HAL
open science

Entrepôts de données multidimensionnelles NoSQL

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, Ronan Tournier

To cite this version:

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, Ronan Tournier. Entrepôts de données multidimensionnelles NoSQL. 11e Journées Francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2015), Apr 2015, Bruxelles, Belgique. pp. 161-176. hal-01360873

HAL Id: hal-01360873

https://hal.science/hal-01360873v1

Submitted on 6 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive TOULOUSE Archive Ouverte (OATAO)
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 15242

The contribution was presented at EDA 2015 :
https://eda2015.ulb.ac.be/

To cite this version : Chevalier, Max and El Malki, Mohammed and Kopliku, Arlind and Teste, Olivier and Tournier, Ronan *Entrepôts de données multidimensionnelles NoSQL*. (2015) In: 11e Journées Francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2015), 2 April 2015 - 3 April 2015 (Bruxelles, Belgium).

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

Entrepôts de données multidimensionnelles NoSQL

Max Chevalier*, Mohammed El Malki*,** Arlind Kopliku*,
Olivier Teste*, Ronan Tournier*

*Université de Toulouse, IRIT UMR 5505, Toulouse, France
http://www.irit.fr Prénom.Nom@irit.fr
**Capgemini, 109, avenue du Général Eisenhower,
BP 53655- 31036 Toulouse, France
http://www.capgemini.com

Résumé. Les données des systèmes d'analyse en ligne (OLAP, On-Line Analytical Processing) sont traditionnellement gérées par des bases de données relationnelles. Malheureusement, il devient difficile de gérer des mégadonnées (de gros volumes de données, « Big Data »). Dans un tel contexte, comme alternative, les environnements « Not-Only SQL » (NoSQL) peuvent fournir un passage à l'échelle tout en gardant une certaine flexibilité pour un système OLAP. Nous définissons ainsi des règles pour convertir un schéma en étoile, ainsi que son optimisation, le treillis d'agrégats pré-calculés, en deux modèles logiques NoSQL : orienté-colonnes ou orienté-documents. En utilisant ces règles, nous implémentons et analysons deux systèmes décisionnels, un par modèle, avec MongoDB et HBase. Nous comparons ces derniers sur les phases de chargement des données (générées avec le benchmark TPC-DS), de calcul d'un treillis et d'interrogation.

1 Introduction

Pour faciliter le processus d'aide à la prise de décision, les données à analyser sont centralisées de manière uniforme dans un entrepôt de données Kimball et Ross (2013). Au sein de l'entrepôt, une analyse interactive des données est effectuée via un processus d'analyse en ligne (OLAP On-Line Analytical Processing), Colliat (1996), Chaudhuri et Dayal (1997). Les données sont souvent décrites au moyen d'un modèle multidimensionnel tel qu'un schéma en étoile, Chaudhuri et Dayal (1997), basé sur des sujets d'analyse (appelés faits) et des axes d'analyses (appelés dimensions). Les faits regroupent de manière conceptuelle des indicateurs d'analyse (des mesures). Ces mesures sont associées à des dimensions qui sont composées de différents niveaux de détails (niveau d'agrégation ou paramètres) permettant de constituer des perspectives (hiérarchies) d'analyse. Ces hiérarchies sont des structures employées pour faciliter le pré-calcul des agrégations induites par les analyses OLAP (par exemple, calculer des ventes annuelles à partir des valeurs mensuelles). Ces pré-calculs sont souvent modélisés via un treillis d'agrégats pré-calculés, Gray et al. (1996). Ainsi, un treillis représente l'ensemble des pré-calculs d'un schéma multidimensionnel où chaque noeud du treillis représente un agrégat et chaque arc représente le chemin pour calculer l'agrégat à partir d'autres agrégats. De nos jours, le volume des données d'analyses atteint des tailles critiques, Jacobs (2009), défiant les

approches classiques d'entreposage de données, dont les solutions actuelles reposent principalement sur des bases de données relationnelles (implémentations R-OLAP). Ces approches classiques sont remises en cause devant l'importance des volumes des données, Stonebraker (2012), Cuzzocrea et al. (2013), Dehdouh et al. (2014). Avec l'apparition des grandes plateformes Web, telles que Google, Facebook, Twitter, Amazon (...) des solutions pour gérer les mégadonnées [1] (Big Data) ont été développées. Elles sont basées sur des ap-proches décentralisées et ont largement contribué à l'apparition de solutions de gestion de données dites Not-Only-SQL (NoSQL), Stonebraker (2012). Le NoSQL permet d'envisager de nouvelles approches pour implanter un entrepôt de données, en particulier leur implantation multidimensionnelle.

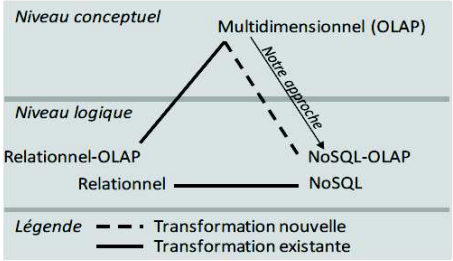

FIG. 1 – *Règles de transformations d'un modèle conceptuel en un modèle logique existantes.*

Comme l'illustre la figure 1, nous proposons des règles de transformation permettant de passer directement depuis un modèle conceptuel multidimensionnel (un schéma en étoile) vers un modèle logique NoSQL (modèle orienté colonnes ou orienté documents). Les règles prennent en compte également le treillis d'agrégats pré-calculés afin de l'implanter dans le modèle logique NoSQL ciblé. Comme illustré dans la figure 1, notre approche repose sur les niveaux d'abstraction des systèmes d'information où nous remplaçons le modèle logique traditionnel R-OLAP par deux modèles NoSQL. Notre contribution finale est multiple :

- nous définissons des règles permettant de transformer le modèle conceptuel multidimensionnel en modèles logiques : orienté colonnes et orienté documents ;
- nous définissons des règles pour inclure le treillis de pré-agrégats avec ces modèles ;
- nous implantons la base de données multidimentionnelle en MongoDB et HBase en appliquant nos règles afin d'étudier les phases de chargement des données, de calcul du treillis et d'exécution de requêtes OLAP en NoSQL.

