Entrepôts de données orientés documents : cuboïdes étendus - Modèles et cuboïdes NoSQL orientés documents

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

Entrepôts de données orientés documents : cuboïdes étendus

Modèles et cuboïdes NoSQL orientés documents

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

Université de Toulouse, IRIT (UMR 5505), Toulouse, France
Max.Chevalier@irit.fr ; Mohammed.ElMalki@irit.fr ; Arlind.Kopliku@irit.fr ; Olivier.Teste@irit.fr ; Ronan.Tournier@irit.fr

RÉSUMÉ. *Avec l'essor ces dernières années des grandes plateformes Web (par exemple, Google, Facebook, Twitter, Amazon), ont été développées des solutions de gestion des mégadonnées (big data) basées sur des approches décentralisées permettant la gestion et le stockage de gigantesques masses de données. Cette approche décentralisée repose sur le principe de la scalabilité, c'est-à-dire l'ajustement d'une manière progressive et continue du stockage et des traitements au volume des données. Ce type d'architecture distribuée a connu récemment le développement de systèmes de gestion de fichiers massivement distribués et de nouvelles techniques de parallélisation massive des traitements. Adossés à ce contexte de distribution massive, différents systèmes de stockage sont apparus ces dernières années. Ces systèmes, qualifiés de systèmes not-only-SQL (ou NoSQL), relaxent les fondements de l'approche relationnelle pour pouvoir supporter les masses de données distribuées. De ce fait, il est envisageable de construire des entrepôts de données massives reposant sur ce principe de scalabilité de l'espace de stockage. Dans ce papier, nous étudions l'instanciation d'entrepôts de données avec les systèmes orientés documents. Dans un premier temps, nous étudions les enjeux primaires des entrepôts tels que la modélisation, l'interrogation, le chargement des données et les cubes OLAP. Dans un deuxième temps, nous proposons des améliorations qui sont spécifiques aux systèmes orientés documents. En particulier, nous proposons des versions étendues des cubes OLAP qui exploitent l'imbrication. Nous montrons que ces cubes répondent plus rapidement à des charges de travail composées de requêtes OLAP de type "drill-down".*

ABSTRACT. *With the rise of large Web platforms (e.g, Google, Facebook, Twitter, Amazon), solutions have been developed recently for big data management based on decentralized approaches allowing managing and storing a large amount of data. These solutions have permitted the development of NoSQL data management systems (Not Only SQL). These NoSQL solutions allowed us to consider different responses, especially from the point of view of managing large amounts of data systems). On the one hand, we analyze several issues including modeling, querying, loading data and OLAP cuboids. We compare document-oriented models (with and without normalization) to analogous relational database models. On the other hand, we suggest improvements in order to benefit from document-oriented features. We focus particularly on extended versions of OLAP cuboids that exploit nesting and arrays. They are shown to work better on workloads with drill-down queries.*

MOTS CLÉS : NoSQL, système orienté-document, entrepôts de données big data, cuboïde OLAP système d'information.

KEYWORDS: NoSQL, document-oriented system, big data warehouse, OLAP cuboid information system.

DOI: 10.3166/dn.2017.00001

1. Introduction

Pour faciliter le processus d'aide a la prise de décision, les données a analyser sont centralisées de manière uniforme dans un entrepôt de données. Au sein de ce dernier, une analyse interactive des données est effectuée via un processus d'analyse en ligne (*On-line analytical processing* [OLAP]). Les implémentations les plus courantes d'entreposage de données, appelées approches R-OLAP (Morfonios *et al.*, 2007), reposent sur des systèmes relationnels et une philosophie de centralisation des données sur une seule machine ; or il est peu efficace, voire impossible de stocker toutes les données massives sur une seule machine (Stonebraker *et al.*, 2007 ; Stonebraker, 2012). Les mégadonnées (*big data*) sont principalement caractérisées par un important volume de données. Devant les difficultés des SGBDR à prendre en charge efficacement ces masses de données, une nouvelle génération de systèmes de stockage de données est apparue et rencontre un succès notable ; il s'agit des systèmes Not-Only-SQL, (Pavlo *et al.*, 2009 ; Cattell, 2011 ; Shute *et al.*, 2013).

Contrairement aux SGBDR, les systèmes NoSQL possèdent les caractéristiques de passage à l'échelle horizontale. On dénombre 4 modèles de systèmes NoSQL : orientés documents, orientés colonnes (Abadi *et al.*, 2007), orientés graphes et ceux orientés paires clé/valeur (Dey *et al.*, 2013). L'étude de nouvelles approches d'entreposage de données en exploitant les systèmes NoSQL devient pertinente. Certains travaux de recherche se sont intéressés aux modèles orientés colonnes et ceux orientés documents en se focalisant en général sur des questions de faisabilité et sur des implémentations spécifiques aux technologies (D'Orazio et Bimonte, 2010).

L'instanciation d'un entrepôt de données multidimensionnelles (Colliat, 1996 ; Chaudhuri et Dayal, 1997 ; Teste, 2001) avec cette nouvelle technologie n'est pas un processus facile (Cuzzocrea *et al.*, 2013). Les données doivent être extraites et transformées dans un modèle plus approprié pour les systèmes orientés documents. Les requêtes OLAP doivent être réécrites dans un langage spécifique (Ravat *et al.*, 2002), leur exécution doit être optimisée et les cubes OLAP précalculés afin de réduire le temps de réponse aux requêtes utilisateurs. La plupart des instanciations d'entrepôt de données avec les systèmes NoSQL sont des applications directes de l'approche R-OLAP. Cependant, il faut distinguer les modèles logiques NoSQL des modèles de données relationnelles. Nous avons besoin d'une formalisation explicite du modèle orienté documents et comme il n'est pas garanti que les approches d'entreposage théorisées pour le modèle relationnel (R-OLAP) (Morfonios *et al.*, 2007) fournissent les mêmes avantages dans les modèles NoSQL, nous devons également nous pencher sur les avantages propres aux NoSQL pour l'entreposage de données. Cet article étudie le potentiel des systèmes orientés documents pour l'entreposage de données multidimensionnelles.

Comme d'autres systèmes NoSQL, les systèmes orientés documents sont connus pour la flexibilité du schéma facilitant la gestion de données hétérogènes. Les systèmes orientés documents supportent des structures plus riches (imbrication, tableaux, etc.) et des traitements des données parallélisés conformément au paradigme Map-Reduce (Dean et Ghemawat, 2010 ; Vajk *et al.*, 2013 ; Mior, 2014 ; Kanade *et al.*, 2014).

