Optimisation des performances dans les entrepôts de données NoSQL en colonnes

Mohamed Boussahoua*, Omar Boussaid*, Fadila Bentayeb *

*Université de Lyon, Université Lyon 2, ERIC EA 3083
5 avenue Pierre Mendes-France, F-69676 Bron Cedex, France
{mohamed.boussahoua, omar.boussaid, fadila.bentayeb}@univ-lyon2.fr

Résumé. Le modèle NoSQL orienté colonnes propose un schéma de données flexible et hautement dénormalisé. Dans cet article, nous proposons une méthode d'implantation d'un entrepôt de données dans un système NoSQL en colonnes. Notre méthode est basée sur une stratégie de regroupement des attributs issus des tables de faits et de dimensions, sous forme de familles de colonnes. Nous utilisons deux algorithmes *OEP* et *k-means*. Pour évaluer notre méthode, nous avons effectué plusieurs tests sur le benchmark TPC-DS au sein du SGBD NoSQL orienté colonnes *Hbase*, avec une architecture de type *MapReduce* sur une plateforme *Hadoop*.

1 Introduction

Les entrepôts de données jouent un rôle important dans la collecte et l'analyse de grandes masses de données pour l'aide à la décision. Généralement, ils sont souvent implémentés sous les systèmes de gestion de bases de données relationnelles (SGBDR). Ces derniers s'imposent par la richesse de leurs fonctionnalités et les performances de leurs requêtes. Cependant, ils sont peu appropriés pour construire des entrepôts de données distribuées, nécessaires pour faire face à l'augmentation du volume de données et à la scalabilité de l'espace de stockage Leavitt (2010). De plus, l'exécution des requêtes décisionnelles dégrade les performances des entrepôts de données dans un SGBDR. Pour améliorer les performances des entrepôts de données relationnels, différents travaux de recherche existent et portent notamment sur les techniques d'indexation, la fragmentation ou la compression des données. De la même manière, il est nécessaire de recourir à de nouvelles solutions de stockage fiables et à moindre coût dans les systèmes décisionnels distribués. Parmi ces solutions, il y a lieu de citer la plateforme Hadoop [1], qui comprend différents modules, tels que Apache Hive [2], un système d'entreposage de données muni d'une interface de type SQL, Apache Pig et de nouveaux modèles de données dits NoSQL (*Not Only SQL*) [3] apparus ces dernières années sous l'impulsion des grands acteurs du Web (*Google*, *Yahoo*, *Facebook*, *Twitter*, *Amazon*...). Ces SGBD, dits aussi non relationnels, s'appuient sur le théorème de CAP (*Consistency,*

1. http ://hadoop.apache.org
2. https ://hive.apache.org
3. https ://fr.wikipedia.org/wiki/NoSQL

Availability, *Partition Tolerence*) Brewer (2000). Ils offrent une grande flexibilité de représentation et de gestion de gros volumes de données sur des serveurs de stockage distribués.

Nos travaux se focalisent sur la modélisation et l'implémentation d'un entrepôt de données selon le modèle NoSQL orienté colonnes. Dans les entrepôts relationnels, la construction d'un cube OLAP nécessite l'accès aux attributs des tables des faits et des dimensions. Par conséquent, pour sélectionner une colonne (axe d'analyse), il faut charger toute la table de dimension concernée. Contrairement aux systèmes relationnels, dans le modèle NoSQL en colonnes, il est possible d'accéder uniquement à la colonne souhaitée puis de la charger en mémoire, ceci facilite l'application d'une fonction d'agrégation.

Dans cet article, nous nous intéressons aux techniques de regroupement des attributs pour la constitution de familles de colonnes. Notre objectif est d'obtenir des ensembles d'attributs homogènes permettant d'accroître les performances des requêtes décisionnelles. Nous proposons alors une stratégie de regroupement d'attributs, à partir d'une charge initiale de requêtes, en utilisant deux algorithmes : une méta-heuristique, en l'occurrence l'*Optimisation par Essaim Particulaire (OEP)*, et l'algorithme de fouille, *k-means*.

Nous avons étudié le gain en performance de requêtes décisionnelles en comparant les résultats obtenus et en exécutant celles-ci sur un entrepôt construit selon notre approche, puis sur d'autres entrepôts construits selon deux autres méthodes de regroupement : (1) l'ensemble des attributs des tables de faits et des dimensions est mis dans une même famille de colonnes ; (2) chacune des tables de faits et des dimensions correspond à une famille de colonnes.