Exemple : Notre étude repose sur un flux RSS de dépêches d'informations en provenance d'un site Web. Le contenu de ces dépêches (le fait) est analysé selon trois dimensions : *Mot-Clef* (contenu dans les dépêches), *Temps* (date de diffusion de la dépêche) et *Localisation* (lieu concerné par la dépêche). Le fait possède deux mesures : le nombre de dépêches (*NbDépêches*) et le nombre d'occurrences (*NbOccurrences*). Le schéma du modèle conceptuel est décrit dans la figure 2 et utilise un formalisme graphique basé sur Golfarelli et al. (1998) et Ravat et al. (2007). Un agrégat (un noeud du treillis) représente une agrégation des valeurs des mesures (un résultat de requête employant des fonctions d'agrégation SUM, COUNT, MAX) en fonction

1. Mégadonnées : données massives ou « Big Data », Journal Officiel de la République Fran-çaise, JORF du 22.08.2014

de niveaux de détails (paramètres) de différentes dimensions ; par exemple, agréger le nombre de dépêches en fonction du mois de publication et de la ville concernant les dépêches. Les paramètres étant organisés hiérarchiquement, les noeuds suivent cette organisation et constituent un treillis. Chaque agrégat (noeud) est pré-calculé et stocké au niveau logique. Dans notre approche, nos règles s'appliquent sur deux modèles logiques NoSQL : 1) un modèle orienté colonnes qui organise les données verticalement au moyen de familles de colonnes et 2) un modèle orienté documents qui organise les données de manière horizontale au sein de collections constituées de documents imbriqués.

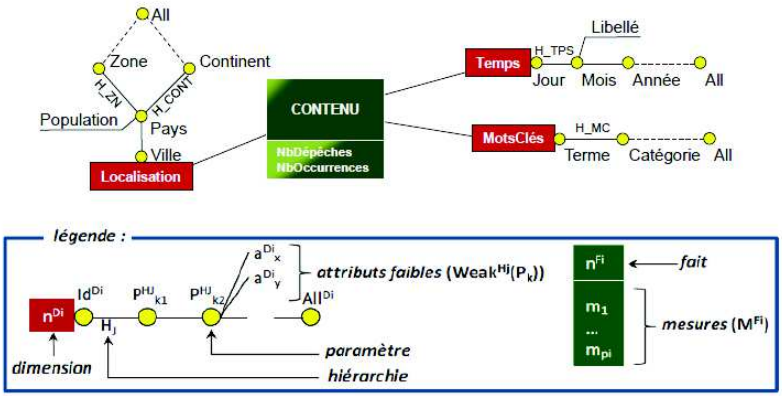

FIG. 2 – *Exemple de modèle conceptuel multidimensionnel.*

2 Etat de l'art

A notre connaissance, il n'existe pas de travaux présentant une solution directe et automatique pour convertir un entrepôt de données multidimensionnelles en NoSQL. Plusieurs travaux proposent de convertir les concepts d'un entrepôt de données en une implantation logique R-OLAP, Morfonios et al., 2007. Les bases de données multidimensionnelles sont principalement implantées en utilisant ces bases de données relationnelles. Des règles sont utilisées pour convertir les structures du niveau conceptuel (faits, dimensions et hiérarchies) en un modèle logique basé sur des relations (R-OLAP) Kimball et Ross (2013). En outre, de nombreux travaux se sont focalisés sur l'implantation de structures logiques d'optimisation basées sur des agrégats pré-calculés (appelés parfois vues matérialisées) comme Gray et al. (1996), Morfonios et al. (2007). Toutefois les implantations ROLAP souffrent du passage à l'échelle induit par des mégadonnées qui sont produites de nos jours. Des recherches sont en cours pour de nouvelles solutions utilisant des systèmes NoSQL, Lee et al. (2012). Notre approche consiste à revisiter ces processus pour implanter automatiquement des modèles conceptuels multidimensionnels en modèles NoSQL.

D'autres travaux ont etudié le processus de transformation de bases de données relationnelles en modèles logiques NoSQL (cf. Fig 1). Dans Li (2010) l'auteur propose une approche pour transformer une base de données relationnelles en une base de données NoSQL orientée

colonnes via HBase (Han et Stroulia (2012)). Dans Vajk et al. (2013), les auteurs présentent un algorithme pour transformer un schéma relationnel en un schéma NoSQL orienté documents via MongoDB (Dede et al. (2013)). Toutefois aucune de ces approches ne considère le niveau conceptuel des entrepôts de données. Elles se limitent au niveau logique : transformer un modèle relationnel quelconque en un modèle orienté colonnes.La dualité fait/dimension au niveau conceptuel requiert de garantir un certain nombre de contraintes, habituellement gérées via l'intégrité référentielle, qui, dans les approches logiques ne peuvent être garanties. Il n'existe pas actuellement d'approche pour transformer automatiquement et directement un modèle conceptuel multidimensionnel d'entrepôt de données en un modèle logique NoSQL. Malgré une certaine complexité, il est possible, aujourd'hui, de transformer un modèle conceptuel multidimensionnel en un modèle logique relationnel et, seulement ensuite, de le transformer en un modèle logique NoSQL. Cette transformation, utilisant comme modèle pivot le modèle relationnel, n'a pas été formalisée car les transformations ont été étudiées de manière indépendante.

Citons également deux travaux récents qui visent à développer un entrepôt de données dans un système NoSQL orienté colonnes (Dehdouh et al. (2014)) ou orienté clé-valeur (Zhao et Ye (2013)). L'objectif de ces articles est de proposer des benchmarks et n'est donc pas centré sur le processus de transformation de modèles. En outre, ces propositions se limitent à un seul modèle NoSQL, contrairement à nos travaux qui considèrent deux modèles orthogonaux permettant une distribution des données verticalement (modèle orienté colonnes) ou horizontalement (modèle orienté documents). En outre et contrairement aux approches existantes, nous prenons en compte dans le processus de transformation les hiérarchies en spécifiant des règles de transformation pour gérer les treillis d'agrégats pré-calculés.