Dans ce contexte, nous présentons de nouvelles investigations pour la mise en œuvre d'entrepôts de données multidimensionnelles orientés documents (Chevalier *et al.*, 2015a). Nous étudions deux modèles de données et nous proposons deux extensions de cubes OLAP qui peuvent être implémentés en utilisant des documents. Les contributions de cet article sont résumées comme suit :

– nous instancions des entrepôts de données multidimensionnelles dans les systèmes orientés documents en utilisant deux modèles logiques orientés documents différents équivalents au stockage de données normalisé et dénormalisé. Il est connu que certains systèmes NoSQL supportent mieux des données plates (dénormalisées), contrairement aux bases de données relationnelles. Nous montrons l'implantation directe de ces modèles à partir du modèle multidimensionnel. Les avantages de chaque modèle sont présentés, en comparant les différentes instanciations et les différentes étapes d'implantation et de construction à savoir : le chargement et l'interrogation des données, ainsi que le calcul de cubes OLAP ;

– nous proposons et étudions des versions étendues de cubes OLAP, bénéficiant de l'utilisation de l'imbrication et des tableaux. Ceux-ci offrent des capacités de navigations (*drill-down*) rapides et permettent de répondre à plusieurs requêtes (Simitsis *et al.*, 2005). Ces types de cubes ont déjà été étudiés pour des entrepôts de données (Jiang *et al.*, 2007 ; Zhao *et al.*, 2011), mais ils ne sont pas supportés en relationnel. Cependant, dans les systèmes orientés documents, ces cubes peuvent être nativement stockés ;

– nous comparons nos modèles orientés documents avec des modèles classiques relationnels. Notre objectif est d'illustrer les différences actuelles en considérant les spécificités techniques de chaque système (MongoDB et PostgreSQL) (Fotache *et al.*, 2013).

Le reste cet article s'articule comme suit. La section 2 étudie l'état de l'art du domaine. En section 3, nous formalisons le modèle conceptuel multidimensionnel de données ainsi que le modèle logique orienté documents et les définitions des règles de transformation. La section 4 présente les cuboïdes étendus. En section 5, nous présentons et discutons les résultats de nos expérimentations.

2. État de l'art

Au cours des dernières années, les systèmes NoSQL attirent une attention croissante (Abadi *et al.*, 2007 ; Floratou *et al.*, 2012 ; Schindler, 2012 ; Zhao et Ye, 2014 ; Mior, 2014). Ces systèmes de bases de données représentent une bonne alternative aux bases de données relationnelles, offrant de nouvelles fonctionnalités intéressantes, y compris de nouveaux langages d'interrogation (Not Only SQL) et des nouvelles techniques de

stockage et de traitements de données. Ces différentes solutions NoSQL ont été comparées dans des contextes différents. Dans ce qui suit, nous discutions dans un premier volet les travaux importants de comparaisons des modèles orientés documents avec des bases aux bases de données relationnelles. Le second volet de cet état de l'art s'intéresse aux problèmes de structuration et de modélisation des données dans ces modèles orientés documents.

2.1. Travaux de comparaisons et de migration

L'émergence de ces nouveaux systèmes de gestion de données NoSQL, dédiés à absorber et à traiter des données massives, a suscité un vif intérêt auprès des deux communautés, celle de l'industrie mais aussi celle de la recherche. Les premiers travaux se sont focalisés sur une comparaison de ces derniers avec les bases de données relationnelles (Nayak *et al.*, 2013 ; Chickerur *et al.*, 2015), considérées jusqu'ici comme la référence dans la gestion des données. Dans (Parker *et al.*, 2013), les auteurs comparent le temps d'exécution des requêtes OLTP (OnLine Transaction Processing) dans un système orienté documents (MongoDB) et un système relationnel (SQLServer). Dans les travaux de (Floratou *et al.*, 2012), les auteurs comparent l'exécution des requêtes dans le système orienté documents MongoDB et un système relationnel distribué et dressent les avantages et les inconvénients des architectures. Dans MongoDB, à des fins de comparaisons entre ces deux systèmes, les requêtes d'évaluation sont traduites par des traitements reposant sur le paradigme MapReduce.

En conséquence, rapidement, un intérêt est porté sur la transposition des bases de données relationnelles en systèmes NoSQL. Dans les travaux de (Schram et Anderson, 2012 ; Hsu *et al.*, 2014 ; Zhao et Ye, 2014 ; Abdullah et Zhuge, 2015 ; Zhao *et al.*, 2013 ; Mior, 2014 ; Banerjee *et al.*, 2015 ; Lee et Zheng, 2015 ; Batra *et al.*, 2016), les auteurs définissent des règles pour traduire le schéma relationnel vers une base de données NoSQL, le plus souvent MongoDB. Dans ces travaux, les règles définies dénormalisent les données et les regroupent dans la même collection pour éviter l'utilisation des jointures dans ces systèmes distribués. Les données sont ensuite migrées selon le schéma résultant. Outre les approches académiques, il existe des solutions industrielles de transformations des données Dans la solution Apache Sqoop[1], les données sont transformées depuis une base de données relationnelle vers l'espace de stockage HDFS, puis en second lieu, les données sont transformées de l'espace HDFS vers une base de données NoSQL. Bien que Sqoop assure tout le job d'importation dans les deux sens, il ne permet pas d'importer les dépendances entre les tables. Une deuxième solution est Datax. Elle permet, et avec des performances relativement intéressantes, de faire des migrations entre deux bases de données a très grande vitesse. Datax fournit des fonctionnalités pour la planification des processus d'importation. Tous les processus d'importation et d'exportation sont assurés par des plugins Datax. En

1. https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html.

revanche, et comme la solution Sqoop, les relations entre les tables ne sont pas assurées dans ces mécanismes d'échanges.

En vue de ces expériences, ces nouveau systèmes constituent une voie intéressante pour la construction des entrepôts de données multidimensionnelles capables de supporter des grandes masses de données mais ils nécessitent de revisiter les principes de la modélisation des entrepôts de données multidimensionnelles en l'occurrence ceux reposants sur des données structurées en documents.