Pour réaliser nos expérimentations, nous avons utilisé le banc d'essai TPC-DS [4] (*Transaction Processing Council Decision Support*) conçu pour mesurer les performances des entrepôts de données relationnels. De plus, nous avons étendu la charge de requêtes initiale de TPC-DS afin d'obtenir un schéma de regroupements d'attributs plus significatif. Nos tests montrent que nos méthodes de regroupement OEP et K-means améliorent de façon significative le temps d'exécution des requêtes décisionnelles dans l'entrepôt CN-DW (Column NoSQL Data Warehouse) pour un nombre de familles de colonnes compris entre 7 et 12. Ces résultats sont également valables en comparaison avec les deux autres méthodes.

Ce papier est organisé comme suit. La Section 2 présente un état de l'art des travaux portant sur le développement des entrepôts de données selon le modèle NoSQL orienté colonnes. La Section 3 détaille la problématique liée à l'accès aux données dans un entrepôt basé sur un systéme NoSQL en colonnes. Dans la Section 4, nous détaillons l'approche proposée. La Section 5 présente l'évaluation de notre approche. Enfin, nous concluons cet article et présentons quelques perspectives dans la section 6.

2 Etat de l'art

Les bases de données NoSQL en colonnes sont caractérisées par leur schéma de données libre et leur modèle logique favorisant la dénormalisation, et dont la définition

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

reste tout de même une tâche délicate. Plusieurs travaux de recherche y sont consacrés.

Dans Li (2010), les auteurs proposent une approche pour transformer un schéma d'une base de données relationnelle en un schéma d'une base NoSQL orientée colonnes via *HBase*, où le schéma relationnel est restructuré en une grande table où chaque table relationnelle devient une famille de colonnes.

Les auteurs de Abelló et al. (2011), présentent trois méthodes pour construire un cube OLAP, à partir d'un entrepôt implémenté dans *HBase*. Les auteurs ont recouru à la dénormalisation du schéma de l'entrepôt, et ont défini une famille de colonnes pour chaque attribut. Dans la première méthode, ils proposent d'utiliser le paradigme *MapReduce* de manière naïve et sans aucune optimisation, et de bénéficier de la puissance de *HBase* pour parcourir les données et construire le cube. L'inconvénient est la taille importante des données manipulées lors de l'interrogation. Pour résoudre ce problème, les auteurs proposent d'utiliser, dans une seconde méthode, des indexes au niveau des valeurs d'une colonne. L'inconvénient est le nombre important de blocs de données parcourus. Ils sont amenés ainsi à utiliser, dans une troisième méthode, l'index bitmap comme index supplémentaire pour réduire davantage le nombre de valeurs parcourues et réaliser un accès direct aux blocs de données.

Les travaux de Dehdouh et al. (2014) proposent des méthodes d'implantation des cubes OLAP dans le modèle NoSQL en colonnes. Les auteurs ont développé un banc d'essai décisionnel en NoSQL colonnes (*CNSSB : Columnar NoSQL Star Schema Benchmark*) basé sur SSB (*Star Schema Benchmark*). Pour représenter les tables de faits et des dimensions dans un système NoSQL en colonnes, les auteurs ont proposé, dans Dehdouh et al. (2015), trois approches à savoir : NLA (*Normalized Logical Approach*) où les tables de faits et des dimensions sont stockées séparément sur différentes tables ; DLA (*Denormalized Logical Approach*), ce processus favorise la dénormalisation du schéma conceptuel dimensionnel et regroupe les faits et les dimensions dans une table unique appelée *BigFactTable*, où chaque famille de colonnes est composée d'un seul attribut ; DLA-CF (*Denormalized Logical Approach by using Column Family*) permet d'encapsuler les tables de faits et des dimensions dans une même table, où chacune devient une famille de colonnes.

Dans les traveaux de Chevalier et al. (2015), les auteurs ont présenté une approche basée sur des régles de transformation d'un modèle conceptuel multidimensionnel en un modèle logique NoSQL orienté colonnes ou orienté documents. Ils proposent alors trois modèles pour implémenter un entrepôt de donnée dans *HBase*. MLC0 (*simple flat model*) : les attributs des tables des faits et des dimensions sont combinés dans une même table en une seule famille de colonnes ; MLC1 : l'avantage de ce modèle est de regrouper, au sein d'une même table, les attributs des faits dans une famille de colonnes, et ceux de chaque dimension dans une famille de colonnes distincte ; MLC2 (*shattered model*) : chacune des tables des faits et des dimensions correspondrait à une table avec une seule famille de colonnes.