3 Modèle conceptuel multidimensionnel

Pour assurer un processus de transformation générique, nous définissons le modèle multidimensionnel employé au niveau conceptuel Ravat et al. (2008). Un **schéma multidimensionnel**, noté *E*, est défini par (F^E, D^E, $Star^E$) où :

- $F^E = \{F_1, \ldots, F_n\}$ un ensemble fini de faits,
- $D^E = \{D_1, \ldots, D_m\}$ un ensemble fini de dimensions,
- $Star^E : F^E \rightarrow 2^{D^E}$ est une fonction qui associe les faits de F^E à des ensembles de dimensions, selon lesquelles ils peuvent être analysés (2^{D^E} étant l'ensemble des parties de l'ensemble D^E).

Une **dimension**, notée $D_i \in D^E$ (abusivement notée D), est définie par (N^D, A^D, H^D) où :

- N^D est le nom de la dimension,
- $A^D = \{a_1^D, \ldots, a_u^D\} \cup \{id^D, All^D\}$ est un ensemble d'attributs de dimension,
- $H^D = \{H_1^D, \ldots, H_v^D\}$ est un ensemble de hiérarchies.

Une **hiérarchie**, notée $Hi \in H^D$, est définie par (N^{Hi}, $Param^{Hi}$, $Weak^{Hi}$) où :

- N^{Hi} est le nom de la hiérarchie,
- $Param^{Hi} = < id^D, p_1^{Hi}, \ldots, p_{vi}^{Hi}, All^D >$ est un ensemble ordonné de $v_i + 2$ attributs appelés paramètres qui représentent les graduations de la dimension, $\forall k \in [1 \ldots v_i]$, $p_k^{Hi} \in A^D$,

– $Weak^{Hi} : Param^{Hi} \rightarrow 2^{A^D - Param^{Hi}}$ est une fonction associant aux paramètres un ou plusieurs attributs faibles.

Un **fait**, noté $F \in F^E$, est défini par (N^F, M^F) où :

– N^F est le nom du fait,
– $M^F = \{f_1(m_1^F), \ldots, f_v(m_v^F)\}$ est un ensemble de mesures associées à une fonction d'agrégation f_i.

4 Conversion en un modèle NoSQL orienté colonnes

Le modèle orienté colonnes considère chaque enregistrement comme une clef associée à une valeur décomposée en plusieurs colonnes. Les données forment un ensemble de lignes dans une table composée de colonnes qui peuvent varier d'une ligne à une autre.

4.1 Modèle orienté colonnes NoSQL

Dans les bases de données relationnelles, la structure des données est déterminée en avance avec un nombre limité de colonnes (quelques milliers), chacune similaire pour tous les enregistrements (« n-uplets »). Le modèle orienté colonnes fournit un schéma flexible (des colonnes non typées) avec un très grand nombre de colonnes pouvant varier entre chaque enregistrement (chaque ligne), sans engendrer de valeurs nulles. Une base de données orientée colonnes est un ensemble de tables qui sont définies ligne par ligne (mais dont le stockage physique est organisé par groupe de colonnes, « familles de colonnes », entraînant un partitionnement vertical des données). Ainsi, dans ces systèmes chaque table est une représentation logique de lignes et de leurs familles de colonnes respectives. Une famille de colonnes peut contenir un très grand nombre de colonnes. Pour chaque ligne, une colonne contient une valeur.

Une **table** $T = \{R_1, \ldots, R_n\}$ est un ensemble de lignes R_i. Une ligne R_i = (Key_i, $CF_i^1, \ldots, CF_i^m$) est composée d'une clef de ligne Key_i et d'un ensemble de familles de colonnes noté CF_i^j.

Une **famille de colonnes** CF_i^j = (C_i^{j1}, v_i^{j1}),. . ., (C_i^{jp}, v_i^{jp}) consiste en un ensemble de colonnes, chacune associée à une valeur atomique. Chaque valeur atomique peut être « historisée » avec une étiquette temporelle (cf. Fig 3). Dans cet article, ce principe utile dans le cadre de gestion des historiques, Wrembel (2009), ne sera pas employé.

La flexibilité d'une base NoSQL orientée colonnes permet de gérer l'absence de certaines colonnes entre les différentes lignes de la table. Toutefois, dans notre contexte, les données sont habituellement fortement structurées Malinowski et Zimányi (2006), impliquant que la structure d'une famille de colonnes sera la même pour toutes les lignes de la table.

4.2 Règles de correspondance avec le niveau conceptuel

Les éléments (faits, dimensions) du modèle conceptuel multidimensionnel doivent être transformés en éléments du modèle NoSQL orienté colonnes (cf. Fig 4).

– Chaque schéma en étoile conceptuel (chaque fait F_i de F^E, et ses dimentions $Star(F_i)$) est transformé en une table T^E.
– Le fait F_i est transformé en une famille de colonnes CF^M de T^E dans laquelle chaque mesure m_i est une colonne $C_i \in CF^M$.

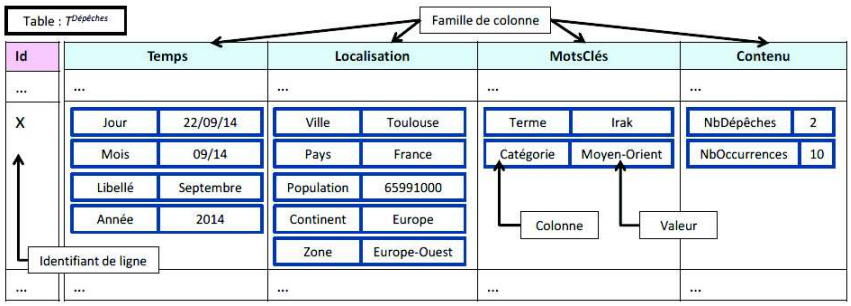

FIG. 3 – *Exemple d'une ligne dans la table* $T^{Dépêches}$.

– Chaque dimension $D_i \in Star^E(F_i)$ est transformée en une famille de colonnes CF^{D_i} où chaque attribut de dimension $A_i \in A^{D_i}$ (paramètres et attributs faibles) de la dimension D_i est transformé en une colonne C_i de la famille de colonnes CF^{D_i} ($C_i \in CF^{D_i}$), à l'exception du paramètre All^D.

