Clé de partition multi-attributs pour un partitionnement horizontal optimal des entrepôts de données NoSQL en colonnes

Mohamed Boussahoua*, Fadila Bentayeb *, Omar Boussaid *, Nadia Kabachi **

*Université Lumière Lyon 2, ERIC EA 3083
5, avenue Pierre Mendês 69676 Bron-France
{Mohamed.Boussahoua, Fadila.Bentayeb, Omar.Boussaid}@univ-lyon2.fr
**Université Claude Bernard Lyon 1
43, boulevard du 11 novembre 1918, 69100, Villeurbanne-France
Nadia.Kabachi@univ-lyon1.fr

Résumé. Les systèmes NoSQL en colonnes offrent des techniques de stockage adaptées à la construction d'entrepôts de données. Plusieurs scénarios sont possibles pour stocker des entrepôts de données sur ces systèmes. Dans cet article, nous étudions une nouvelle approche de placement des données d'un entrepôt sur un cluster dans un système NoSQL en colonnes. Notre approche s'appuie sur une méthode basée sur une stratégie de regroupement d'attributs pour définir les clés de partition *RowKey* d'un schéma de partitionnement horizontal des données. Nous obtenons ainsi un modèle physique de données qui permet de garantir une localisation et une distribution plus homogène des données dans les différents nœuds du cluster. Nous utilisons la méthode des règles d'association pour obtenir les partitions sus-mentionnées. Pour évaluer notre méthode, nous avons effectué plusieurs tests sur le benchmark TPC-DS au sein du SGBD NoSQL HBase. Les résultats obtenus montrent que notre stratégie de placement des données augmente les performances des entrepôts NoSQL en colonnes de l'ordre de 48%.

1 Introduction

Le stockage et la gestion de données volumineuses, complexes, structurées ou non structurées constituent une tâche difficile dans les environnements informatiques actuels. Pour faire face à l'essor des volumes et de la diversité des données, de nouveaux systèmes de stockage et de traitement de données robustes et efficaces ont été mis en place par les institutions scientifiques et industrielles tels que MapReduce (Dean et al., 2001), C-Store (Stonebraker et al., 2005), Apache Hadoop [1] ainsi que les systèmes

1. http ://hadoop.apache.org/

NoSQL [2] etc. Aujourd'hui, l'utilisation des modèles NoSQL dans le domaine décisionnel constitue une bonne opportunité pour la construction des entrepôts de données capables de supporter de grandes masses de données dans un environnement fortement distribué. En effet, l'intérêt porté aux modèles NoSQL est motivé d'une part, par l'avènement des données massives (*big data*) et d'autre part, par l'incapacité du modèle relationnel, habituellement utilisé pour implémenter les entrepôts de données, à permettre le passage à très grande échelle notamment dans le contexte du *big data*. Par ailleurs, le stockage des données en colonnes est plus approprié à la structure des données multidimensionnelles et au calcul d'agrégats. En effet, il permet d'avoir, dans un même espace de stockage, les valeurs d'une même colonne de dimension (axe d'analyse) ou celles d'une même mesure (indicateur) à agréger (Stonebraker et al., 2005) réduisant ainsi les accès aux données tout en augmentant par conséquent les performances des systèmes décisionnels. D'autre part, pour créer un cube OLAP (*On-Line Analytical Processing*) à partir d'un entrepôt de données, il est nécessaire d'accéder à des données provenant à la fois de la table des faits et de plusieurs tables de dimensions en exécutant l'opération de jointure entre les tables. L'opération de jointure est une opération très coûteuse notamment dans le contexte des big data, c'est pourquoi plusieurs techniques d'optimisation de requêtes décisionnelles dans les entrepôts de données NoSQL en colonnes ont émergé ces dernières années (Romero et al., 2015), (Yang et al., 2015) et (Mior et al., 2017) pour améliorer la performance des requêtes décisionnelles.

Dans cet article, nous nous intéressons en particulier à la performance des entrepôts de données NoSQL en colonnes dans un environnement distribué. Notre objectif est de définir un bon schéma de partitionnement horizontal des données de telle façon à réduire le temps d'exécution des requêtes décisionnelles. Autrement dit, il faut trouver la meilleure stratégie de placement et de distribution des données sur les différents nœuds d'un cluster pour permettre de regrouper dans un même nœuds les valeurs des colonnes (dimensions, mesures) formant un même fait. Nous rappelons que le partitionnement horizontal des données dans un système NoSQL en colonnes se base sur le concept clé : *RowKey* (clé de partition). Nous proposons à cet effet, une stratégie de placement et de distribution des données entreposées basée sur la clé de partition *RowKey* définie par la concaténation de plusieurs attributs les plus fréquents utilisés par des prédicats employés dans une charge de requêtes. Chaque attribut pouvant appartenir à une table de dimension ou une table de faits. L'objectif étant de sélectionner une configuration optimale d'un *RowKey* multi-attributs permettant de colocaliser les blocs de données qui sont joignables entre elles afin de minimiser le temps d'exécution de la jointure dans les requêtes décisionnelles.

Pour atteindre notre objectif, notre stratégie de placement et de distribution des données entreposées dans les différents nœuds d'un cluster que nous baptisons *Balanced-CN-DW* suit les étapes suivantes : (1) à partir d'une charge de requêtes et en se basant sur des prédicats de sélection, il faut trouver le meilleur regroupement des attributs les plus fréquemment utilisés ensemble. Pour cela, nous utilisons un algorithme de fouille de données, à savoir l'algorithme des règles d'association ; (2) à partir des groupes d'at-

2. http ://nosql-database.org/

tributs obtenus, nous construisons les clés de partitionnement *RowKey* ; (3) enfin, en s'appuyant sur les *RowKey* obtenus, le chargement des données peut alors être réalisé. Par ailleurs, nous avons fait en sorte que les différents nœuds d'un cluster soient équilibrés en termes de volume de données.

