affinité soit inférieure ou égale à la moyenne (ou la variance) des noeuds formant le cycle initial. Pour réaliser cette étude, nous avons fixé le seuil de fragmentation à 200 ($W = 200$), et nous avons varié les noeuds de 1 à 8 et pour chaque valeur nous calculons le nombre d'E/S nécessaire pour l'exécution des 55 requêtes.

Figure 5 montre que les deux critères (moyenne et la variance) donnent pratiquement les mêmes résultats, avec un léger avantage à la moyenne. En conséquence, nous utilisons ce critère dans la suite des expérimentations.

Nous avons également étudié l'impact de critères de regroupement des fragments. L'algorithme de (Navathe et Ra, 1989) que nous avons adapté à notre étude, génère des cycles, où chacun contient des attributs ayant une grande affinité. Dans notre cas, nous générons des cycles où chacun contient des fragments de faible affinité. Nous avons mené deux expérimentations pour étudier la qualité de notre approche selon le type d'affinité. Figure 6 présente les résultats de cette comparaison où nous avons considéré un seuil de fragmentation de 200. Nous avons fait varier le nombre de noeuds de 1 à 7 et pour chaque valeur, nous calculons le nombre d'E/S nécessaire pour exécuter les 55 requêtes par chaque algorithme. Nous remarquons que l'allocation avec basse affinité est légèrement meilleure que celle avec haute affinité. Cela est dû au fait que le regroupement selon les affinités les plus basses augmente le degré du parallélisme. En conséquence, nous optons pour un regroupement à basse affinité pour le reste des expérimentations.

Figure 7 compare la performance de l'approche itérative et l'approche conjointe. Pour cela, nous calculons le facteur de rapidité (speed-up) qui mesure la diminution du temps de réponse d'une requête pour une taille d'un entrepôt de données constante et une augmentation des capacités de la configuration (nombre de noeuds). Pour un seuil de fragmentation de 200, nous avons fait varier le nombre de noeuds de 1 à 12 et pour chaque valeur, nous calculons le nombre de speed up pour chaque algorithme. Nous remarquons que l'approche conjointe détient un speed up linéaire et l'approche itérative possède un speed up sous linéaire. Ce résultat

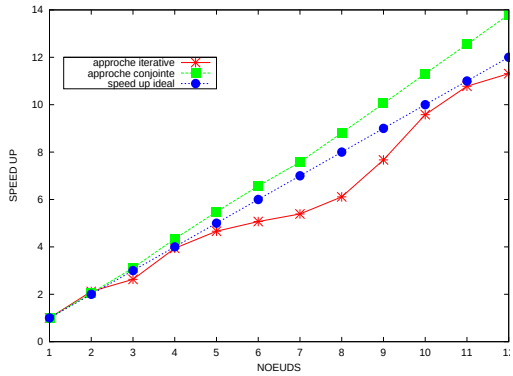

FIG. 7 – *Comparaison du speed up des approches itérative et conjointe*

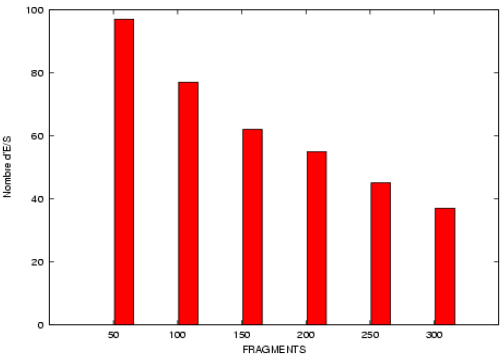

FIG. 8 – *Effet du Seuil de Fragmentation sur la Performance*

confirme que l'approche conjointe est plus adaptée à la conception d'un entrepôt de données parallèle que l'approche itérative. De plus, le speed up de l'approche combinée n'est pas idéal (speed(p)=p) cela est dû à l'absence d'un module d'équilibrage de charge.