2.2. Entrepôts orientés documents

Les systèmes orientés documents offrent des structures de données intéressantes telles que les documents imbriqués et les tableaux. Ces caractéristiques existent également dans les systèmes orientés objet et XML et ont notamment été étudiés dans le cadre de l'entreposage des données. Selon (Mahboubi *et al.*, 2009), les travaux portant sur l'entreposage des données XML peuvent être scindés en deux parties. La première partie traite la conception des données XML, que nous n'abordons pas dans cette article (Hachicha et Darmont, 2013). La seconde partie, plus proche de nos travaux, traite l'implémentation des données XML. Nous nous intéressons particulièrement aux travaux des entrepôts de données modélisées en étoile ou en constellation avec des hiérarchies strictes (Pedersen *et al.*, 1999 ; Hachicha et Darmont, 2013; Chevalier *et al.*, 2015d). Dans ce contexte, nous citons les premiers travaux qui proposent d'intégrer les documents XML dans des bases de données relationnelles (Kit, 2010). Grâce à l'approche des chemins, chaque nœud est transformé en deux enregistrements, stockés dans deux tables relationnelles différentes ; la première est dédiée aux chemins absolus tandis que la seconde est utilisée pour contenir les informations sur les nœuds. Dans la même logique, les travaux (Jensen *et al.*, 2001 ; Niemi *et al.*, 2017 ; Pedersen, 2003) proposent une approche complète pour gérer les données provenant de sources de format relationnel et de format XML. Dans les travaux de (Jensen *et al.*, 2001), les différentes données sont modélisées en UML, puis intégrées dans une base de données relationnelles. Avec le même objectif, les travaux de (Niemi *et al.*, 2017), proposent aussi une approche d'intégration des données XML mais avec un modèle de données OLAP reposant sur le langage d'interrogation MDX, souffrant de l'absence d'opérateurs OLAP. Ce choix rend difficile l'interrogation des données. Les travaux de (Pedersen, 2003), soignent ces problèmes en proposant deux nouvelles contributions, la possibilité de fédérer des collection XML et des tables relationnelles tout en utilisant, pour l'interrogation des données, des extensions du langages SQL. En revanche, cette approche engendre un coût supplémentaire, en termes de temps d'exécutions, pour associer les requêtes XPATH aux requêtes SQL.

Par ailleurs, d'autres travaux ont fait le choix d'abandonner les systèmes relationnels en utilisant uniquement des collections de données XML mais ont rapidement été confronté aux problèmes d'interrogations. En 2005, Park *et al.* (2005) introduisent la notion de cube XML où chaque élément conceptuel est représenté par un document XML. Contrairement au travaux (Niemi *et al.*, 2017), les auteurs proposent, ici, une

extension au langage MDX offrant quelques opérateurs d'agrégation inspirés de ceux utilisés dans le modèle relationnel. L'idée même est reprise dans les travaux de (Wang *et al.*, 2005), qui tentent d'offrir un cadre d'analyse plus complet couvrant aussi des analyses statistiques, mais en utilisant le langage d'interrogation XQuery, difficilement utilisable pour l'expression des requêtes (Beyer *et al.*, 2005).

En somme, les travaux s'intéressant à l'entreposage XML proposent des approches d'intégrations qui aboutissent soit sur l'utilisation des SGBDR qui soufrent du passage à l'échelle, soit sur l'utilisation des collections XML qui souffrent d'un manque d'opérateurs OLAP. En outre, ces approches n'ont pas été étudiées dans un environnement extensible et ne font pas état de nouvelles approches de modélisations du treillis avec les documents XML.

Récemment, dans un environnement distribué, de nouveaux travaux ont expérimenté des opérations OLTP et OLAP avec les systèmes NoSQL et ont montré que ces derniers sont des candidats pertinents pour l'implémentation d'entrepôts de données (Zhao *et al.*, 2011 ; Dehdouh *et al.*, 2014 ; Chevalier *et al.*, 2015a, 2015b, 2015c). Dans (Zhao *et al.*, 2011), les auteurs mettent en place un entrepôt de données sur un système de stockage orienté colonnes HBase (Zhao et Ye, 2014). Ils montrent comment instancier efficacement des cubes OLAP basés sur le paradigme Map-Reduce. Dans (Dehdouh *et al.*, 2014), les auteurs (Bosworth *et al.*, 2009) comparent un système orienté colonnes (Hive on Hadoop) avec une version distribuée d'un système relationnel (SQLServer PDW) en évaluant le temps d'exécution des requêtes OLAP ; le système relationnel affiche de meilleurs résultats dans la plupart des cas. Dans (Chevalier *et al.*, 2015a,b), nous avons déjà étudié les modèles orientés documents et colonnes pour l'implantation d'entrepôts de données multidimensionnelles. Cependant, ce premier travail se focalise sur l'implantation du schéma conceptuel multidimensionnel en étoile dans le modèle logique NoSQL et n'explore pas réellement les spécificités techniques du modèle orienté documents (Tahara *et al.*, 2014).

3. Passage du modèle multidimensionnel aux documents

3.1. Modèle des données multidimensionnel pour les entrepôts de données

De manière classique, le niveau conceptuel consiste à décrire l'entrepôt de données multidimensionnelles indépendamment des technologies de stockage. Nous adoptons le modèle conceptuel multidimensionnel développé dans notre équipe qui décrit les données sous forme d'un schéma en constellation (Ravat *et al.*, 2007), (Golfarelli *et al.*, 1998). Le schéma comprend un ensemble de faits modélisant les sujets d'analyse et un ensemble de dimensions hiérarchisées modélisant les axes d'analyse (Kimball et Ross, 2013).

Définition : un **schéma en constellation**, noté S, est défini par (F^S, D^S, $Star^S$) où :

– $F^S = \{F_1, \ldots, F_n\}$ est un ensemble fini de n faits ;

– $D^S = \{D_1, \ldots, D_m\}$ est un ensemble fini de m dimensions ;

– $Star^S : F^S \rightarrow 2^{D^S}$ associe à chaque fait de F^S un ensemble de dimensions qui peuvent être utilisées pour analyser le fait.

Un schéma en étoile est limité à un seul fait, $|F^S| = 1$; il s'agit d'un cas particulier du schéma en constellation.

Un fait est constitué d'un ensemble d'attributs appelés mesures. Chaque mesure est associée à une fonction d'agrégation.

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

– N^F est le nom du fait ;

– $M^F = \{f_1(m_1^F), ..., f_V(m_V^F)\}$ est un ensemble de **mesures**, chacune associée à une **fonction d'agrégation** $f_i \in \{sum, max, min, count, avg\ldots\}$.

Une dimension est constituée d'un ensemble d'attributs représentants différents niveaux de granularité sur les données analysées (mesures). Ces attributs sont organisés en hiérarchies d'un niveau racine à un niveau général, appelé extrémité.