Pour valider notre approche *Balanced-CN-DW*, nous avons construit au sein du SGBD HBase deux entrepôts de données NoSQL en colonnes à partir du banc d'essai TPC-DS (Transaction Processing Council Decision Support) et utilisé une charge de requêtes (extraite à partir de celle de TPC-DS) afin de mener des tests de performance. Les deux entrepôts de données ont été construits selon deux approches différentes : le premier entrepôt est construit selon notre approche *Balanced-CN-DW* et le deuxième est construit selon une clé de partition séquentielle. Nous avons ainsi réalisé plusieurs tests pour évaluer l'efficacité de notre approche. Nous avons étudié le gain en performance des requêtes en exécutant celles-ci sur les deux entrepôts de données NoSQL en colonnes. Nos tests montrent que notre approche *Balanced-CN-DW* de construction d'entrepôts de données NoSQL en colonnes améliore de façon significative le temps d'exécution des requêtes de l'ordre de 48%.

La suite de cet article est organisé comme suit. La section 2 est consacrée à la description détaillée des différents concepts liés à notre travail de recherche, à savoir, la clé de partition *RowKey* et le mode de partitionnement horizontal des données adopté par les systèmes NoSQL en colonnes. La Section 3 présente un état de l'art des travaux portant sur le développement des entrepôts de données selon le modèle NoSQL orienté colonnes et détaille la problématique liée à l'accès aux données. Dans la Section 4, nous détaillons notre approche de placement et de distribution des données *Balanced-CN-DW*. La Section 5 présente l'évaluation de notre approche. Enfin, nous concluons cet article et présentons quelques perspectives de recherche dans la section 6.

2 Systèmes NoSQL en colonnes

La conception d'un schéma de partition horizontal pour implémenter un entrepôt de données dans un système NoSQL en colonnes est très différente de celle d'un schéma de partitionnement horizontal dans un SGBD relationnel. En effet, lorsqu'on crée un schéma de partitionnement horizontal avec le modèle NoSQL en colonnes, on s'appuie sur un concept clé qui caractérise les systèmes NoSQL en colonnes : la clé de partition *RowKey*. Dans un SGBD NoSQL en colonnes comme HBase et Cassandra, les données sont stockées sous forme d'un ensemble de colonnes où chaque colonne stocke ses valeurs sous forme de *Clé/Valeur* (FIG. 1). La partie *Clé* de la valeur se compose d'un

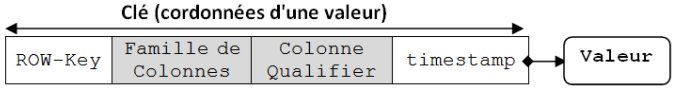

FIG. 1 – *Structure de stockage d'une donnée sous un SGBD NoSQL en colonnes*

identifiant (*RowKey*), du nom de la famille de colonnes (*Family*), du nom de la colonne (*Column*), et un horodatage (*timestamp*). La combinaison de ces informations permet d'identifier une valeur d'attribut, qui est l'équivalent de la clé primaire (*primary key PK*) d'une table relationnelle. Le *RowKey* est une partie de la clé primaire qui peut être vue comme une colonne spécifique dans une table NoSQL en colonnes, et qui est utilisée pour garantir le lien entre un ensemble de valeurs des attributs qui forme le même enregistrement, et de déterminer sur quel nœud du cluster cette ligne de données va être stockée. Pour détailler le concept de l'identifiant *RowKey*, prenons un exemple d'une base de données de "films" [3], composée de trois tables : la table des films, la table des artistes et la table des personnages qui stocke les correspondances entre les films et les artistes.

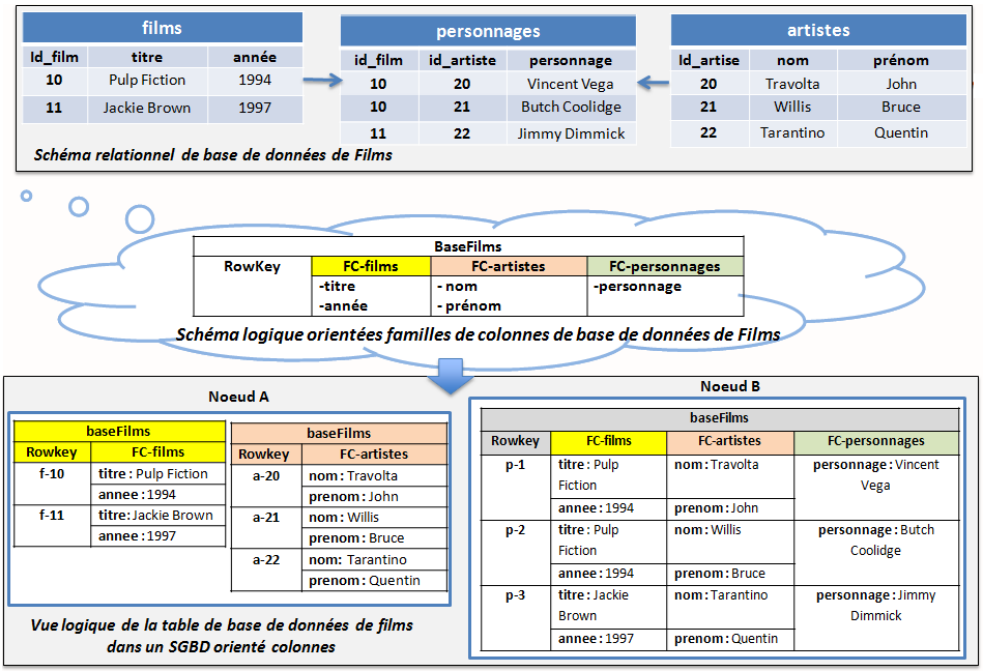

FIG. 2 – *Exemple de passage d'un schéma relationnel à un schéma NoSQL en colonnes*

Généralement, dans un SGBD NoSQL en colonnes, on stocke toutes les données d'une base de données dans une table unique composée de plusieurs familles de colonnes. Dans le cas de notre exemple, pour construire un schéma de stockage orienté colonnes, une solution de modélisation possible consisterait à dénormaliser la base de données de films. Les trois tables sont alors regroupées dans une table appelée (*baseFilms*). Celle-ci contient trois familles de colonnes : *FCfilms* regroupe les attributs *(titre, année)*, *FCartistes* regroupe les attributs *(nom, prénom)*, *FCpersonnages* composé d'un seul attribut *(personnage)*. De plus, la table *baseFilms* est composée d'une

3. http ://b3d.bdpedia.fr/docstruct.html

seule colonne appelée clé de partition (*RowKey*).