Chaque instance de fait et les instances associées des dimensions sont transformées en une ligne R_x de T_E. L'instance du fait est donc composée de la famille de colonnes CF^M (les mesures et leurs valeurs respectives) et des familles de colonnes des dimensions $CF^{D_i} \in CF^{D^E}$ (les attributs de chaque dimension et leur valeurs respectives).

A l'instar d'un schéma en étoile dénormalisé, Kimball et Ross (2013), l'organisation hiérarchique des attributs n'est pas représentée dans le système NoSQL, mais elle est exploitée pour construire le treillis d'agrégats. Bien que nous ne considérions pas les processus ETL dans cet article, les hiérarchies peuvent également servir lors de la construction des instances afin de respecter les contraintes induites par ces structures conceptuelles notamment selon le principe de hiérarchie stricte, Malinowski et Zimányi (2006).

Exemple. Soit $E^{Dépêches}$ un schéma multidimensionnel conceptuel implanté dans une table $T^{Dépêches}$ (cf. Fig 4). Le fait ($F^{Contenu}$) et ses dimensions (D^{Temps}, $D^{Localisation}$, $D^{MotsClés}$) sont implantés dans quatre familles de colonnes CF^{Temps}, $CF^{Localisation}$, $CF^{MotsClés}$, $CF^{Contenu}$. Chaque famille de colonne contient un ensemble de colonnes correspondant soit à des attributs de dimensions, soit à des mesures du fait. Par exemple la famille de colonnes $CF^{Localisation}$ est composée des colonnes C^{Ville}, C^{Pays}, $C^{Population}$, $C^{Continent}$ et C^{Zone}.

Cette implantation a l'avantage de regrouper au sein d'une même table aussi bien les données du fait que celles des dimensions. Cela permet d'éviter des jointures, mais a pour conséquence une importante redondance dans les données (les données des dimensions sont dupliquées pour chaque instance du fait). La conséquence est une augmentation du volume total des données mais pour favoriser une diminution du temps de calcul des requêtes d'interrogation. Dans un contexte NoSQL les problèmes liés au volume des données peuvent être palliés par distribution massive et extensible des données. En outre, cette redondance est motivée par le fait que, dans le contexte des entrepôts de données, les mises à jour des données sont essentiellement des insertions de nouvelles données ; les coûts additionnels imputés aux changements sont donc limités dans ce contexte.

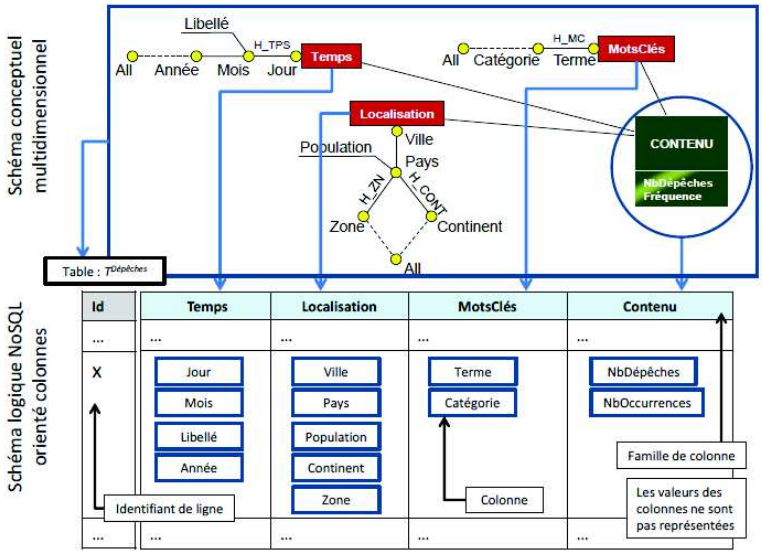

FIG. 4 – *Transformation du MCM au modèle logique NoSQL orienté colonnes*

4.3 Règles de correspondances avec le treillis

Nous utiliserons les notations suivantes pour définir les règles de correspondance du treillis. Un **treillis d'agrégats pré-calculés**, noté L, est un ensemble de noeuds A^L reliés par des arcs E^L (chemins envisageables pour calculer les agrégats les uns à partir des autres). Un noeud (dit d'agrégat) $A \in A^L$ est composé d'un ensemble de paramètres p_i (un par dimension) et d'un ensemble de mesures m_i agrégées $f_i(m_i)$. $A =< p_1, ..., p_k, f_1(m_1), ..., f_v(m_v) >, k \leq m$ (m étant le nombre de dimensions, v étant le nombre de mesures du fait). Pour implanter le treillis dans une base de données NoSQL orientée colonnes, les règles suivantes sont appliquées :

- Chaque noeud d'agrégat $A \in A^L$ est stocké dans une table T^A, $T^A \neq T^E$.
- Pour chaque dimension D_i correspondant au noeud A, une famille de colonne CF_{Di} est créée. Chaque attribut a_i de la dimension est stocké dans une colonne C_{a_i} de CF_{Di},
- L'ensemble des mesures agrégées est aussi stocké dans une famille de colonnes CF_F où chaque mesure agrégée est stockée dans une colonne C.

Exemple. Considérons le treillis associé au schéma des Dépêches (cf. Fig 2). Ce treillis est stocké dans un ensemble de tables. Le noeud ($Terme$, $Jour$, $ALL^{Localisation}$) est stocké dans une table $T^{Terme_Jour_All}$ composée des familles de colonnes $CF^{MotsClés}$ et CF^{Temps}.

Les attributs $Terme$ et $Jour$ sont stockés dans une colonne $C^{MotsClés}$ et C^{Temps} respectivement. Les deux mesures ($NbDepeches$ et $NbOccurrences$) sont stockées dans une famille de colonnes $CF^{Contenu}$. Nous considérons un pré-calcul de l'ensemble des agrégats, Kimball et Ross (2013), mais il est également possible de spécifier des critères de sélection des agrégats à pré-calculer.

5 Conversion en un modèle NoSQL orienté documents

Le modèle orienté documents considère chaque enregistrement comme un document : des paires « attribut/valeur » ; ces valeurs sont soit atomiques, soit complexes (imbriquées dans un sous enregistrement), chaque sous-enregistrement pouvant être assimilé à un document.