Définition : Une **dimension** $D_i \in D^S$ (notée de manière abusive D), est définie par (N^D, A^D, H^D) où :

– N^D est le nom de la dimension ;

– $A^D = \{a_1^D, ..., a_U^D\} \cup \{id^D, all^D\}$ est un ensemble de $u+2$ attributs de la dimension ;

– $H^D = \{H_1^D, ..., H_{h_i}^D\}$ est un ensemble de h_i hiérarchies, organisant les attributs en fonction de la granularité qu'ils représentent.

Définition : Une **hiérarchie** de la dimension D, $H_j \in H^D$, est définie par (N^{Hj}, $Param^{Hj}$, $Weak^{Hj}$) où :

– N^{Hj} est le nom de la hiérarchie ;

– $Param^{Hj} = id^D, P_1^{Hj}, ..., p_w^{Hj}, all^D$ est un ensemble ordonné de $w+2$ attributs (avec $w \leq u$) appelés **paramètres** fournissant une échelle graduée de la hiérarchie, $\forall k \in [1\ldots w]$, $P_k^{Hj} \in A^D$;

– $Weak^{Hj} : Param^{Hj} \subset 2^{A^D - param^{Hi}}$ est une fonction associant à chaque paramètre un ou plusieurs attributs représentant des informations complémentaires appelées **attributs faibles**.

Nous appelons id^D le paramètre racine tandis que all^D correspond au paramètre extrémité.

Un exemple d'un schéma conceptuel multidimensionnel en étoile est décrit sur la figure 1. Le jeu de données utilisé est celui du banc d'essai Star Schema Benchmark (SSB) (ONeil *et al.*, 2009) ; le fait *Lineorder* est observé selon les dimensions *Customer*, *Part*, *Date* et *Supplier*. Nous pouvons aussi y observer l'organisation hiérarchique des paramètres de dimensions telle que {*id, date, month, year, all*}.

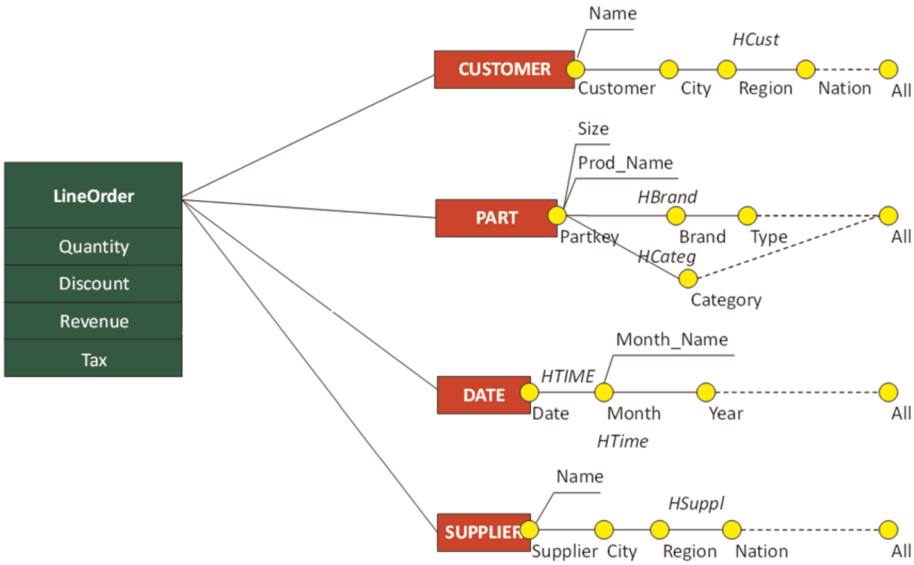

Figure 1. Exemple d'un schéma conceptuel multidimensionnel en étoile (données issues de SSB)

3.2. Modèle documents

Nous fournissons ici les définitions clés et les notations que nous utilisons pour formaliser le modèle orienté documents.

Le modèle orienté documents est constitué de paires clé/valeur. La clé identifie de manière unique une valeur structurée, appelée document. Un document est considéré comme une hiérarchie d'éléments pouvant être soit des valeurs atomiques, soit des valeurs composées (valeurs atomiques multiples ou documents imbriqués). Le niveau d'imbrication n'est pas limité. L'ensemble des documents est contenu dans une collection qui suit un stockage physique horizontal. L'ensemble des clés permettent de définir la structure du document, autrement dit le schéma du document (un document générique sans les valeurs). Dans les systèmes NoSQL orientés documents, chaque document possède son propre schéma, pouvant varier d'un document à l'autre, même au sein d'une même collection.

Dans le modèle orienté documents, un document est une structure lisible par le moteur NoSQL. Il est défini dans un format textuel balisé, généralement le format JSON (*Java script objet notation*). Il facilite la représentation de données structurées notamment d'une manière hiérarchique. Nous notons $C(id)$ le document d'identifiant *id* appartenant à la collection C ; nous utilisons le symbole « : » pour séparer une clé de sa valeur, « [] » pour désigner les tableaux, « {} » pour désigner les documents et une virgule « , » pour séparer les paires de clé/valeur.

Exemple. Le document ci-dessous appartient à la collection « *Lineorder* », son identifiant est 30001 et il contient des clés telles que « Customer », « Part », « Date ». Les valeurs de la clé *Customer* correspondent à un tableau et la valeur de *Date* correspond à un document imbriqué. Le schéma du document est donc :

{LineOrder(),

Customer: [{custkey, name,city, region, nation}],

Part: {partkey, category, brand, type},

Date: {day, month, year},

Supplier: [{supkey, name,city, region nation}]}

Une instance conforme à ce schéma est :

{LineOrder(30001),

Customer:[{custkey: "A3013", name: "IRIT", city: "Toulouse", region:"Toulouse", nation: "France"}],

Part: {partkey:"Yosim16", category:"inform", brand:"1", type:"PC"},

Date: {day: "25012017", month: "012017", year:"2017"}},

Supplier: [{supkey: "AP3009", name:"",city: "Toulouse", region:"Paris" nation: "France"}]}

3.3. Modèles logiques document pour les entrepôts de données

Au niveau logique, nous devons faire des choix de modélisations spécifiques à la technologie adoptée. Nous considérons deux modèles logiques orienté documents, le modèle DFL (*Document flat logical*) et le modèle DSH (*Document shattered logical*) ; le premier correspond à une dénormalisation totale des données tandis que le second est analogue à l'approche R-OLAP. Nous décrivons ci-dessous chacun de ces deux modèles. À titre d'illustration, nous utilisons un exemple de modèle conceptuel avec un fait nommé "*LineOrder*" et des mesures M^F = {"*quantity*", "*shipmode*", "*price*"} et une dimension "*Customer*" composée des attributs A^D = {"*c_name*", "*c_city*", "*c_nation_name*"}.