Pour simplifier le partitionnement des données de la table *baseFilms*, nous considérons un cluster basé sur un SGBD NoSQL en colonnes et composé de deux machines (*nœud A* et *nœud B*). Ainsi, pour localiser les données de la table des films dans le *nœud A*, nous utilisons une clé de partition composite, on ajoute le préfixe *f* (qui représente le nom de la table des films) à la clé primaire *Id_film* de la table films. De la même manière, pour localiser les données de la table des artistes dans le *nœud A*, on ajoute le préfixe *a* à la clé primaire *Id_artiste* de la table des artistes. Concernant la relation personnages, dans un SGBD relationnel, pour établir une liaison entre les données des tables des films et des artistes, cette relation est représentée par une table intermédiaire supplémentaire qui est la table des personnages, dont chaque ligne possède une combinaison unique des valeurs de colonnes *Id_film* et *Id_artiste*. Cependant, dans un SGBD NoSQL en colonnes, le mécanisme de liaison entre les données des tables est différent de celui du SGBDR et le concept de jointure n'existe pas (c'est l'un des grands avantages des SGBD NoSQL), mais ce manque est compensé par la dénormalisation des tables jointes pendant la phase de transformation, et par les données redondantes stockées dans des fragments différents. Rappelons qu'une table de données dans un SGBD NoSQL en colonnes peut avoir des lignes différentes dans différentes colonnes appartenant à différentes familles de colonnes. En d'autres termes, le principe consiste donc à ce que chaque famille de colonnes puisse retourner les colonnes nécessaires pour stocker les informations associées à chaque ligne de données en précisant les valeurs de la clé de partition. Donc d'une certaine manière, les lignes de données de la table personnages sont définies par une structure de colonnes qui est composée des trois sous-structures élémentaires *(titre, année), (nom, prénom), (personnage)*.

Dans la Figure 2, on remarque que les lignes ne contiennent pas les mêmes colonnes et que les données des trois tables ne sont pas mélangées dans le même fichier sur le disque. Par exemple, les lignes de données qui représentent les informations relatives aux films sont au même niveau d'accès. Dans le cas d'une consultation de l'historique de tous les titres des films, une requête sera donc faite sur la partition des films, ce qui ne nécessite pas d'accès aux autres partitions. Pour trouver les artistes correspondant aux différents personnages du film (*Pulp Fiction*), il faut formuler une requête qui est composée de colonnes provenant de trois familles de colonnes *FCfilms*, *FCartistes* et *FCpersonnages*, afin de récupérer les lignes d'enregistrement identifiées par les valeurs *("p-1" et "p-2")* de la clé de partition.

3 Etat de l'art

Dans le contexte des entrepôts de données, plusieurs travaux de recherche portent sur le développement des entrepôts de données avec le modèle en colonnes comme l'attestent les différents travaux suivants (Li, 2010), (Dehdouh et al., 2015), (Chevalier et al., 2015) et (Yangui et al., 2016). Ils se résument en 2 grandes catégories (FIG. 3).

La première catégorie des travaux repose sur une dénormalisation complète des données dans le système NoSQL en colonnes cible. L'objectif de ces travaux est de proposer un schéma de stockage qui combine les tables de faits et de dimensions en

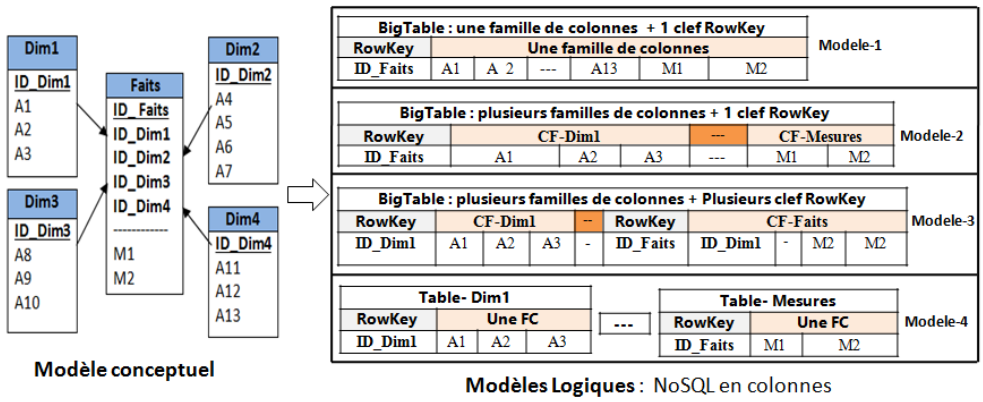

FIG. 3 – *Entrepôt de données - Modèles logiques NoSQL en colonnes*

une seule table. Trois solutions sont couramment appliquées pour construire le modèle loqique de données :

- **Modèle 1 :** tous les attributs des tables de faits et de dimensions sont combinés dans une seule famille de colonnes, les lignes de données sont identifiées par une clé *RowKey* qui correspond à la clé primaire de la table de faits.
- **Modèle 2 :** une famille de colonnes pour chaque table de dimensions ou de faits, et toutes les familles de colonnes sont référencées par une même clé *RowKey* qui correspond à la clé primaire de la table de faits.
- **Modèle 3 :** les attributs de chaque dimension sont regroupés dans une même famille de colonnes et identifiés par la clé primaire de la table de dimension correspondante. Les mesures du fait et les références représentant les dimensions sont regroupées dans une même famille de colonnes référencée par la clé primaire de la table de faits.