5.1 Modèle orienté documents NoSQL

Dans le modèle orienté documents chaque clef est associée à une valeur, structurée comme un document. Ces valeurs sont groupées en collections. Un document est une hiérarchie d'éléments pouvant être soit des valeurs atomiques, soit des documents. Dans l'approche NoSQL, le schéma des documents n'est pas établi à l'avance.

Formellement, une base de données orientée documents peut être définie comme une **collection** $C^D = \{D_1, \ldots, D_n\}$ constituée d'ensemble de documents D_i. Chaque **document** D_i est défini par un ensemble de paires $\{(Att_i^1, V_i^1)$,..., $(Att_i^{mi}, V_i^{mi})\}$ où Att_i^j est un attribut avec $j \in [1, m_i]$ (similaire à une clef) et V_i^j est une valeur qui peut être de deux formes :

- soit la valeur est atomique ;
- soit la valeur est elle-même composée d'un document imbriqué défini comme un nouvel ensemble de paires (attribut, valeur).

Nous distinguons les **attributs simples** dont les valeurs sont atomiques des **attributs composés** dont les valeurs sont des **documents imbriqués** (cf. Fig 5).

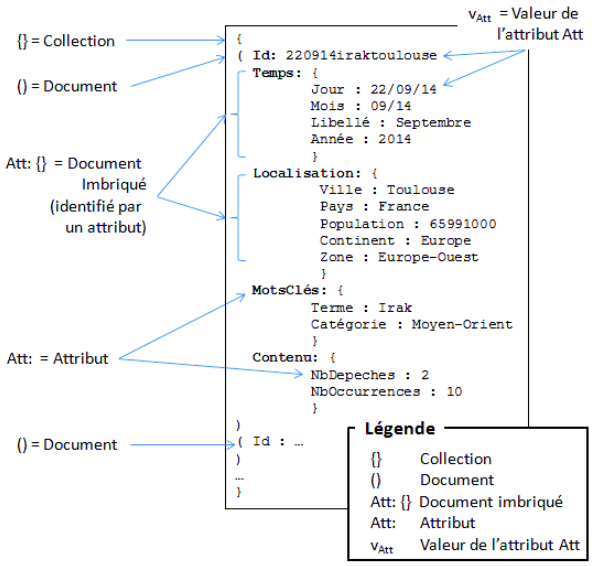

FIG. 5 – *Représentation graphique de la collection* $C^{Dépêches}$.

5.2 Règles de correspondances avec le niveau conceptuel

Via le modèle NoSQL orienté documents, les données sont organisées en lignes et colonnes, mais l'ensemble est structuré en documents imbriqués (cf. Fig 6).

- Chaque schéma en étoile (chaque fait F_i et ses dimensions associées $Star(F_i)$) est traduit en une collection C^E.
- Le fait F_i est traduit en un attribut composé. Chaque mesure m_i est traduite en un attribut simple C^E.
- Chaque dimension $D_i \in Star^E(F^i)$ est convertie en un attribut composé Att^{CD} (un document imbriqué). Chaque attribut $A_i \in A^D$ (paramètres et attributs faibles) de la dimension D_i est converti en un attribut simple Att^{A_i} contenu dans Att^{CD}.

Une instance de fait est convertie en un document. Les valeurs des mesures sont combinées au sein d'un document imbriqué. Chaque dimension est convertie également en un document imbriqué contenu dans le même document que l'instance de fait. L'organisation hiérarchique de la dimension n'est pas préservée, mais celle-ci permet de construire le treillis d'agrégats.

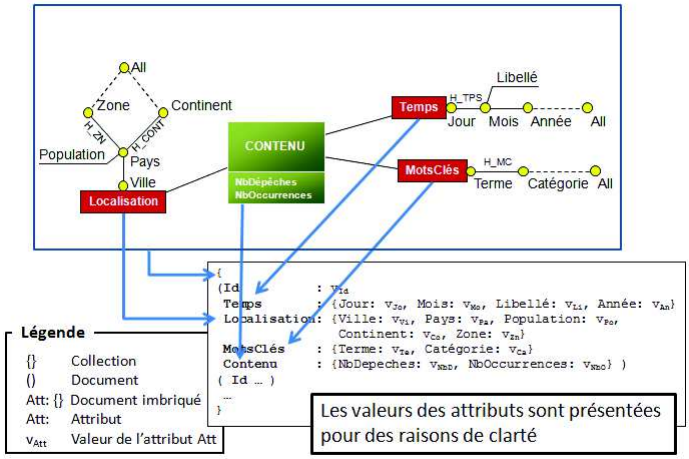

FIG. 6 – *Transformation du MCM au modèle logique NoSQL orienté documents*

Exemple. Le document noté d_x est composé de quatre documents imbriqués, $Att^{Contenu}$, regroupant les mesures et $Att^{Localisation}$, $Att^{MotsClés}$, Att^{Temps}, correspondant aux instances des dimensions associées.

5.3 Règles de correspondances avec le treillis

Nous utilisons la même définition pour le treillis d'agrégats que précédemment (section 4.3). Toutefois, lors de l'utilisation d'une base de données NoSQL orientée documents, les règles d'implantation sont :

- Chaque noeud A est stocké dans une collection séparée C_A, $C_A \neq C^E$.

– Pour chaque dimension D_i correspondant au noeud A, un attribut composé (un document imbriqué) Att_{Di}^{CD} est créé, chaque attribut a_i de cette dimension est stocké dans un attribut simple de Att_{Di}^{A},
– L'ensemble des mesures agrégées est stocké dans un attribut composé Att_{F}^{CD} où chaque mesure agrégée est stockée comme un attribut simple Att_{mi}.

Exemple. Considérons le treillis associé au schéma des Dépêches. Ce treillis est stocké dans une collection $C^{Dépêches}$. Le noeud ($Mois$, $Pays$, $ALL^{MotsClés}$) est stocké dans un document. Les attributs concernés dans les dimensions Temps et Localisation sont stockées dans des documents imbriqués d^{Temps} et ^{dLocalisation}. Ainsi, l'attribut *Mois* est stocké comme un attribut simple dans le document imbriqué d^{Temps}. L'attribut *Pays* est stocké dans le document imbriqué $d^{Localisation}$ comme un attribut simple. Les deux mesures sont stockées dans un document imbriqué dénoté d^{Fact}.