3.3.1. Le modèle DFL (Document flat logical)

Ce modèle correspond à une traduction (dénormalisation) plate où chaque fait $F \in F^S$ et ses dimensions associées $Star^S(F)$ correspondent à une collection de même nom (N^F, E^F). Chaque instance du fait est traduite par un document d_i. Les attributs représentant les mesures du fait et les attributs de dimensions sont contenus dans un même document sans imbrication.

Un document d_i est défini par un schéma constitué de la manière suivante :

– *id* est l'identifiant du document ;

– chaque mesure de M^F forme un attribut m_k^F simple ;

– chaque attribut de $\cup_{Dj \in Star^S(F)^{A^{Dj}}}$ forme un attribut a_k simple.

Le schéma S_F de la collection C^F est :

$$S_F = \left\{ id_F, m_1, m_2, \ldots m_{|M^F|}, a_1^{D_1}, a_2^{D_1}, \ldots a_{|A^{D_1}|}^{D_1}, a_1^{D_2}, a_2^{D_2}, \ldots a_{|A^{D_2}|}^{D_2}, \ldots \right\}$$

Dans notre exemple, ce schéma correspond à {id, quantity, shipmode, price, c_name, c_city, c_nation_name} avec une possible instance :

{id:1,quantity: 4,shipmode: “mail”, price:400.00, c_name: “John”, c_city: “Rome”, c_nation_name: “Italy”}

3.3.2. Le modèle DSH (Document shattered logical)

Une traduction éclatée distribue les données au sein de plusieurs collections : les données issues d’un fait $F \in F^S$ sont placées dans une première collection et les données de chaque dimension associée $StarS^{(F)}$ sont placées dans des collections distinctes (une collection par dimension).

Les attributs des dimensions sont convertis en attributs simples, à plat sans imbrication et stockés dans un document de la collection dédiée, C^{D_j}. Les mesures du fait sont converties en attributs simples, à plat sans imbrication, et stockés dans un document d’une autre collection C^F. Les documents des faits contiennent également les références aux documents représentant les dimensions associées, stockées dans les collections des dimensions.

Pour chaque instance de **dimension**, un document $d_i \in C^{D_j}$ est défini par un schéma constitué de la manière suivante :

– *id* est l’identifiant du document, correspondant à la racine id^{D_j} de la dimension ;

– chaque attribut $a_k^{D_j}$ de la dimension D_j forme un attribut simple.

Pour chaque instance du **fait**, un document $d_i \in C^F$ est défini par un schéma constitué de la manière suivante :

– *id* est l’identifiant du document ;

– chaque mesure de M^F forme un attribut m_k^F simple ;

– pour chaque dimension $D_j \in Star^S(F)$ associée, un attribut simple id^{D_j} est ajouté référençant un document lié.

Le schéma S_F de la collection C^F et le schéma S_D de la collection C^D sont comme suit :

$$S_F = \left\{ id_F, m_1, m_2, \ldots m_{|M^F|}, id_{D_1}, id_{D_2}, \ldots \right\} \quad S_D = \left\{ id_D, a_1^D, a_2^D, \ldots a_{|A^D|}^D \right\}$$

Dans notre exemple, cela correspond à deux collections, une pour les documents du fait avec le schéma : {*id, quantity, shipmode, price*} et une autre correspond aux documents de dimensions avec le schéma : {*c_name, c_city, c_nation_name*}. Deux instances possibles sont comme suit :

– {id:1,quantity:4,shipmode: “mail”,price:400.0,c_id:4} ;

– {id:4,c_name: “John”, c_city: “Rome”, c_nation_name: “Italy”}.

4. Cubes OLAP étendus

Les analyses nécessitent parfois des requêtes complexes pouvant provoquer des temps de réponse importants (Harinarayan *et al.*, 1996). Une des solutions les plus courantes est la matérialisation des vues (précalcul des résultats). L'ensemble des résultats est appelé treillis d'agrégats précalculés.

Dans cette section, nous rappelons les définitions de base des cubes OLAP et ensuite, nous présentons des versions étendues de ces derniers qui exploitent les propriétés des documents.

4.1. Définition d'un cube OLAP

Comme illustré sur la figure 2, un cube peut être défini par un treillis d'agrégats précalculés correspondant à un ensemble de cuboïdes, chacun étant un sous ensemble de mesures d'un fait $F \in F^S$, agrégées en fonction d'un sous-ensemble de paramètres des dimensions $Star^S(F)$ associées au fait, à raison d'un paramètre par dimension.

Un cuboïde correspond à une requête d'analyse. Les cuboïdes sont précalculés et stockés à l'avance pour accélérer le temps de réponse aux requêtes en anticipant le calcul du résultat. Typiquement, les données des mesures sont regroupées selon un sous-ensemble de dimensions et des fonctions d'agrégation sont utilisées pour agréger les données des mesures selon les groupes créés.

Définition : un **cube OLAP** est défini par un treillis d'agrégats construit à partir d'un fait $F \in F^S$. Ce cube OLAP noté L est défini par (V^L, E^L) où :

– V^L est un ensemble de pré-agrégats appelés cuboïdes ;

– E^L est un ensemble d'arcs reliant les cuboïdes (v_i, v_j) lorsque v_j est calculable à partir de v_i.

Ordre partiel : les cuboïdes peuvent être considérés comme un ensemble partiellement ordonné où l'opérateur d'ordre partiel $\prec$ peut être appliqué aux attributs des dimensions. Ainsi, nous pouvons dire que $\{a\}^c \prec \{b\}$ lorsque a et b sont deux attributs d'une hiérarchie de la même dimension et a est plus haut que b dans l'organisation hiérarchique où a regroupe les valeurs de b (défini par la liste ordonnée Param^H). Par exemple, $\{\text{Month}\} \succ \{\text{Day}\}$.