La seconde catégorie est une approche normalisée **(Modèle 4)**. Le but est de garder un aspect d'un schéma des données plus proche de la normalisation pour réduire la redondance des données des dimensions. Dans cette approche les faits et les dimensions composant le modèle dimensionnel de l'entrepôt sont stockés séparément sous forme de tables avec une seule famille de colonnes, les lignes de données dans chaque table sont identifiées par la clé primaire de la table de dimension ou de faits correspondants.

D'autres travaux de recherche ont tenté d'améliorer les performances des entrepôts de données NoSQL en colonnes en s'intéressant aux problème d'accès aux données. Par exemple, dans (Romero et al., 2015), pour résoudre le problème de balayage complet des valeurs d'une colonne et réaliser un accès direct aux blocs de données qui satisfont les prédicats de la requête, les auteurs proposent l'utilisation d'index au niveau de la table HBase. De même que les travaux de (Scabora et al., 2016), (Yang et al., 2015) et (Mior et al., 2017) ont essayé d'améliorer le schéma de distribution de données dans un système NoSQL en colonnes. Ils traitent les problèmes de la distribution des attri-

buts entre les familles de colonnes et l'impact du nombre de familles de colonnes sur la performance du schéma physique de données. Ils proposent des techniques pour la conception du schéma de familles de colonnes optimal.

Discussion et positionnement. Nous avons vu dans la littérature que les implémentations des entrepôts de données selon le modèle NoSQL en colonnes se basent généralement sur les méthodes de dénormalisation. Ces travaux reposent essentiellement sur un seul paramètre d'entrée qui est le modèle conceptuel ou le modèle logique relationnel et sur certaines règles de passage des schémas relationnels aux schémas NoSQL en colonnes. D'autre part, nous avons remarqué que les méthodes d'optimisation proposées restent limitées par l'utilisation des index et par la conception du schéma de familles de colonnes. Ces travaux ne tirent pas profit de ce que proposent les systèmes NoSQL en colonnes comme le concept de *RowKey* qui assure la fragmentation horizontale naturelle des données. Par ailleurs, nous avons vu que les propositions du modèle logique BigTable qui est composé de plusieurs familles de colonnes et d'une seule clé RowKey (Figure 3 - modèle 2) s'avèrent plus appropriées pour implémenter des entrepôts de données dans un SGBD NoSQL en colonnes. En s'inspirant de ce modèle logique, notre objectif est de proposer un nouveau modèle logique de données qui généralise la notion de *RowKey* à plusieurs attributs, le but étant de faciliter le regroupement des données appartenant à différentes tables (fait et dimensions) et ayant la même clé de patition *RowKey* sur un même nœud de cluster, de sorte que le traitement des requêtes soit plus optimisé. Dans ce cas, nous proposons une méthode de conception des clés de partition *RowKey*, qui contrairement à la conception naïve, permet la colocalisation des données qui sont souvent et/ou conjointement utilisées sur un même nœud ou groupe de nœuds, dans le but d'optimiser la performance des requêtes. Plus précisément, nous exploitons les attributs utilisés par des prédicats dans une charge de requêtes pour définir une configuration optimale d'une clé de partition *RowKey multi-attributs*. Nous proposons alors une approche basée sur un algorithme de fouille de données, plus précisément la méthode des règles d'association. Ce choix se justifie principalement par le fait que cet algorithme permet de trouver un bon regroupement des données (groupes d'attributs) en utilisant un minimum de ressources (temps d'exécution, complexité des fonctions).

4 Conception de clé de partition *RowKey*

Principe : L'approche de conception des clés de partition que nous proposons est plus adaptée aux entrepôts de données modélisés par un schéma en étoile car elle permet de distribuer équitablement les données sur les différents nœuds, de contrôler le nombre de partitions généré par la table, et de préserver les informations utiles pour les partitions et leurs emplacements à travers les nœuds du cluster.

Notre méthode de conception des clés de partition est subdivisée en trois étapes qui sont détaillées dans les sections suivantes. Pour illustrer nos propos, nous avons simplifié les requêtes décisionnelles (Figure 5) puisque nous nous intéressons qu'aux attributs de la clause WHERE de la requête sachant que de vraies requêtes décisionnelles sont utilisées dans la partie expérimentation (Section 5).

4.1 Extraction des attributs utilisés par des prédicats simples

Cette première étape consiste à traiter l'ensemble des attributs candidats pour la configuration de la clé *RowKey*, ces attributs utilisés par des prédicats simples présents dans la condition de recherche des clauses Where (à l'exception des prédicats de jointure) relatif à la charge de requêtes $Q = \{q_1, ..., q_N\}$. Ainsi, le but de cette extraction est de construire la matrice (QRA : Requête-RowKey-Attributs). Soit $R = \{A_1, A_2, ..., A_M\}$ un ensemble d'attributs candidats pour la création d'une configuration *RowKey*. Dans la matrice QRA, les colonnes représentent les attributs candidats et les lignes représentent les requêtes. La matrice QRA a pour terme général QA_{ij} égal à "1" si l'attribut A_j est impliqué dans la requête q_i et à "0" sinon. La figure 5 montre un exemple de matrice QRA relative à la charge $Q = \{q_1, q_2, q_3, q_4\}$. Notons qu'un prédicat simple noté PS est de la forme : $PS =$ "$A_i\ \beta\ Valeur$", avec $\beta \in \{=, <, >, \leqslant, \geqslant, \neq\}$; $Valeur \in D_i$, D_i est le domaine de définition de l'attribut A_i.