Dans Scabora et al. (2016), les auteurs s'orientent vers un modèle de données NoSQL orienté colonnes pour résoudre le problème de distribution des attributs entre les familles de colonnes afin d'optimiser les requêtes et faciliter la gestion des données. Pour ce faire, ils ont implémenté les données de l'entrepôts dans une table *HBase* composée de deux familles de colonnes. La première réunit les attributs des faits et des dimensions

fréquemment interrogés. La deuxième regroupe les attributs des autres dimensions. Cependant, les auteurs n'utilisent pas des techniques de regroupement.

3 Définition du problème

L'implémentation d'un entrepôt de données dans un système NoSQL en colonnes prend en compte les spécificités de l'environnement du stockage physique des données. Rappelons que ces dernières sont organisées en familles de colonnes composées d'un ensemble d'attributs. Ainsi, il s'agit d'un principe de partitionnement vertical des données. La figure 1 montre un exemple représentant une table T avec 7 attributs et 3 familles de colonnes. Chaque famille CF_i consiste en un ensemble d'attributs ayant chacun une valeur, chaque ligne des données est référencée par une clé de ligne Ri.

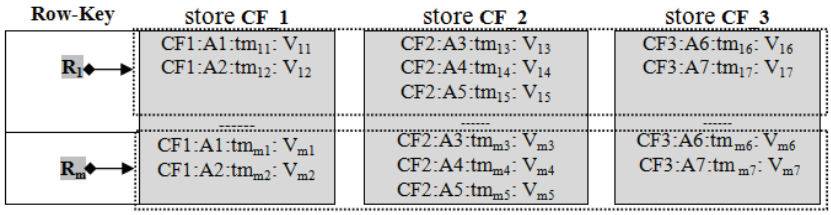

FIG. 1 – *Stockage orienté colonnes d'une table*

En réalité d'un point de vue stockage (figure 1), toutes les données, faisant référence à une même clé de ligne (*Rowkey*), sont stockées ensemble, le nom de la famille de colonnes agissant alors comme clé de chacune de ses colonnes et la clé de la ligne comme clé de l'ensemble des attributs d'un même « *enregistrement* » au sens relationnel du terme. On peut remarquer dans cette technique de stockage, que deux éléments influent sur la vitesse d'exécution des requêtes : le choix des colonnes et celui des lignes. Pour une requête portant, par exemple, sur les valeurs d'attributs $\{CF\_1 : A1, CF\_2 : A4, CF\_3 : A6\}$, la table est parcourue au niveau des familles de colonnes 3 fois ; la recherche se fera sur 3 partitions de données différentes : CF_1 , CF_2 et CF_3 avec un balayage complet des valeurs de l'attribut concerné. Ceci nécessite alors un temps d'exécution élevé pour atteindre les valeurs. Cependant, l'implémentation des entrepôts de données, selon un modèle NoSQL en colonnes, se base sur les méthodes de dénormalisation, qui regroupe à la fois les données des faits et celles des dimensions dans une même table. Ceci impose la duplication des données des dimensions pour chaque instance de fait, ce qui minimise l'avantage du regroupement d'attributs en familles de colonnes. De ce fait, pour éviter l'accès à plusieurs familles de colonnes (plusieurs méta-données) et un balayage complet de toutes les données, il est important de proposer un regroupement des données qui soit plutôt approprié aux besoins des utilisateurs. De plus, les implémentations NoSQL habituelles s'exécutent entièrement *in memory*, générant ainsi un coût. Celui-ci sera particulièrement élevé selon l'importance du volume des données. De ce fait, le problème qui se pose lors de la construction de la

table est de savoir comment définir le bon nombre (seuil) de familles de colonnes pour une bonne stratégie de regroupement des attributs.

Nous discuterons, dans la section suivante, l'approche proposée pour l'implantation des entrepôts de données selon le modèle NoSQL orienté colonnes.

4 Regroupement des colonnes en familles

4.1 Principe

Notre méthode d'implantation d'un entrepôt relationnel (schéma en étoile ou en flocon de neige) selon un modèle logique NoSQL orienté colonnes, s'appuie sur le processus suivant :

1. extraction des attributs des tables des faits et des dimensions ;
2. regroupement des attributs et construction des familles de colonnes selon *OEP* ou *k-means* ;
3. génération d'un schéma (métadonnées) de l'entrepôt de données dans le modèle NoSQL en colonnes en fonction des regroupements obtenus ;
4. préparation des données et chargement de l'entrepôt.