Figure 8 étudie l'effet du seuil de fragmentation sur la performance de notre approche. Pour cela, nous fixons le nombre de noeuds à 10 et nous faisons varier le seuil de fragmentation W de 50 à 300. Nous remarquons que l'augmentation du seuil de fragmentation améliore considérablement la performance des requêtes car en relâchant W, plus d'attributs sont utilisés pour fragmenter l'entrepôt. Lorsque W est grand, les domaines sont décomposés en plus de partitions et donc chaque partition est moins volumineuse. Cela implique moins de données chargées pour exécuter les requêtes utilisant les attributs de fragmentation. En conséquence, nous concluons que le nombre de fragments et d'E/S sont proportionnels au nombre d'attributs de fragmentation utilisés et le nombre de noeuds de la machine parallèle.

Figure 9 étudie la charge des noeuds (qui sont 8). Nous supposons que les schémas de fragmentation et d'allocation sont générés. Nous remarquons que la charge de travail n'est pas distribuée d'une manière équilibrée. Cette expérimentation confirme la nécessité d'un module d'équilibrage de charge ; le noeud N4 par exemple est moins chargé que les deux premiers.

Figure 10 étudie le comportement de notre approche avec la prise en compte du module équilibrage de charges. Pour un seuil de fragmentation de 200, nous avons varié le nombre de noeuds de 1 à 12 et pour chaque valeur nous calculons le facteur de rapidité (speedup) pour exécuter les 55 requêtes après l'ajout d'équilibrage de charges. Nous remarquons que le speed up est linéaire pour les deux cas. Le speed up pour l'approche combinée avec équilibrage de charge est prés d'être idéale vue que nous négligeons le coût de communication.

7 Conclusion & perspective

La taille des entrepôts de données a explosé durant les dernières années. Afin d'accélérer les requêtes complexes, le traitement parallèle est devenu une nécessité. La conception d'un entrepôt de données parallèle passe par cinq étapes : le choix de l'architecture matérielle, la

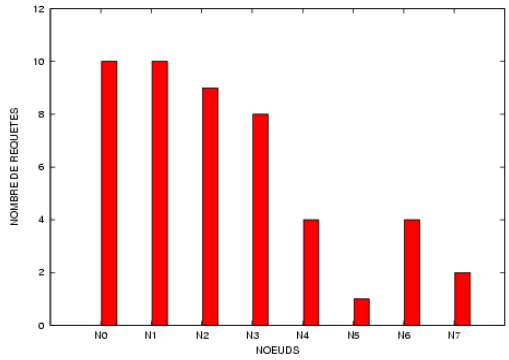

FIG. 9 – *Répartition des Charges*

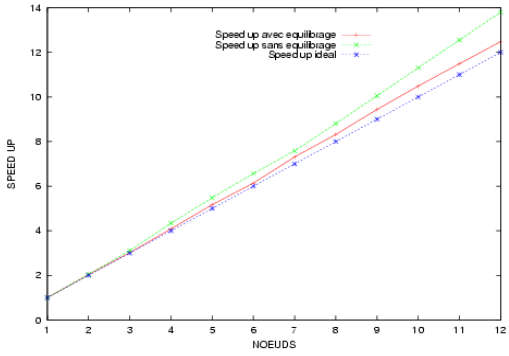

FIG. 10 – *Speedup des Approches Équilibrée et non Équilibrée*

fragmentation, l'allocation des données, le traitement des requêtes et la répartition de la charge entre les noeuds. La majorité des travaux sur la conception des entrepôts de données parallèles traite la fragmentation et l'allocation d'une manière séquentielle. Dans ce papier, nous avons montré l'intérêt de traiter ces deux problèmes d'une manière simultanée. Ensuite nous avons proposé un algorithme composé de deux procédures, une pour fragmenter l'entrepôt de données et l'autre pour allouer les fragments générés. La décision de placement se fait lors de la fragmentation. Un algorithme d'équilibrage de charge est présenté. Les algorithmes proposés sont validés en utilisant les données du banc d'essai ABP1. Les résultats obtenus sont encourageants. Pour Une fois le schéma d'allocation et de fragmentation sélectionné, nous avons présenté une méthode d'équilibrage de charge pour atteindre la haute performance du système. Notons que ce travail est, à notre connaissance, le premier qui propose une solution intégrant la fragmentation et l'allocation avec un équilibrage de charge.