4.2 Sélection d'une configuration de RowKey multi-attributs

La deuxième étape prend en entrée la matrice QRA obtenue à l'étape une. Elle consiste à sélectionner les attributs les plus "intéressants" parmi les attributs candidats pour générer une configuration composée d'une clé de partition *RowKey* multi-attributs. En effet, dans le contexte des entrepôts de données, le nombre d'attributs prédits dans une charge de requêtes peut être important, vu le nombre de tables de dimension et le nombre d'attributs non clés de chaque table. Par conséquent, plusieurs d'attributs sont candidats, ce qui augmente le nombre des configurations possibles d'une *RowKey* multi-attributs. Cette deuxième étape est réalisée à travers l'application de l'algorithme des règles d'association (Agrawal et al., 1993) qui permet de découvrir des corrélations et des relations pertinentes entre attributs binaires extraits à partir d'un ensemble de données réelles d'une base de données. L'algorithme suit les trois phases suivantes :

A) Génération de l'ensemble des combinaisons d'attributs possibles : Cette phase consiste à construire un treillis pour énumérer toutes les combinaisons d'attributs possibles, et de calculer le *support (indicateur de fiabilité de la règle)* associé à chaque combinaison d'attributs du treillis (FIG. 4). L'application de cette phase sur la matrice QRA obtenue dans l'étape précédente permet de créer un ensemble L composé de $k = 7$ combinaisons d'attributs possibles. Chaque combinaison $C_{k(1 \leq k \leq 2^M - 1)} \in L$ est constituée d'un groupe d'attributs $R = \{produit, année, q\_vendue\}$ *(R ensemble de $M = 3$ attributs candidats).* Chaque C_k représente une clé de partition *RowKey multi-attributs* potentiel. Le *support* permet de mesurer pour chaque combinaison d'attributs le nombre de transactions d'apparition simultanée des attributs. Par exemple, le *support* de la combinaison des deux attributs $C = \{produit, année\}$ est égal à "$sup(C) = 2$" *(il ya 2 requêtes contenant les deux attributs à la fois).*

B) Extraction des combinaisons d'attributs fréquents selon les principes de l'algorithme A-priori : Dans la première phase, le nombre de configurations est très élevé, par exemple pour un ensemble de M attributs candidats, le nombre

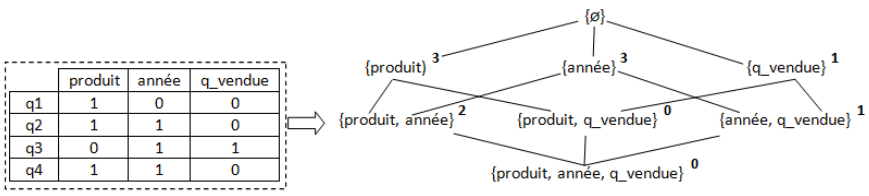

FIG. 4 – *Treillis des combinaisons d'attributs possibles*

total des combinaisons possibles est de $(2^M - 1)$. Il n'est donc pas envisageable de chercher toutes les configurations possibles. Pour réduire la prolifération des règles, nous utilisons le critère du seuil minimal de support défini par l'utilisateur *minsup*. En pratique, nous avons utilisé la méthode A-priori (Agarwal et al., 1994), qui est une implémentation de l'algorithme des règles d'association. Afin de limiter le nombre de sous-ensembles candidats, cette méthode se base sur les principes suivants : *1- Si un ensemble d'éléments est fréquent, alors tous ses sous-ensembles sont fréquents* et *2- Si un ensemble d'éléments contient des combinaisons qui ne répondent pas à la condition du minsup alors cet ensemble est non fréquent*. Par exemple, avec la matrice QRA et le treillis obtenus dans l'étape précédente et un $minsup = 2$, les différentes étapes du déroulement de la méthode A-priori sont :

1.1 : A-priori commence par calculer le support des combinaisons d'attributs de premier niveau du treillis et nous donne l'ensemble des combinaisons de taille 1 : $C1 = (produit)$ avec $sup(C1) = 3$, $C2 = (année)$ avec $sup(C2) = 3$, $C3 = (q\_vendue)$ avec $sup(C3) = 1$. Par la suite, toutes les combinaisons dont la fréquence est inférieure à $minsup = 2$ sont supprimées.
1.2 : Une fois l'étape de suppression terminée, A-priori retourne le nouveau ensemble $L1$ qui contient toutes les combinaisons les plus fréquentes, ici, $L1 = \{C1, C2\}$.
2 : Un second balayage du treillis au niveau 2 est nécessaire pour extraire l'ensemble des combinaisons les plus fréquentes de taille 2. Dans ce cas et selon les principes de l'algorithme A-priori, une seule combinaison possible sera générée, c'est $C4 = (produit, année)$ avec $sup(C4) = 2$. Les combinaisons $(produit, q\_vendue)$ et $(année, q\_vendue)$ seront supprimées à cause de l'attribut q_vendue non fréquent qui le compose. Ceci implique, dans cette itération le nouvel ensemble $L2 = \{C4\}$ qui contient une seule combinaison de 2 attributs. Il n'est alors pas nécessaire de balayer le troisième niveau du treillis ; ce qui met fin à l'exécution de la méthode A-priori car la génération des sous-ensembles de taille 3 d'attributs va donner un ensemble vide.
3 : L'algorithme retourne finalement l'ensemble des combinaisons d'attributs : $C1 = (produit)$, $C2 = (année)$ et $C4 = (produit, année)$, relatives à $L1$ et $L2$.

C) Filtrage de l'ensemble des listes d'attributs fréquents : La phase B permet de retrouver toutes les combinaisons d'attributs les plus fréquents. Cependant elle peut générer plusieurs combinaisons d'attributs qui ont des supports identiques

et supérieurs au *minsup*. Pour remédier à ce problème, nous avons utilisé deux paramètres de mesure supplémentaires proposés dans la méthode des règles d'association : le critère de *min.confiance (indicateur de précision des règles d'association)* et le critère $Card(C)$ qui représente le nombre d'attributs dans les combinaisons fréquentes. Pour illustrer cette étape de filtrage, revenons à notre exemple. A-priori a généré en tout 3 combinaisons d'attributs fréquents : 2 combinaisons de taille 1 et une combinaison de taille 2. Supposons que $conf.min = 60\%$ et $Card(C) = 2$, les combinaisons $C1 = (produit)$ et $C2 = (année)$ seront supprimées et on ne conserve que la combinaison $C4 = (produit, année)$ de cardinal 2. Cependant, cette combinaison $C4 = (produit, année)$ permet de créer deux configurations de la clé de partition *RowKey* qui sont "*produit|année*" et "*année|produit*". En pratique l'ordre d'apparition des deux attributs est très important dans la configuration de la clé *RowKey*, il s'agit donc de trouver le meilleur ordre des attributs. Pour cela, nous exploitons le critère de la confiance pour donner une priorité aux attributs. Pour calculer la confiance de la combinaison $C4$ cela nécessite de calculer les 2 possibilités d'occurrences en termes probabilistes.