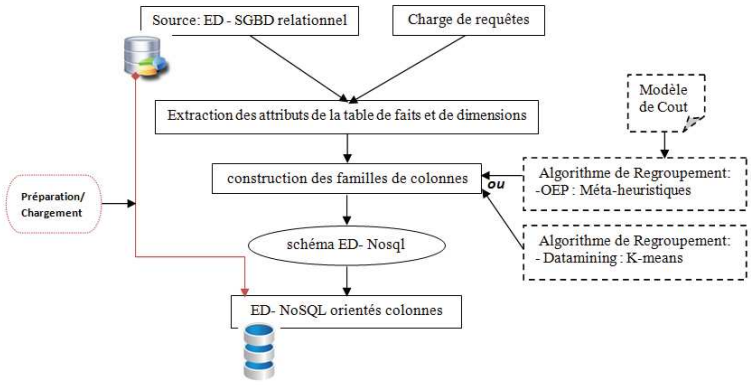

FIG. 2 – *Processus semi-automatique pour l'implémentation d'un entrepôt selon le modèle NoSQL en colonnes*

4.2 Formalisation

Un entrepôt de données est défini par $D = \{d_1, d_2, ..., d_j\}$ tables de dimension et une table de faits F de n attributs. Chaque table de dimension $(d_m)_{m=1,...,j}$ est composée de plusieurs attributs A^i_m avec $i \in [1, k]$, tel que $d_m = \{A^1_m, A^2_m, ..., A^k_m\}$ où k peut varier d'une dimension à l'autre. Nous considérons également un ensemble des requêtes $Q = \{q_1, q_2, ..., q_q\}$. Celles-ci exploitent la totalité du schéma de l'entrepôt

par des opérations de sélection, de jointure et d'agrégation ($MAX, COUNT, SUM...$). Chaque requête q_i contient un ensemble d'attributs.

Soit $F = \{FC_1, FC_2, ..., FC_w\}$ l'ensemble des familles de colonnes qui sera généré. Le nombre de familles de colonnes (w) est défini tel que : $2 \leq w \leq W$. Pour contrôler le nombre maximum de familles de colonnes à créer, un seuil W est fixé à priori.

4.3 Extraction des attributs des tables de faits et de dimensions

Notre technique d'implantation d'un entrepôt selon un modèle NoSQL en colonnes est un processus semi-automatique (figure 2). Il est basé sur une analyse statistique des requêtes les plus fréquentes et s'appuie sur des informations tant qualitatives que quantitatives de l'utilisation des données. Celles-ci portent sur les relations, l'ensemble d'attributs et les tuples accédés, les prédicats portant sur les attributs, le type des requêtes, le nombre de requêtes de lecture, la fréquence d'exécution d'une requête, le site à partir duquel la requête est exécutée, la capacité de stockage et le coût de transfert des données entre sites. Cette première étape consiste alors à traiter l'ensemble des attributs (*présents dans les clauses Select et Where*) relatifs à la charge de requêtes, pour construire la matrice d'usage des attributs (MUA) et la matrice d'affinités (MAA). Pour ce faire, nous nous sommes inspirés des travaux de Navathe et al. (1984). Ces derniers utilisent le principe des affinités entre attributs pour concevoir des groupes d'attributs.

4.4 Construction des familles de colonnes

Dans cette étape, notre objectif est de définir un schéma logique NoSQL de l'entrepôt qui optimise au mieux l'accès aux données pour les requêtes. Notre solution consiste à mettre en œuvre un processus visant le regroupement des attributs qui sont fréquemment interrogés ensemble. Ce regroupement permettra de former des familles de colonnes composant le schéma logique de l'entrepôt NoSQL. Nous avons choisi pour cela d'utiliser deux types d'algorithmes : (1) une méta-heuristique *OEP* développée par Eberhart et al. (1995) ; (2) et l'algorithme *k-means* (MacQueen et al., 1967). Notre choix d'utiliser ces deux algorithmes *OEP* et *k-means* est motivé par le fait qu'on puisse contrôler le nombre de classes (familles de colonnes) dans *k-means* ; cela s'avère être un avantage, du moment que nous voulons limiter le nombre des familles de colonnes. De plus, *EOP* offre la possibilité de contrôler le nombre d'attributs par groupe (famille de colonnes), et d'avoir en plus le même avantage que les *k-means*. Cela nous aide à construire des familles de colonnes avec le même nombre d'attributs pour équilibrer la charge.