6 Expérimentations

Notre but est d'étudier l'instanciation des modèles logiques avec leur temps de géné-ration, au niveau du schéma en étoile au travers du chargement des données et au niveau de la construction du treillis. Nous étudions aussi le comportement de ces deux modèles NoSQL face à un jeu de requêtes utilisateurs.

6.1 Protocole expérimental

Nous utilisons HBase pour tester le modèle orienté colonnes et MongoDB pour celui orienté documents. Le benchmark TPC-DS [2] est utilisé pour générer les données. Une fois les données chargées, le treillis est calculé par agrégations Map/Reduce via HBase et MongoDB.

Matériel et systèmes de gestion de données. Nous employons un cluster de trois PC (i5-4 coeurs, 8Go de RAM, 2To de disque, 1Gb/s de réseau), chacun étant un noeud de données(datanode) et l'un jouant aussi le rôle de namenode. Les bases NoSQL sont HBase (v.0.98) et MongoDB (v.2.6). Toutes deux sont des systèmes clé-valeurs utilisant respectivement un stockage orienté colonnes et orienté documents. Hadoop (v.2.4) est utilisé en tant que système distribué de stockage pour répartir les données entre les noeuds du cluster et Hive (v.0.13.1) pour faciliter l'expression des requêtes sur HBase.

Jeu de données. Le benchmark TPC-DS (Decision Support) fournit un total de 7 « tables de faits » et 17 « tables de dimensions » partagées. Les données correspondent à une activité de vente. Nous utilisons uniquement la table de fait *store_sales* et ses 10 tables de dimensions associées dont certaines sont des regroupements de niveaux hiérarchiques supérieurs à d'autres dimensions. Nous considérons l'agrégation des dimensions suivantes : *date* (*day*, *month*, *year*), *customer address* (*city*, *country*), *store address* (*city*, *country*) et *item* (*class*, *category*).

Génération et chargement des données. L'outil DSGen (v.1.3) génère des fichiers de données dans un format proche du CSV (Coma Separated Values), à raison d'un fichier par fait ou dimension. Nous ne conservons que les mesures de *store_sales* avec les valeurs des dimensions associées (en joignant les données des différents fichiers). Les volumes générés sont : 1Go, 10Go et 100Go. Dans sa version actuelle, MongoDB recommande un format JSon en

2. TPC Decision Support benchmark v.1.3.0 de , www.tpc.org/tpcds

entrée, ce qui représente un volume 3,4 fois plus important en raison du balisage. Le processus complet est présenté en figure 7. Les données sont chargées à partir de fichiers dans HBase et MongoDB en utilisant leurs instructions propres.

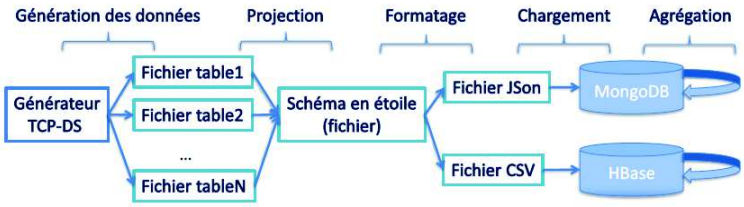

FIG. 7 – *Processus du protocole expérimental.*

Calcul du treillis. Pour calculer le treillis d'agrégats pré-calculés, des fonctions Map/Reduce, efficaces dans un environnement distribué, de HBase et MongoDB sont utilisées. Les données sont agrégées via les dimensions **item**, **store** et **customer**. Quatre niveaux d'agrégats sont calculés : toutes les combinaisons des trois dimensions ; puis de deux dimensions et d'une seule dimension, le tout sur l'ensemble du jeu de données. Pour chaque niveau d'agrégation, les fonctions d'agrégation sont : *Max, Min, Sum* et *Count*.

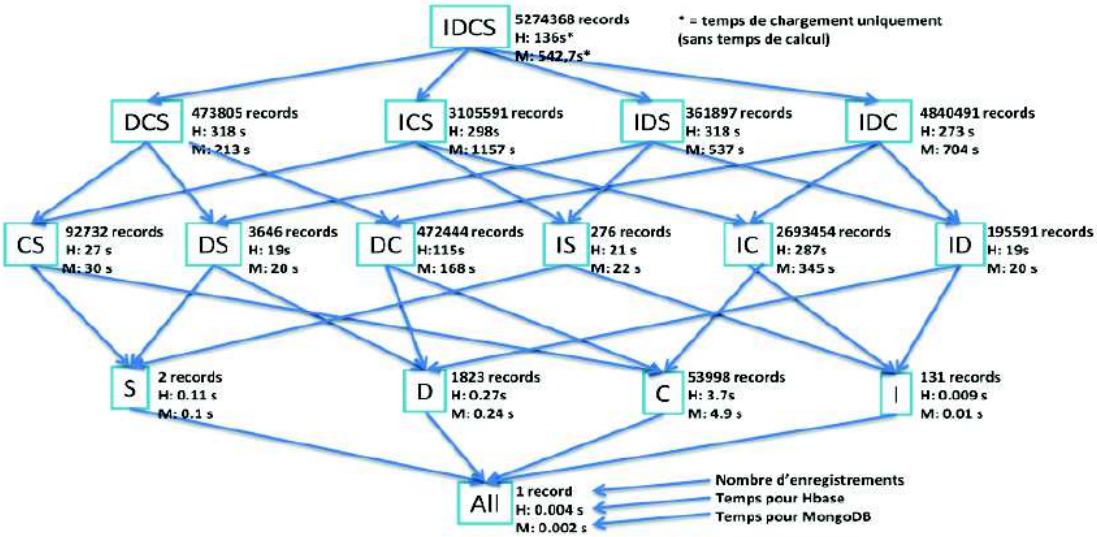

FIG. 8 – *Le treillis d'agrégats avec le temps de calcul (en secondes) et la taille (en enregistrement/documents). Les dimensions sont abréviés (D : Date, I : item, S : store, C : customer)..*

Requêtes. Nous avons généré 12 requêtes organisées par dimensionnalité et sélectivité. La dimensionnalité concerne le nombre de dimensions dans les clauses de regroupement (équivalent au *Group By* SQL) : 1D = 1 dimension, 2D = deux dimensions, 3D = trois dimensions.