$$conf(produit, année) = \frac{sup(produit, année)}{sup(produit)} = 2/3 = 66\% > conf.min$$

$$conf(année, produit) = \frac{sup(année, produit)}{sup(année)} = 2/3 = 66\% > conf.min$$

Dans cet exemple, les 2 possibilités de configurations sont valides avec une confiance de 66% pour une configuration finale de la clé *RowKey*. On peut dans ce cas choisir l'une ou l'autre des configurations pour constituer la clé de partition *RowKey*.

4.3 Préparation des données

Dans la section 3, nous avons présenté les différents modèles logiques d'entrepôt de données NoSQL en colonnes. Nous avons constaté que le modèle *"BigTable- plusieurs familles de colonnes + 1 seule clé RowKey"* s'avère le plus adapté à l'entreposage des données. Pour illustrer la phase de préparation des données, nous prenons comme exemple un entrepôt de données avec un schéma en étoile contenant 4 tables fait_ventes, dim_client, dim_produit et dim_temps (FIG. 5). Dans un système NoSQL en colonnes, toutes les données de l'entrepôt peuvent être stockées dans une grande table agrégée avec 4 familles de colonnes, chaque table de dimension est convertie en famille de colonnes plus une famille de colonnes qui contient les mesures. Toutes les familles de colonnes partagent une même clé de partition *RowKey multi-attributs* définie par la concaténation des valeurs des 2 attributs *année et produit suivis par un numéro d'ordre*. Ainsi, chaque ligne dans cette table est constituée par un ensemble de familles de colonnes issues du fait (imbrication des mesures) et de chaque dimension associée (imbrication des attributs de la dimension), de plus, elle est identifiée par une seule valeur de clé de partition *RowKey multi-attributs*. Par exemple, si la valeur de la clé est égale à $< 2010p1 - 1 >$ alors cette valeur de la clé sert à identifier la ligne suivante : en jour $d1$ de l'année 2010, le client $C5$ achète 6 unités du produit $p1$ pour

un prix de 10 avec montant total de 60. Donc toutes ces données, faisant référence à la valeur $< 2010p1 - 1 >$ de la clé, sont stockées ensemble sur même nœud.

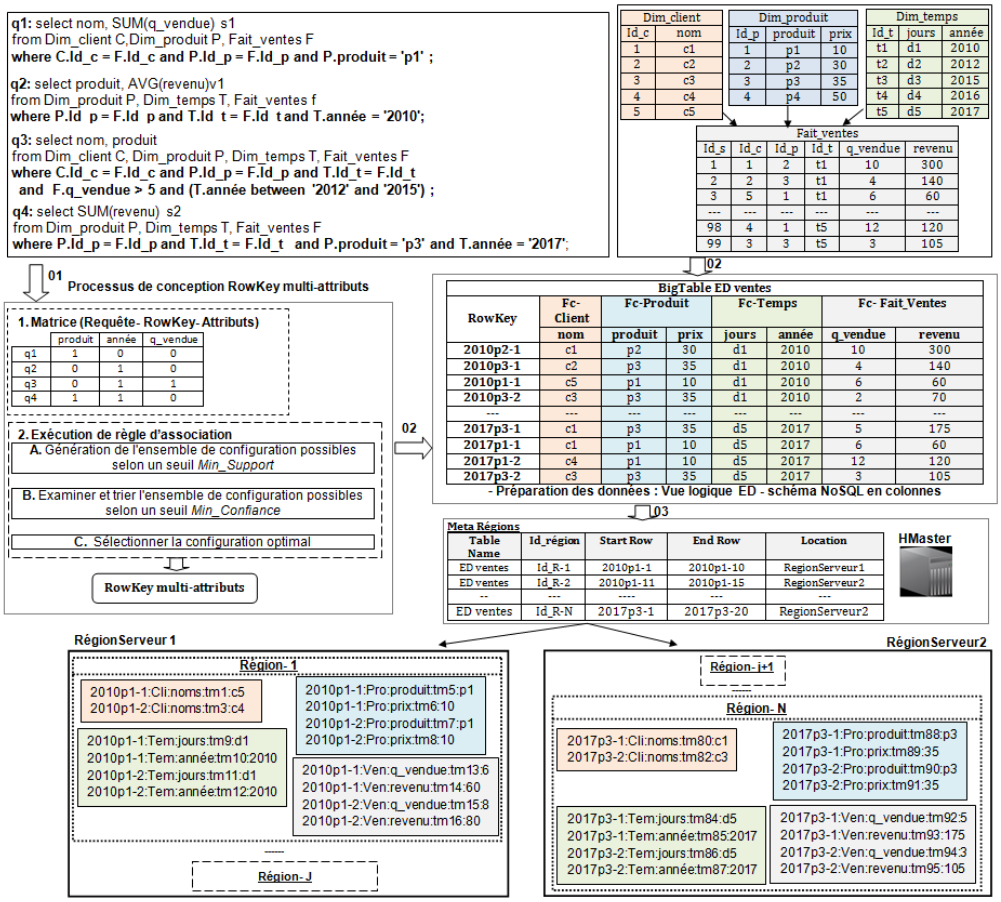

FIG. 5 – *RowKey multi-attributs et partitionnement des données sous HBase*

5 Implémentation, expérimentations et résultats

1. Entrepôt de données : Pour mener à bien l'évaluation de notre approche *Balanced-CN-DW*, nous avons utilisé le banc d'essai TPC-DS [4]. TPC-DS utilise un schéma en constellation, composé de 17 tables de dimensions et 7 tables de faits. Dans notre cas, nous utilisons la table des faits STORE_SALES et ses 9 tables de dimensions. Le générateur de données DSDGEN de TPC-DS permet de générer des fichiers de données dans un format *.dat* avec différentes tailles selon un facteur d'échelle (*SF : Scale*

4. Benchmark (TPC-DS) v2.0.0, http ://www.tpc.org/tpcds/.

Factor), chaque fichier correspond à une table de faits ou une table de dimension. Nous avons fixé SF à 100 pour produire 100 Go de données, ce qui produit dans la table de faits STORE_SALES 287.997.024 tuples.