4.4.1 Optimisation par Essaim Particulaires (OEP)

Cet algorithme est inspiré des essaims d'insectes ou d'animaux et de leurs déplacements en groupes pour trouver de la nourriture. Au départ l'essaim est réparti au hasard dans l'espace, chaque particule ayant une vitesse aléatoire. Ensuite, les particules se déplacent dans l'espace de recherche en se basant sur des informations limitées, c'est-à-dire chaque particule doit décider de son prochain mouvement et sa nouvelle

vitesse. Pour ce faire, elle combine linéairement trois informations : (1) sa vitesse actuelle ; (2) sa meilleure performance : chaque particule est capable d'évaluer la qualité de sa position et de garder en mémoire sa meilleure performance, c'est-à-dire le meilleur point par lequel elle est déjà passée ; (3) la meilleure performance de ses voisines (ses informatrices) : chaque particule peut interroger ses plus proches voisines pour connaître leurs meilleures performances afin de décider de son déplacement. Les algorithmes à essaim de particules peuvent s'appliquer aussi bien à des données discrètes que continues.

Adaptation de l'algorithme OEP pour la regroupement des Attributs
Notre problème est défini par :
— $R = \{A_1, A_2, .., A_{nb}\}$: ensemble des attributs
— Q : nombre total des requêtes fréquentes
— fq : fréquence d'accès de la requête q, pour $q = 1, 2, .., Q$
— W : nombre maximum des familles de colonnes $(2 \leq W)$
— P : nombre d'attributs par famille de colonnes $(1 \leq P < nb)$
— C : ensemble de contraintes physiques *(nombre de nœuds du cluster, capacité de stockage des nœuds, espace mémoire utilisé par les machines, taille de l'entrepôt...)*
— $S = \{S_1, S_2, ..., S_t\}$: ensemble de toutes les solutions réalisables, tel que t est le nombre maximum d'itérations de l'OEP, $(S_i)_{i=1,..,t} = \{FCi_1, FCi_2, .., FCi_w\}$, où les $(FCi_j)_{j=1,..,W}$ sont des sous-ensembles finis de P attributs de R ;
— F : une fonction objectif qui prend ses valeurs sur S.

Le problème consiste à trouver une solution $S^* \in S$ optimisant la valeur de la fonction objective F tel que : $F(S^*) \leq F(S_i)$, pour tout élément $S_i \in S$, et F la fonction objectif à minimiser.

La fonction objectif F permet de mesurer la qualité des solutions $(S_i)_{i=1,..,t}$ obtenues après chaque itération d'*OEP*. Notre fonction de coût s'inspire des travaux de Derrar et al. (2008), qui consistent à évaluer, lors de la phase de conception, la pertinence des schémas de partitionnement générés. Initialement, cette fonction est calculée en utilisant l'erreur au carrée (*Square-Error*) et en tenant compte des fréquences d'accès des requêtes aux attributs des différents groupes. L'erreur au carrée du schéma global de partitionnement de la relation R se calcule comme suit :

$$E_W^2 = \sum_{j=1}^{W} \sum_{q=1}^{Q} [(f_q)^2 \times s_j^q (1 - \frac{(s_j^q)}{P})]$$

Avec s_j^q le nombre d'attributs du groupe FCi_j sur un schéma de partitionnement S_i accédé par la requête q.

Afin de trouver la solution optimale S^*, nous utilisons l'algorithme *OEP-FC* qui exploite la fonction objectif F. Cette fonction évalue, pour chaque itération i de l'algorithme, l'erreur carrée E_W^2 du schéma S_i généré lors d'une itération. Il s'agit donc de déterminer un schéma S^* qui minimise la valeur de E_W^2, sous l'ensemble des contraintes C, cette fonction est définie comme suit :

$$F(S^*) = Min(E_W^2, S_i)_{i=1,..,t}$$

4.4.2 Application de *k-means* pour le regroupement d'attributs

Cet algorithme prend en entrée un ensemble de points et un entier k ; le problème consiste à diviser les points en k groupes en minimisant la somme des carrés des distances entre les points d'un groupe et son centre. Les points sont alors les attributs et leurs distances sont les affinités. *k-means* prend en entrée la matrice des affinités d'attributs (MAA) et le nombre de clusters k, et retourne les familles de colonnes.

5 Implémentation et expérimentations

5.1 Protocole expérimental

Nous avons développé un outil nommé TA-EDRC (Transformation Automatique d'un Entrepôt de Données Relationnel en NoSQL orienté Colonnes), avec le langage de programmation Java, qui implémente les méthodes *OEP-FC* et *k-means*.