TAB. 1 – *Filtres des requêtes.*

Filtre / Restriction	Nombre de lignes	Classe de sélectivité
store.c_city='Midway', date.d_year='2002'	883641	L
store.c_city='Midway', item.i_class='bedding'	31165	A
customer.ca_city='Sullivan', date.d_year='2003'	31165	H
customer.ca_city='Sullivan', item.i_class='bedding'	27	VH

TAB. 2 – *Temps de chargement des données en fonction du système NoSQL.*

Volume des données	**1GB**	**10GB**	**100GB**
MongoDB	9.045m	109.58m	132m
HBase	2.26m	2.078m	10,3m

La sélectivité est le degré de filtrage des données quand on applique les conditions (équivalent au *Where* SQL). Le niveau de sélectivité est divisé en 4 groupes selon la formule suivante :

- *very high (VH)* : sélection de 0 à e^k lignes,
- *high (H)* : sélection de e^k à e^{2k} lignes,
- average (A) : sélection de e^{2k} à e^{3k} lignes,
- low (L) : sélection de e^{3k} à e^{4k} lignes,

avec k défini en fonction de la taille $|C|$ de la collection C ; nous utilisons $k = \lceil \frac{\ln |C|}{4} \rceil$. Cette formule nous permet de diviser en 4 parties en fonction du nombre de lignes sélectionnées. La fonction est sub linéaire. Nous avons opté pour une combinaison de deux dimensions sur les restrictions (résumé dans la table 1).

Au total, nous avons 12 requêtes : 4 par dimension et 3 par sélectivité. Nous effectuons nos expériences sur le jeu de données de taille de 1Go. Le calcul des requêtes se fait selon deux stratégies : *wL* (with lattice) en utilisant le noeud pré-agrégat optimal, *nL* (no lattice) sans l'utilisation du treillis, donc sur les donnée détaillées.

6.2 Résultats expérimentaux

Chargement des données. Le chargement des données confirme que HBase est plus rapide pour charger des données. Toutefois il faut noter que nous n'avons pas réglé finement les noeuds du cluster pour optimiser les performances en fonction du système cible choisi. En outre, pour un nombre identique d'enregistrements, l'écart en taille du format JSON implique un transfert réseau plus lourd pour MongoDB.

Calcul du treillis. Les résultats sont présentés en figure 8. Le niveau supérieur correspond à *IDCS* (les données détaillées : *Item*, *Date*, *Customer*, *Store*). Sur le second niveau, nous conservons toutes les combinaisons de trois dimensions, puis de deux et ainsi de suite. Pour chaque noeud nous montrons le nombre d'enregistrements/documents (*records*) qu'il contient

ainsi que le temps nécessaire au calcul (en secondes) annoté, « H : » pour HBase et « M : » pour MongoDB.

Dans HBase, le temps total pour calculer tous les agrégats est de 1700 secondes avec respectivement, 1207s, 488s, 4s et 0,004s pour chaque niveau complet (en partant du niveau le plus détaillé). Dans MongoDB, le temps total pour calculer tous les agrégats est de 3210 secondes avec respectivement 2611s, 594s, 5s et 0,002s par niveau complet. Systématiquement, la taille des données étant plus petite, le calcul des niveaux les moins détaillés est beaucoup plus rapide. Cette taille des agrégats (en nombre d'enregistrements) décroit aussi lorsque nous descendons le long de la hiérarchie : 8,7 millions (niveau 2), 3,4 millions (niveau 3), 55 000 (niveau 4) et 1 enregistrement au niveau le plus bas.

Requêtes. La table 3 résume les résultats requête par requête. Les requêtes ne sont pas écrites, mais seulement les dimensions de regroupement, le niveau de sélectivité et les temps d'exécution sur MongoDB et HBase. Sur la table, nous spécifions le pré-agrégat qui permet d'optimiser l'exécution de la requête. Nous comparons les résultats sur deux cas : utilisation des noeuds du treillis (pré-agrégats) pour optimiser l'exécution de la requête ou l'utilisation de base C^{store_sales} ou T^{store_sales}.

TAB. 3 – *Résultats d'exécution des requêtes utilisateurs où I = Item, C = Customer, D = Date, S = Store, No = Number, Dim = dimensions, Pré-ag = pré-agrégat optimal, Slct = sélectivité, Mg= MongoDB, Hb = Hbase, +T = avec treillis, -T = sans le treillis*

No.	**Dim**	**Pré-ag.**	**Slct.**	**Mg+T**	**Mg-T**	**Hb+T**	**Hb-T**
1.	I	DIS	L	254ms	3640ms	976 ms	48000ms
2.	S	IS	A	5ms	1881ms	28ms	38000ms
3.	D	CD	H	138ms	1810ms	414ms	43 000ms
4.	C	CI	VH	1008ms	1797ms	3528 ms	42 000ms
5.	IS	DIS	H	115ms	2060ms	356 ms	42000ms
6.	CI	CIS	A	905ms	2237ms	2986ms	43000ms
7.	DS	DS	L	16ms	3921ms	59 ms	39000ms
8.	CD	CDI	VH	1406ms	2049ms	4246 ms	43000ms
9.	DIS	DIS	L	249ms	4311ms	781 ms	43000ms
10.	CIS	base	A	2198ms	2198ms	8754 ms	42000ms
11.	CDI	CDI	L	1420ms	2052ms	4261 ms	41000ms
12.	CDS	base	VH	2051ms	2051ms	6094 ms	41000ms

Le niveau de sélectivité est précisé dans la table 3. Pour le calcul sans l'aide du treillis, nous observons que les requêtes moins sélectives (plus de lignes sélectionnées) prennent plus de temps d'exécution. Cette observation ne s'applique plus quand nous utilisons le treillis pour optimiser les résultats. En effet, l'utilisation du treillis permet d'améliorer significativement l'exécution de certaines requêtes, ce qui explique une irrégularité sur les résultats concernés.