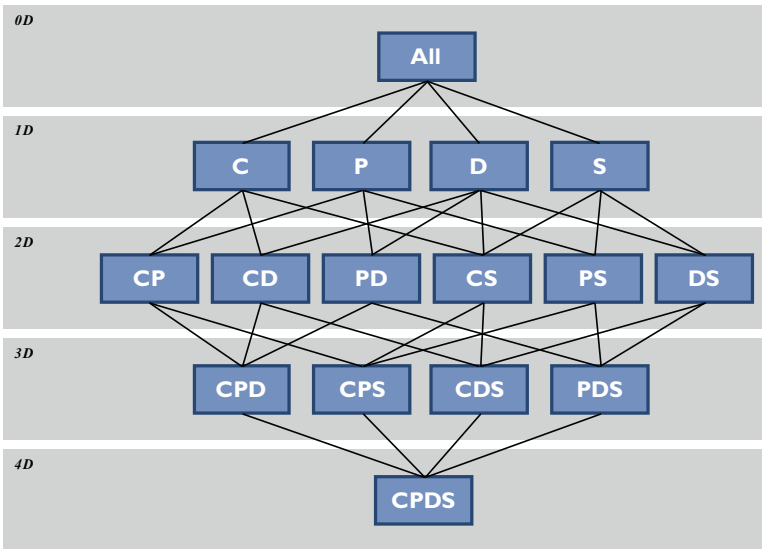

Figure 2. Exemple de cube OLAP par treillis de cuboïdes (ou préagrégats). Les lettres représentent les initiales des dimensions de la figure 1 : C : customer ; P : part ; D : date ; S : supplier

Définition : un **cuboïde** v_i dans un treillis L est défini par $\left(Pi, {}^{c}M_i^F, [V_i]\right)$ où :

– $A_i = \left\{\forall D_j \in Star^S(F), p_k^{D_j}\right\}$ est un ensemble de paramètres issus de chaque dimension $Star^S(F)$ avec $k < m$ (étant le nombre de dimensions) à raison d'un attribut par dimension ;

– $M_i^F = \{f_{k1}(m_{k2}) | f_{k1} \in \{\text{sum, max, min, count}\} \text{ et } m_{k2} \in M^F\}$ est un ensemble de mesures combinées par des fonctions d'agrégation ;

– $V_i = [v_1, \ldots v_k]$ où $v_i = \left[A'_i, {M'}_i^F\right]$ avec $A'_i = \left\{\forall D_j \in Star^S(F), {a'}_k^{D_j} \middle| \left\{{a'}_k^{D_j}\right\} \prec \left\{a_k^{D_j}\right\}\right\}$.

Remarques. Dans le cas d'un cuboïde classique v_i est réduit à $v_i = \left\{A_i, M_i^F\right\}$. Dans le cas où les données imbriquées sont du niveau hiérarchique le plus fin, ***seules les mesures détaillées sont*** imbriquées, $V_i = \left[M_i^F\right]$.

4.2. Les cubes étendus

Nous proposons deux types de cubes OLAP. Les deux exploitent les propriétés des documents, notamment la possibilité d'imbriquer des sous-documents et d'utiliser des tableaux. Plus précisément, nous proposons les 2 versions de cubes OLAP suivantes.

4.2.1. Le cube OLAP imbriqué

Le cube OLAP imbriqué est une extension du cube OLAP classique. En effet, dans le modèle classique, seule la mesure agrégée est présente (il n'y a pas la possibilité de voir les mesures agrégées du niveau hiérarchique inférieur).

Nous proposons une version étendue du cuboïde où l'imbrication des données est possible. Le cuboïde imbriqué est défini comme suit. Soit les deux ensembles d'attributs (au plus un attribut par dimension) tels que $A \prec A'$. Un cuboïde imbriqué *v(A,M,[A',M'])* est une extension du cuboïde classique *v(A,T)* où les données sont groupées d'abord sur les attributs provenant de l'ensemble *A*, puis groupés sur les attributs de l'ensemble *A'*. Au niveau de chaque document, nous avons un enregistrement constitué des attributs provenant de l'ensemble *A* et des résultats d'agrégations, d'un tableau constitué des dimensions avec un niveau de granularité plus haut et les fonctions d'agrégations provenant de *M'*. Par exemple, c(*nation, sum*(*revenue*),[*city, sum*(*revenue*)]) est un cuboïde imbriqué dont l'enregistrement est :

{Country: "FRA", sum_revenue: 45.0,by_city: [

{city: "Paris", sum_revenue: 12.0},

{city: "Toulouse", sum_revenue: 13.0},

{city: "Lyon", sum_revenue: 20.0}]}

Remarque : pour plus de clarté, le nom de l'attribut composé (qui contient les valeurs agrégées du niveau inférieur de la hiérarchie) est précédé de la mention « by_ » afin de faciliter la lecture du document.

Dans cet exemple, nous pouvons observer que le cuboïde groupe les données selon l'attribut *country*, puis il imbrique les résultats agrégés selon l'attribut $city(\{city\} \prec \{country\})$. Le principal inconvénient du cuboïde imbriqué par rapport au cuboïde classique est un volume accru du fait que les documents stockent des données supplémentaires. Toutefois, cet inconvénient est compensé par plusieurs avantages pour ce cuboïde :

– il peut être utilisé pour combiner plusieurs cuboïdes en un seul ;

– il permet de reconstituer les hiérarchies des données (d'où l'avantage ci-après) ;

– il permet un forage vers le bas (*drill-down*) en utilisant directement les données du cuboïde sans faire appel aux données détaillées.

Nous avons également envisagé une variante où les données détaillées sont stockées au sein du cuboïde.

4.2.2. Le cube détaillé

Le cube OLAP détaillé est une deuxième extension du cube OLAP classique qui, comme la précédente extension, ne peut être facilement implantée dans un système relationnel dû à l'absence d'imbrication native des données.

Le cuboïde détaillé est défini comme suit. Soit *M*, un ensemble de mesures. Un cuboïde détaillé *v(A,M,[M'*]) contient les données groupées selon les attributs de l'ensemble *A* à l'image du cuboïde classique *v(A,T)*, avec en plus, l'ajout d'un tableau des valeurs des mesures provenant de l'ensemble *M'*. Par exemple*, v(country, sum (revenue),* [*id, revenue*]) est un cuboïde détaillé ayant pour enregistrement :

{Country: "FRA", sum_revenue: 45.0,detail: [{id: 15, revenue: 2.0},

{id: 18, revenue: 3.0}, …,

{id: 10048, revenue: 12.0}]}

Dans cet exemple, le cuboïde groupe les données selon l'attribut *country,* de plus, il ne stocke pas uniquement les résultats de la fonction d'agrégation *sum* mais aussi les données détaillées de *revenue* avec les *id* (dans le tableau imbriqué ayant pour clé *détail*). Cette extension devient coûteuse en termes d'espace de stockage. En revanche, comparé au modèle relationnel, elle bénéficie des avantages suivants :

– même s'il est moins performant qu'avec le modèle précédent, le forage vers le bas (*drill-down*) est facilité et une vue détaillée des données est possible ;

– le calcul d'un ensemble d'opérations est facilité : intersection, union, articles fréquemment utilisés par exemple, achats communs pour un groupe de clients donné.