2. Charge de de requêtes : Le benchmark TPC-DS propose 99 requêtes. Nous avons sélectionné une charge composée de 19 requêtes distinctes qui exploitent la table de faits STORE_SALES et ses dimensions. Elles sont organisées selon : (1) le nombre des tables (de faits et dimensions) parcourues par les requêtes et (2) le nombre des prédicats de sélection définis sur les différents attributs des tables de faits et de dimensions. En raison de certaines exigences qui ne sont pas réalisables avec Apache Phoenix [5] (sur les capacités de lecture de requêtes) et le SGBD HBase, ces requêtes nécessitent quelques modifications et changements de syntaxe.

	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18	q19
Tables	1		2	3							4		5						6
Attributs	4	9	9	4	4	6	6	6	6	9	6	7	10	10	10	11	11	12	14
Prédicats	4	3	10	4	5	12	7	7	6	6	5	7	22	13	10	8	23	15	15

TAB. 1 – *Caractéristiques de la charge de requêtes*

3. Configuration expérimentale : Pour mener nos expérimentations, nous avons mis en place un environnement de stockage NoSQL distribué. C'est un cluster d'ordinateurs composé d'un serveur maître (*NameNode*) et de trois machines esclaves (*DataNodes*). Le *NameNode* est équipé d'un processeur Intel-Core TMi5-3550 CPU@3.30 GHZx4 avec une mémoire RAM de 16 Go et de 1 TB d'espace disque. Les *DataNodes* sont tous équipés d'un processeur Intel-Core TMi5-3550 processor CPU@3.30 GHZx4, de 16 Go de RAM, et de 500 Go d'espace disque. Ces machines fonctionnent sous Ubuntu-14.04 LTS de 64 bits et la version Java JDK 8. Nous avons utilisé Hadoop (v 2.6.0), HBase (v 0.98.8), MapReduce pour le traitement, Zookeper, *Phoenix* (v4.6.0) et SQuirreL pour interroger les données et augmenter les performances de HBase.

4. Tests : Nous avons choisi 2 méthodes pour implémenter le banc d'essai *TPC-DS* dans le système HBase :

1. La table de faits STORE_SALES et ses 9 tables de dimension sont stockées dans une BigTable avec 10 familles de colonnes, chacune des tables de faits et de dimensions correspond à une famille de colonnes. Chaque ligne de données est identifiée par une clé *RowKey*, qui correspond à l'ordre séquentiel croissant des tuples de la table de faits STORE_SALES qui contient *287 997 024* faits. Nous baptisons cette méthode *Naïve-CN-DW*.
2. la table de faits STORE_SALES et ses 9 tables de dimension sont stockées dans une BigTable avec 10 familles de colonnes, chacune des tables de faits et de dimensions correspond à une famille de colonnes. Toutes les familles de colonnes sont référencées par une même clé *RowKey multi-attributs*, obtenue selon notre

5. https ://phoenix.apache.org/

méthode. Cette méthode est baptisée *Balanced-CN-DW*. La méthode des règles d'association a généré une configuration de la clé de partition *RowKey* de taille 4 attributs dont l'ordre d'apparition est *d_year, d_moy, cd_marital_status, cd_education_status*, autrement dit, dans ce modèle d'entrepôt *Balanced-CN-DW*, la clé *RowKey* est définie par la concaténation des valeurs des 4 attributs *d_year* / *d_moy* / *cd_marital_status* / *cd_education_status suivie par un numéro d'ordre.*

3. Les 19 requêtes sélectionnées ont été exécutées sur les différents modèles de l'entrepôt NoSQL issus des deux configurations décrites ci-dessus, à savoir *Naïve-CN-DW* et *Balanced-CN-DW*.