Discussion. Ces expérimentations montrent que le chargement dans le système NoSQL est sensiblement plus rapide dans une approche verticale (de type HBase) que l'approche horizontale (de type MongoDB). Le treillis est calculé en utilisant les fonctions d'agrégations

spécifiques NoSQL. Par ailleurs le temps moyen d'exécution des requêtes sur MongoDB est de 0,78s (780ms) en utilisant le treillis contre 2,50s sans ; sur HBase le temps moyen est de 2,20ms avec le treillis contre 42s sans. L'amélioration est considérable. Ces premières expérimentations doivent être complétées : nous souhaitons comparer nos transformations avec des règles proposant des implantations alternatives du schéma conceptuel en étoile dans le modèle logique NoSQL.

7 Conclusion

Cet article décrit un processus pour implanter automatiquement un entrepôt de données multidimensionnelles dans un système NoSQL. Nous avons défini un processus qui convertit un schéma multidimensionnel conceptuel (schéma en étoile) en deux schémas logiques NoSQL : un modèle orienté colonnes ou un modèle orienté documents. Nous avons étendu le processus au treillis d'agrégats pré-calculés, en fournissant des règles adaptées à chaque modèle logique pour assurer l'implantation du treillis. En outre, nous avons conduit un ensemble d'expérimentations pour étudier le processus de chargement (chargement du schéma en étoile et calcul des agrégats du treillis) et d'interrogation. Nous avons utilisé HBase comme solution NoSQL orientée colonnes et MongoDB comme solution orientée documents. Des jeux de données (1Go, 10 Go et 100Go) basés sur le benchmark TPC-DS ont été utilisés. Les premiers résultats montrent des performances meilleures pour l'approche verticale orientée colonne sur la phase de chargement des données et de pré-calculs d'agrégations. Par contre, l'approche orientée documents est plus performante en interrogation. Nous comptons élargir notre processus de conversion en intégrant plusieurs stratégies de transformation automatisables tenant compte de la distribution des données et d'un ensemble plus large de requêtes utilisateurs.

Références

Chaudhuri, S. et U. Dayal (1997). An overview of data warehousing and OLAP technology. *SIGMOD Record 26*, 65–74.

Colliat, G. (1996). OLAP, relational, and multidimensional database systems. *SIGMOD Record 25*(3), 64–69.

Cuzzocrea, A., L. Bellatreche, et I.-Y. Song (2013). Data warehousing and OLAP over big data : current challenges and future research directions. In *16th International workshop on Data warehousing and OLAP (DOLAP)*, pp. 67–70. ACM.

Dede, E., M. Govindaraju, D. Gunter, R. S. Canon, et L. Ramakrishnan (2013). Performance evaluation of a MongoDB and Hadoop platform for scientific data analysis. In *4th ACM Workshop on Scientific Cloud Computing*, pp. 13–20. ACM.

Dehdouh, K., O. Boussaid, et F. Bentayeb (2014). Columnar NoSQL star schema benchmark. In *4th International Conference on Model and Data Engineering (MEDI)*, LNCS 8748, pp. 281–288. Springer.

Golfarelli, M., D. Maio, et S. Rizzi (1998). The dimensional fact model : A conceptual model for data warehouses. *International Journal of Cooperative Information Systems 7*, 215–247.

Gray, J., A. Bosworth, A. Layman, et H. Pirahesh (1996). Data cube : A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *20th International Conference on Data Engineering (ICDE)*, pp. 152–159.

Han, D. et E. Stroulia (2012). A three-dimensional data model in HBase for large time-series dataset analysis. In *6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, pp. 47–56. IEEE.

Jacobs, A. (2009). The pathologies of big data. *Communications of the ACM 52*(8), 36–44.

Kimball, R. et M. Ross (2013). *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling* (3rd ed.). John Wiley & Sons, Inc.

Lee, S., J. Kim, Y.-S. Moon, et W. Lee (2012). Efficient distributed parallel top-down computation of ROLAP data cube using mapreduce. In *14th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, LNCS 7448, pp. 168–179. Springer.

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

Malinowski, E. et E. Zimányi (2006). Hierarchies in a multidimensional model : From conceptual modeling to logical representation. *Data Knowl. Eng. 59*(2), 348–377.

Morfonios, K., S. Konakas, Y. E. Ioannidis, et N. Kotsis (2007). ROLAP implementations of the data cube. *ACM Computing Surveys 39*(4).

Ravat, F., O. Teste, R. Tournier, et G. Zurfluh (2008). Algebraic and graphic languages for OLAP manipulations. *IJDWM 4*(1), 17–46.

Ravat, F., O. Teste, R. Tournier, et G. Zurlfluh (2007). A conceptual model for multidimensional analysis of documents. In *Conceptual Modeling (ER)*, LNCS 4801, pp. 550–565. Springer.

Stonebraker, M. (2012). New opportunities for new sql. *Commun. ACM 55*(11), 10–11.

Vajk, T., P. Feher, K. Fekete, et H. Charaf (2013). Denormalizing data into schema-free databases. In *2013 IEEE 4th International Conference CogInfoCom*, pp. 747–752.

Wrembel, R. (2009). A survey of managing the evolution of data warehouses. *IJDWM 5*(2), 24–56.

Zhao, H. et X. Ye (2013). A practice of TPC-DS multidimensional implementation on NoSQL database systems. In *5th TPC Technology Conference on Performance Characterization and Benchmarking*, LNCS 8391, pp. 93–108. Springer.

Summary

The traditional OLAP (On-Line Analytical Processing) systems store data in relational databases. Unfortunately, it is difficult to manage big data volumes with such systems. As an alternative, NoSQL systems (Not-only SQL) provide scalability and flexibility for an OLAP system. We define a set of rules to map star schemas and its optimization structure, a pre-computed aggregate lattice, into two logical NoSQL models: column-oriented and document-oriented. Using these rules we analyse and implement two decision support systems, one for each model (using MongoDB and HBase). We compare both systems during the phases of data (generated using the TPC-DS benchmark) loading, lattice generation and querying.