Notons que ces modèles (cuboïde imbriqué et cuboïde détaillé) sont très complexes dans un environnement relationnel. Si l'utilisation d'un tableau imbriqué est possible dans les bases de données XML et dans les bases de données orientées Objet (exemple : « VArray » et « Nested Tables »), les études ne se sont pas focalisées sur la modélisation du treillis et le plus souvent sur l'intégration et les operateurs d'interrogation (cf. Section 2.2).

5. Expérimentations

Le dispositif expérimental est d'abord présenté de manière brève ici avant d'être détaillé dans les paragraphes suivants. Nous générons des données selon le modèle de données du banc d'essai SSB. Les données sont chargées dans le système de gestion de données MongoDB dans sa version 3.0, un système NoSQL orienté documents très répandu. Pour comparer avec le modèle relationnel, nous utiliserons un SGBDR PostgreSQL (version 8.4) très populaire également. Nous évaluons, au niveau de chaque système, les deux modèles : plat et normalisé. Pour chaque ensemble de données, nous évaluons un ensemble de requêtes OLAP et nous calculons le treillis de prés-agrégats avec différentes combinaisons de dimensions. Nous évaluons aussi, avec MongoDB, les cuboïdes étendus avec deux configurations, sur une seule machine et sur un cluster composé de trois machines. Les expérimentations avec PostgreSQL sont menées sur une seule machine. Le reste des détails de ces expérimentations est décrit ci-dessous.

Les données. Nous générons des données en utilisant une version étendue de Star Schéma Benchmark SSB car il est l'un des bancs d'essai les plus utilisés dans le

contexte d'aide à la décision, qui a été adapté aux systèmes NoSQL (Chevalier *et al.*, 2015c). Le SSB se base sur un modèle conceptuel multidimensionnel en étoile, composé d'un fait et de quatre dimensions. Le jeu de données proposé modélise une gestion des lignes de commandes (*Lineorder*) décrites en fonction de ses quatre axes d'analyses (*Customer, Supplier, Part, Date*). La version SSB étendue est une partie de nos précédents travaux [21]. Il permet de générer les données brutes directement dans le format JSON qui est le format optimisé pour le chargement de données dans MongoDB. Dans nos expériences, nous utilisons différents facteurs d'échelle (*sf*) tels que $sf = 1$, $sf = 10$ et $sf = 25$. Dans la version étendue, le facteur d'échelle $sf = 1$ correspond 1×10^7 enregistrements pour le fait *LineOrder*, pour $sf = 10$, nous avons 10×10^7 enregistrements et ainsi de suite.

Configurations/matériels/logiciels. Les expérimentations ont été conduites dans deux configurations. Une architecture avec un seul nœud et une architecture distribuée composée d'un cluster de trois nœuds physiques. Chaque nœud est une machine Unix (CentOS) avec un CPU 4 core i5, 8 Go de RAM, des disques de 2 To, et un réseau de 1GB/s. au niveau du cluster, chaque machine sert de nœud de calcul et de stockage. Une de ces machines joue aussi le rôle «d'orchestrer » (i.e. il assure la gestion de la communication entre les nœuds). Nous avons installé *MongoDB* V3.0 dans chaque nœud (Dede *et al.*, 2013). Dans la terminologie *MongoDB*, cette configuration correspond à trois shards (un shard par machine) et l'un d'eux agit comme maître, gérant la répartition des données et des traitements (*Shardings*).

Modèles. Nous ferons référence à 4 modèles de données en fonction du système de base de données utilisé et le modèle de données logiques. Nous utilisons les abréviations DFL et DSH pour les modèles de documents normalisés plats et éclatés respectivement. Par analogie, nous évaluons les modèles de données plat et normalisé (R-OLAP) relationnels, RFL et RSH.

5.1. Résultats du chargement des données au calcul des cuboïdes OLAP

Chargement. Sur la figure 3, nous illustrons l'espace de stockage nécessaire à chaque approche. Nous menons des comparaisons entre le modèle relationnel et le modèle orienté documents. Nous considérons les quatre implantations logiques pour transposer le schéma multidimensionnel en étoile. Pour le modèle orienté documents, nous utilisons les deux modèles plat (DFL) et éclaté (DSH). Pour le modèle relationnel, nous utilisons le modèle logique dénormalisé (RFL) et le modèle normalisé appelé RSH (analogue à l'approche R-OLAP). Nous avons fait le choix de ces deux implantations car le modèle relationnel ne supporte pas l'imbrication.

L'instanciation sur PostgreSQL a besoin de moins d'espace que MongoDB (3 à 5 fois moins). Cela peut être expliqué par le fait que les systèmes orientés documents répètent les attributs dans chaque document et que dans MongoDB les types de données sont aussi stockés explicitement. Pour stocker les données avec des modèles plats, nous avons besoin d'environ quatre fois plus d'espace dû la redondance de données. Par exemple, pour un $sf = 10$ (10^8 documents), nous avons besoin respectivement de

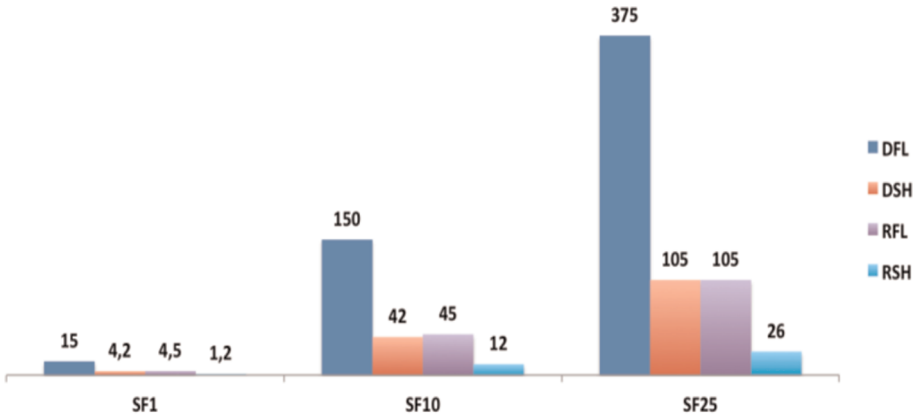

Figure 3. Utilisation de mémoire (Go) selon le modèle et scale factor

150 Go et 42 Go pour DFL (documents plats) et DSH (implantation éclatée orienté documents) et de 45 Go et de 12 Go pour le RFL (relationnel plat) et RSH (relationnel normalisé).