Entrepôt de données : Nous utilisons le benchmark TPC-DS, implémenté sous PostgreSQL. Celui-ci sert aux tests de performance ; il modélise plusieurs aspects des opérations commerciales dans lesquels les données correspondent aux trois modes de vente : en magasin, par catalogue et par Internet. 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 9 tables de dimensions. Le générateur de données DSDGEN de TPC-DS permet de générer des fichiers de données dans un format (*fichier.dat*) avec différentes tailles selon un facteur d'échelle (*Scale Factor* - SF) ; 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, pour les requêtes sur un cluster de 19 noeuds.

Charge 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. Les requêtes concernées sont classées en six catégories, nommées respectivement : $SQ1$, $SQ2$, $SQ3$, $SQ4$, $SQ5$ et $SQ6$.

Configuration expérimentale : Pour mener nos expérimentations, nous avons mis en place deux environnements de stockage. Le premier est relationnel non distribué, et consiste en une machine intel-Core TMi7-4790S CPU@3.20 GHZ avec 8 Go de RAM, et un disque de 500 Go ; celle-ci fonctionne sous le système d'exploitation Ubuntu-14.04 LTS de 64 bits, utilisée comme serveur PostgreSQL dédié au stockage de l'entrepôt de données relationnel. Le deuxième est un environnement de stockage NoSQL distribué ; c'est un cluster d'ordinateurs composé d'un serveur maître (*NameNode*) et de 19 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 d'un disque d'1 To SATA. Les *DataNodes* sont tous équipés d'un processeur Intel-Core i5-2400M, de 8 Go de RAM, et de 300 Go d'espace disque. Ces machines fonctionnent sous Ubuntu-14.04 LTS de 64 bits et la version Java JDK 8. Nous avons utilisé la version Hadoop 2.6.0 et

le SGBD NoSQL orienté colonnes *HBase*-0.98.8-*Hadoop2*, dédié à la gestion des données dans un environnement distribué. Pour simplifier la manipulation des données et augmenter les performances du SGBD *HBase*, nous avons renforcé cette configuration avec une couche SQL dédiée à *HBase*, appelée *Phoenix* (v4.6.0). Pour utiliser cette dernière et interroger les données *HBase*, nous avons utilisé un client JDBC appelé *SQuirreL* (interface graphique). En outre, la machine *NameNode* est configurée pour jouer le rôle de *master server* du système *HDFS* et *Zookeper* de *HBase*. Les autres machines (*DataNodes*) sont considérées comme des *Region-Servers* de *HBase*.

Chargement et transfert des données : Le processus complet de chargement et de transfert des données du modèle relationnel au modèle non relationnel est présenté dans la figure 3. La connexion D-W entre les deux systèmes de gestion de données se fait grâce à un connecteur JDBC. Celui-ci est présent du côté *HBase* et du côté du SGBD relationnel utilisé. Afin de transférer efficacement les données de PostgreSQL à *HBase*, nous avons intégré toutes les fonctionnalités de *Sqoop*[5] (SQL-to-Hadoop) à notre outil TA-EDRC. Nous avons opéré un import de *Sqoop* pour le transfert des lignes des données en les spécifiant ainsi que les colonnes dans une vue intermédiaire que nous avons créée. Celle-ci est totalement dénormalisée, et est composée de l'ensemble des attributs. Les données appartenant à cette vue proviennent de la table de faits STORE_SALES et des instances associées de chaque dimension (CUSTOMER, CUSTOMER_DEMOGRAPHICS, CUSTOMER_ADDRESS, ITEM, TIME, DATE, HOUSEHOLD_DEMOGRAPHICS, PROMOTION, STORE, INCOME) où chaque ligne de données est identifiée par une clé *Key*, qui correspond à l'ordre séquentiel croissant des enregistrements de données de la table de faits. Cette dernière contient 287 997 024 d'enregistrements. En plus, cette clé est utilisée comme une clé de ligne, *Row_Key*, dans la table de données *HBase*.

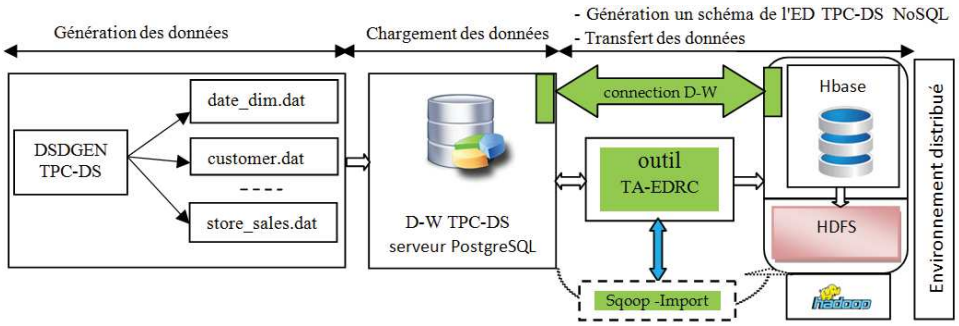

FIG. 3 – *Processus de chargement et de transfert des données.*

5.2 Résultats expérimentaux

Nous avons choisi 4 méthodes de construction des familles de colonnes : (1) l'ensemble des attributs des tables de faits et de dimensions est mis dans une même famille