5. Résultats et discussion :

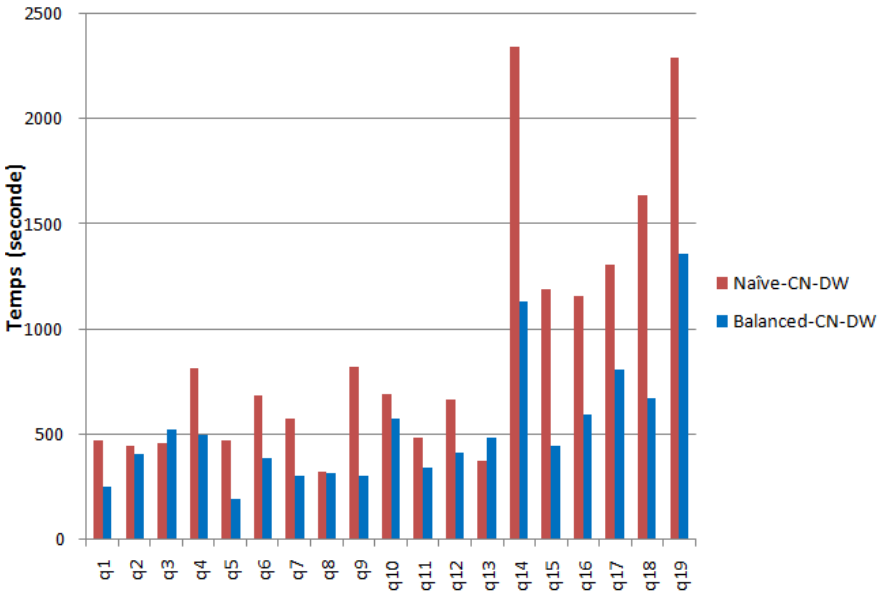

FIG. 6 – *Temps de réponse des requêtes*

Notre stratégie de placement des données d'un entrepôt sur un cluster au sein du SGBD HBase montre clairement l'apport apporté en termes de gain sur le temps d'exécution des requêtes décisionnelles (Figure 6). En effet, la colocalisation des blocs de données qui sont souvent sollicités par les requêtes fréquentes permet de réduire le nombre de partitions des données lues pour chaque requête en évitant ainsi le balayage séquentiel de la table. En particulier, les temps de réponse des requêtes les plus complexes ($q14$, $q15$, $q18$ et $q19$) qui sollicitent un nombre très important des familles de colonnes sont améliorées de 48%. Par ailleurs, en capturant la quantité de données transférées sur le réseau, nous avons prouvé que la colocalisation permet effectivement

la réduction de la quantité des données transférées lors de l'exécution des tâches *map* et *reduce* sur chaque région (une partition horizontale) lors du lancement d'une opération de lecture de HBase. Ceci permet entre autres de réduire la surcharge du réseau nécessaire lorsque les systèmes distribués possèdent une bande passante faible ou limitée.

6 Conclusion

Dans cet article, nous avons présenté une approche de partitionnement et de distribution d'entrepôts de données NoSQL en colonnes performants baptisé *Balanced-CN-DW* dans un environnement distribué. Nous avons procédé à des regroupements d'attributs pour définir les clés de partition *RowKey* en utilisant la méthode des règles d'association afin de répartir les données partageant une même valeur de Rowkey dans un même nœud d'un cluster pour optimiser le traitement des requêtes décisionnelles. Nous avons mené plusieurs expérimentations dont les résultats confirment l'intérêt de l'application des techniques de classification des données pour créer un schéma de partitionnement horizontal des données au sein du SGBD NoSQL en colonnes HBase en tenant compte de la charge de requêtes. L'entrepôt *Balanced-CN-DW* améliore les performances des requêtes de l'ordre de 48% si on le compare avec l'approche naïve proposée par HBase. Nos travaux ouvrent plusieurs perspectives de recherche. A plus court terme nous comptons étudier les effets d'autres paramètres de configuration liés à l'environnement d'entreposage sur notre méthode de distribution des données tels que le changement de la charge de requêtes, la fréquence d'usage des requêtes, la taille des données, le nombre de nœuds dans un cluster ou le facteur de réplication de données. A moyen terme, nous souhaitons comparer notre approche avec d'autres techniques d'optimisation de requêtes dans un environnement distribué telles que la réplication des données par exemple.

Références

Agarwal, R., R. Srikant, et al. (1994). Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pp. 487–499.

Agrawal, R., T. Imieliński, et A. Swami (1993). Mining association rules between sets of items in large databases. In *Acm sigmod record*, Volume 22, pp. 207–216. ACM.

Chevalier, M., M. El Malki, A. Kopliku, O. Teste, et R. Tournier (2015). Implementation of multidimensional databases in column-oriented nosql systems. In *East European Conference on Advances in Databases and Information Systems*, pp. 79–91. Springer.

Dean, J., S. Ghemawat, S. Dill, R. Kumar, K. McCurley, S. Rajagopalan, et D. Sivakumar (2001). Mapreduce : Simplified data processing on large clusters, osdi'04 : Sixth symposium on operating system design and implementation, san francisco, ca, december, 2004. *S. Dill, R. Kumar, K. McCurley, S. Rajagopalan, D. Sivakumar, ad A. Tomkins, Self-similarity in the Web, Proc VLDB*.

Dehdouh, K., F. Bentayeb, O. Boussaid, et N. Kabachi (2015). Using the column oriented nosql model for implementing big data warehouses. In *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 469–475.

Li, C. (2010). Transforming relational database into hbase : A case study. In *2010 IEEE Int. Conf. on Software Engineering and Service Sciences*, pp. 683–687. IEEE.

Mior, M. J., K. Salem, A. Aboulnaga, et R. Liu (2017). Nose : Schema design for nosql applications. *IEEE Transactions on Knowledge and Data Engineering 29*(10), 2275–2289.

Romero, O., V. Herrero, A. Abelló, et J. Ferrarons (2015). Tuning small analytics on big data : Data partitioning and secondary indexes in the hadoop ecosystem. *Information Systems 54*, 336–356.

Scabora, L. C., J. J. Brito, R. R. Ciferri, C. D. d. A. Ciferri, et al. (2016). Physical data warehouse design on nosql databases olap query processing over hbase. In *Inter. Conf. on Enterprise Information Systems, XVIII*, pp. 111–118. Inst. for Systems and Technologies of Information, Control and Communication-INSTICC.

Stonebraker, M., D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al. (2005). C-store : a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pp. 553–564. VLDB Endowment.

Yang, F., D. Milosevic, et J. Cao (2015). An evolutionary algorithm for column family schema optimization in hbase. In *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*, pp. 439–445. IEEE.

Yangui, R., A. Nabli, et F. Gargouri (2016). Automatic transformation of data warehouse schema to nosql data base : Comparative study. *Procedia Computer Science 96*, 255–264.

Summary

The column family NoSQL databases offer several storage techniques that are well adapted to data warehouses. Several scenarios are possible to develop column NoSQL data warehouses. In this paper, we propose a new method to build an efficient distributed data warehouse inside column family NoSQL DBMSs. Our method, named *Balanced-CN-DW*, is based on the association rules method that allows to obtain groups of frequently used attributes in the workload. Hence, the partition keys *RowKey*, necessary to distribute data onto the different cluster nodes, are composed of those attributes groupes. To evaluate our method, we use the TPC-DS benchmark within the NoSQL HBase DBMS and carry out several experiments by executing TPC-DS decision queries. The obtained results show that our data placement and distribution strategy increases the performance of our column NoSQL data warehouse model.