Pour les travaux futurs, il serait intéressant d'adapter les algorithmes que nous avons proposés dans le contexte dynamique. Une autre piste intéressante à explorer est de formaliser notre problème comme un problème d'optimisation multi objectifs, i.e., le premier objectif représenterait le coût d'exécution de requêtes tandis que le deuxième représenterait l'équilibre de la charge.

Références

Apers, P. M. G. (1988). Data allocation in distributed database systems. *ACM Transactions on database systems 13*(3), 263–304.

Bellatreche, L. et K. Boukhalfa (2005). La fragmentation dans les entrepôts de données : une approche basée sur les algorithmes génétiques. In *Entrepôts de Données et Analyse en ligne (EDA'05) : Revue des Nouvelles Technologies de l'Information*, pp. 141–160.

Bellatreche, L., K. Boukhalfa, et H. I. Abdalla (2006). Saga : A combination of genetic and simulated annealing algorithms for physical data warehouse design. *in 23rd British National Conference on Databases (BNCOD'06)*, 212–219.

Bellatreche, L., K. Boukhalfa, et P. Richard (2008). Data partitioning in data warehouses : Hardness study, heuristics and oracle validation. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2008)*, pp. 87–96.

Bouganim, L. (1996). Equilibrage de charges lors de l'exécution de requêtes sur des architectures multiprocesseurs hybrides. Phd. thesis, Université de Versailles Saint Quentin en Yvelines.

Boukhalfa, K., L. Bellatreche, et P. Richard (2008). ragmentation primaire et dérivée : Étude de complexité, algorithmes de sélection et validation sous oracle10g. Techreport http ://www.lisi.ensma.fr/members/bellatreche, LISI/ENSMA.

Chakravarthy, S., J. Muthuraj, R. Varadarajan, et S. B. Navathe (1994). An objective function for vertically partitioning relations in distributed databases and its analysis. *in Distributed and Parallel Databases Journal 2*(2), 183–207.

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

DeWitt, D. J. D., S. Madden, et M. Stonebraker. How to build a high-performance data warehouse. *http ://db.lcs.mit.edu/madden/high_perf.pdf*.

Furtado, P. (2004). Experimental evidence on partitioning in parallel data warehouses. In *DOLAP*, pp. 23–30.

Graefe, G. (1994). Volcano - an extensible and parallel query evaluation system. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 6*(1), 120–135.

Karlapalem, K. et N. M. Pun (1997). Query driven data allocation algorithms for distributed database systems. *in 8th International Conference on Database and Expert Systems Applications (DEXA'97), Toulouse, Lecture Notes in Computer Science 1308*, 347–356.

Lima, A. A. B., C. Furtado, P. Valduriez, et M. Mattoso (2009). Improving parallel olap query processing in database clusters with data replication. *To appear in Distributed and Parallel Database Journal*.

Mahboubi, H. et J. Darmont (2008). Data mining-based fragmentation of xml data warehouses. In *ACM 11th International Workshop on Data Warehousing and OLAP (DOLAP'08)*, pp. 9–16.

Mehta, M. et D. J. DeWitt (1997). Data placement in shared-nothing parallel database systems. *VLDB Journal 6*(1), 53–72.

Menon, S. (2005). Allocating fragments in distributed databases. *IEEE Transactions on Parallel and Distributed Systems 16*(7), 577–585.

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

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

Saccà, D. et G. Wiederhold (1985). Database partitioning in a cluster of processors. *ACM Transactions on Database Systems 10*(1), 29–56.

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

273–284.

Stöhr, T. et E. Rahm (2001). Warlock : A data allocation tool for parallel warehouses. *Proceedings of the International Conference on Very Large Databases*, 721–722.

Summary

Traditionally, designing a parallel data warehouse consists in first fragmenting its schema and then allocating the generated fragments over the nodes of the parallel machine. The main drawback of such approach is its ignorance of the interdependency between fragmentation and allocation processes. Note that number of fragments is one of the inputs of the allocation problem. Both processes optimize the same set of queries defined on the parallel data warehouse. In this paper, we propose a new approach for designing parallel relational data warehouses supported by a shared nothing architecture, where fragmentation and allocation processes are done simultaneously. To ensure a high performance of queries, a load balancing method is proposed. Finally, a validation of our proposed algorithms using the APB-1 release II benchmark is presented.