5. https ://sqoop.apache.org.

de colonnes (approche "Plate") ; (2) chacune des tables de faits et de dimensions correspondrait à une famille de colonnes (approche "Naïve") ; (3) les familles sont déterminées selon le regroupement obtenu à l'aide de l'algorithme OEP ; (4) les familles de colonnes sont obtenues par l'algorithme des *k-means*. Lors de l'exécution de *OEP-FC* et *k-means*, nous avons utilisé 67 attributs. A chaque exécution, nous avons fait varier le seuil W (*OEP*) et le nombre k (*k-means*) entre 2 et 12 familles de colonnes. A chacune de ces variations, correspond un schéma différent de l'entrepôt de données. Nous avons exécuté toutes les séries de requêtes sur les différents schémas de l'entrepôt NoSQL issus des quatre configurations décrites ci-dessus. Les schémas obtenus sont implémentés sous *HBase* avec le même jeu de données (100 Go).

Par manque de place, nous ne pouvons pas présenter et discuter dans cet article tous les résultats obtenus. Nous avons choisi de présenter les expérimentations de la série SQ2 afin de mesurer l'impact du nombre de familles de colonnes sur le temps d'exécution des requêtes, notamment les requêtes $Rq3$.

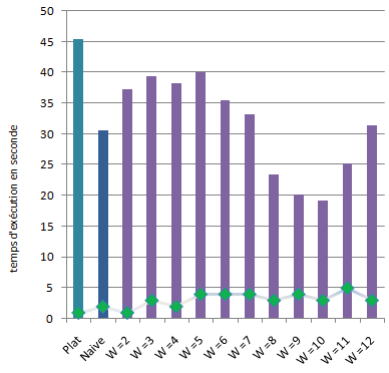

FIG. 4 – *Nombre de familles de colonnes parcourues par Rq3 et temps d'exécution ("Plat","Naïve" et OEP-FC)*

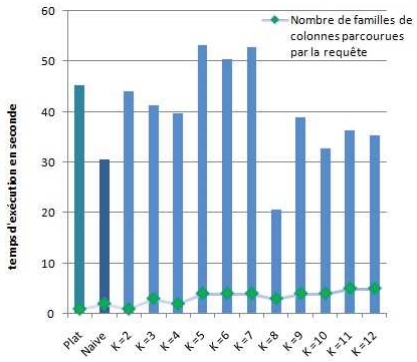

FIG. 5 – *Nombre de familles de colonnes parcourues par Rq3 et temps d'exécution (Plat,"Naïve" et k-means)*

Discussion : A partir des figures 4 et 5, nous constatons que le temps d'exécution de $Rq3$ varie d'un schéma d'entrepôt à un autre. La méthode OEP-FC donne de meilleurs résultats, avec un temps d'exécution égal à 19,140 secondes, pour ($W = 10$). Dans le cas de ce schéma, la requête porte sur les valeurs d'attributs de 3 familles de colonnes parcourues ayant la même petite taille des données (regroupement d'attributs équilibré). Cependant, avec l'algorithme *k-means*, ce temps est plus élevé, entre 50,438 et 53,287 s, pour ($5 \leq k \leq 7$). Cela est dû à l'augmentation des combinaisons des opérations (sélection, filtrage et agrégation) entre les familles de colonnes parcourues par la requête, pour déterminer l'ensemble des prédicats à considérer et identifier les valeurs des attributs. Donc, il est évident que ces opérations représentent une contrainte supplémentaire pour la phase de traitement des données au niveau de la mémoire. Dans ces schémas de l'entrepôt, la requête fait appel à 4 familles de colonnes non homogènes (regroupement d'attributs déséquilibré) ayant des tailles de données trop grandes ou

trop petites. Pour récupérer les données et les résultats intermédiaires des traitements, le système HBase exploite un nombre important de blocs de données HFile correspondant aux différentes familles de colonnes. Ces blocs sont dispersés sur plusieurs noeuds du cluster ; ce qui pénalise le flux de résultats intermédiaires, et nécessite, par conséquent, un temps d'exécution élevé. D'autre part, nous constatons aussi pour la méthode *k-means*, avec ($9 \leq k \leq 12$), les temps d'exécutions observés sont toutefois proches ; ils varient entre 32,782 et 38,912 s. Dans ces schémas, *Rq*3 sollicite une variation de 3 à 5 familles de colonnes ayant des tailles des données moyennes ; ce qui permet entre autre de réduire la surcharge des données au niveau de la mémoire. Cela se traduit par un temps d'exécution moins élevé par rapport à ($5 \leq k \leq 7$). En revanche, avec la méthode "Naïve", le temps d'exécution de la requête ne nécessite que 30,629 s ; ceci s'explique par le fait que celle-ci exploite moins de combinaisons d'opérations, 2 familles de colonnes ayant 2 tailles de données différentes (une avec 21 attributs : taille grande) et l'autre 12 attributs : taille moyenne). Ceci présente un certain équilibre entre l'ensemble de données et les traitements associés au niveau de la mémoire. Ceci permet de réduire le temps d'exécution de la requête. Pour terminer, les résultats montrent l'interdépendance entre la variation de nombre des familles de colonnes parcourues par la requête et leur taille de données. Nous remarquons aussi que la méthode OEP-FC présente de meilleures performances lorsque ($8 \leq W \leq 11$). Nous pensons que cela est dû au fait que OEP-FC prend en compte le choix de la taille des familles de colonnes. De ce fait, pour éviter les situations qui pénalisent le temps de traitement des requêtes, il est nécessaire de définir un nombre maximum d'attributs par familles de colonnes.

6 Conclusion

Dans cet article, nous avons présenté une approche de conception d'un entrepôt de données NoSQL en colonnes sur un cluster de plusieurs nœuds. Nous avons procédé à des regroupements d'attributs en familles de colonnes plus pertinentes, en utilisant les méthodes de fouille *OEP-FC* et *k-means*. Ceci nous a permis d'obtenir un gain de performance dans le traitement des requêtes décisionnelles. Nous avons développé un outil TA-EDRC dans le cadre de la transformation d'un entrepôt relationnel vers un entrepôt NoSQL en colonnes. Nous avons mené plusieurs expérimentations selon plus cas de familles de colonnes. Les résultats obtenus confirment l'intérêt de regrouper les colonnes en familles en tenant compte des besoins d'utilisateurs. Nous planifions par la suite d'étudier des stratégies de partitionnement et de placement intentionnels des blocs de données d'un entrepôt NoSQL en colonnes sur les différents nœuds du cluster.

Références

Abelló, A., J. Ferrarons, et O. Romero (2011). Building cubes with mapreduce. In *Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP*, DOLAP '11, New York, NY, USA, pp. 17–24. ACM.

Brewer, E. A. (2000). Towards robust distributed systems (abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, New York, NY, USA, pp. 7–. ACM.

Chevalier, M., M. El Malki, A. Kopliku, O. Teste, et R. Tournier (2015). Entrepôts de données multidimensionnelles nosql. In *EDA*, pp. 161–176.

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.

Dehdouh, K., O. Boussaid, et F. Bentayeb (2014). Columnar nosql star schema benchmark. In *Int. Conf. on Model and Data Engineering*, pp. 281–288. Springer.

Derrar, H., O. Boussaïd, et M. Ahmed-Nacer (2008). Répartition des données d'un entrepôt basée sur l'optimisation par essaim particulaire. In *EDA*, pp. 141–149.

Eberhart, R. C., J. Kennedy, et al. (1995). A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, Volume 1, pp. 39–43. New York, NY.

Leavitt, N. (2010). Will nosql databases live up to their promise ? *Computer 43*(2), 12–14.

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.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Volume 1, pp. 281–297. Oakland, CA, USA.

Navathe, S., S. Ceri, G. Wiederhold, et J. Dou (1984). Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems (TODS) 9*(4), 680–710.

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.

Summary

NoSQL Column Oriented model offer a flexible and highly non-normalized database schema. In this paper, we propose a method that transforms a relational data warehouse to a NoSQL one with distributed columns in a multi-node cluster. Our method is based on a strategy of grouping attributes from fact tables and dimensions, as families´ columns. In this purpose, we used two algorithms, the first one is a meta-heuristic algorithm, in this case the Particle Swarm Optimization : PSO, and the second one is the k-means algorithm. To evaluate our method, we use TCP-DS benchmark. We conducted several tests to evaluate these algorithms in the generation of families of columns and data partitions in the NoSQL Column Oriented Hbase DBMS, with a MapReduce paradigm and Hadoop distributed system.