Sur la figure 4, nous rapportons le temps de chargement par modèle. Le modèle dénormalisé est chargé environ 23 % à 26 % plus vite avec PostgreSQL (implantation RFL) qu'avec MongoDB (DFL). En revanche, avec l'approche normalisée les données sont chargées environ 35–40 % plus vite dans MongoDB (document plat) que dans PostgreSQL (relationnel plat). Cette dernière observation est expliquée par le fait que dans PostgreSQL, la contrainte des clés étrangères ralentit significativement le processus de chargement. Nous pouvons observer que MongoDB a un rythme environ de 18 MB/s à 21 MB/s tandis que PosgreSQL charge environ 3 MB/s à 8 MB/s. Ceci explique pourquoi le temps de chargement pour les modèles de données dénormalisées

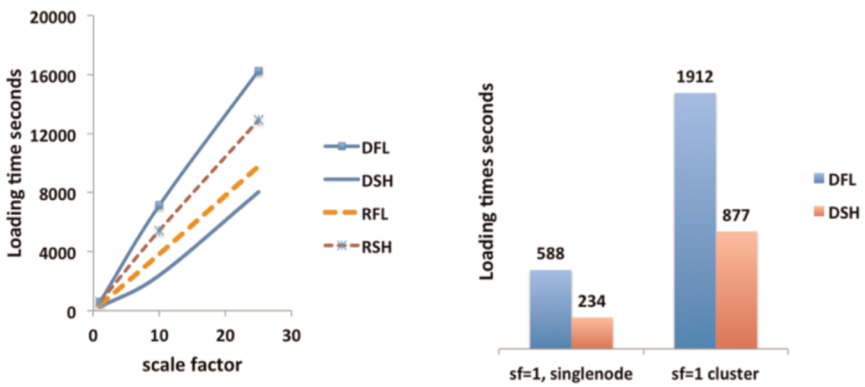

Figure 4. (Gauche) Temps de chargement selon le modèle et le scale factor. (Droite) Temps de chargement sur un seul nœud versus mode cluster avec sf = 1

est comparable dans tous les systèmes, bien que MongoDB nécessite environ 4 fois plus d'espace de stockage.

Nous montrons les temps de chargement dans les deux configurations : avec un seul nœud et avec un cluster avec un facteur d'échelle *sf = 1*. Le chargement de données est alors plus lent dans une architecture distribuée. Par exemple, les données (*sf = 1*) sont chargées dans le modèle DFL en 588 s, alors que cela nécessite 1912 s dans la configuration distribuée (environ 3 fois plus). Ceci est principalement dû à la pénalisation liée au réseau de transfert de données : MongoDB équilibre les volumes de données et essaie de répartir les données équitablement dans tous les shards impliquant plus de communication réseau.

Requêtes OLAP. Nous évaluons pour chaque implantation le temps d'exécution face au jeu de requêtes. Nous montrons le temps d'exécution dans les deux configurations, avec une seule machine et avec un cluster. Nous utilisons un jeu composé de 3 ensembles de requêtes fournies par le générateur SSB, chacun comporte 3 requêtes. Le degré de complexité augmente de Q1 à Q3. Q1 évalue une seule dimension, Q2 évalue deux dimensions et Q3 évalue trois dimensions. Les requêtes sont très sélectives et retournent peu de documents. Quand une requête a un niveau de sélectivité plus faible (elle retourne plus de résultats), cela peut avoir un impact significatif sur l'exécution de la requête. Un niveau de sélectivité plus élevé impacte des opérations telles que les jointures, le regroupement ainsi que le transfert réseau.

Nous donnons la moyenne d'exécution pour les requêtes avec trois variantes de requêtes. Notons que MongoDB possédant son propre langage d'interrogation, ces requêtes sont traduites par conséquent dans le langage de ce dernier.

Dans le tableau 1, nous rapportons le temps d'exécution pour le jeu de requêtes. L'exécution des requêtes dans MongoDB est plus rapide avec le modèle de données dénormalisé (DFL) comparé au modèle normalisé (DSH), cela s'explique par le coût des jointures. Toutefois, les résultats avec le modèle DSH sont meilleurs que ceux qui peuvent être obtenus par une simple traduction dans le langage de MongoDB. Nous avons amélioré l'exécution des requêtes en filtrant les données avant d'effectuer les opérations de jointures. Ce genre d'optimisation est géré par l'utilisateur en attendant la proposition d'une meilleure gestion native des jointures par le système dans les prochaines versions. Nous observons que lors des expérimentations avec PostgreSQL, nous pouvons observer que les trois ensembles de requêtes sont plus rapides avec le modèle normalisé relationnel (RSH) qu'avec le modèle relationnel dénormalisé (RFL) (environ 6 à 12 fois plus rapide). Cela n'a rien de surprenant si l'on considère que les SGBDR sont optimisés pour les jointures et le fait que nous devons charger moins de données en termes de mémoire avec le modèle RSH.

Nous observons que ces requêtes tournent plus vite sur PostgreSQL que sur MongoDB ; avec un temps d'exécution environ 15 à 22 fois plus rapide sur PostgreSQL avec le modèle de données RSH que sur MongoDB avec le modèle de données RFL. Sur ces ensembles de requêtes, PostgreSQL est montré supérieur à MongoDB. Cependant, nous avons observé que ces requêtes sont particulièrement

Tableau 1. Temps d'exécution des requêtes par modèle avec sf = 10, en secondes

	Mongo, 1 machine		PostgreSQL, 1 machine		Mongo, c*luster*	
	DFL (s)	DSH (s)	RFL (s)	RSH (s)	DFL (s)	DSH (s)
Q1.1	1317	1403	720	143	625	400
Q1.2	1448	1587	577	60	585	392
Q1.3	1001	1315	364	60	2091	388
Q1 avg	1255	1433	553	88	1100	393
Q2.1	1284	1384	399	67	364	922
Q2.2	1002	1202	397	70	373	884
Q2.3	1411	1611	443	60	361	885
Q2 avg	1232	1399	413	66	361	897
Q3.1	1438	1645	738	61	363	942
Q3.2	1442	1701	740	61	404	955
Q3.3	1127	1821	657	61	374	963
Q3 avg	1335	1722	711	61	380	953
avg	1274	1518	559	71	610	747

sélectives. En raison de stockage compact dans les systèmes relationnels, toutes les données à traiter peuvent tenir dans la mémoire principale après filtrage. Ceci est un avantage des SGBDR.

Avec MongoDB, nous observons que les délais d'exécution des requêtes sont généralement meilleurs dans une architecture distribuée. Pour beaucoup de requêtes, les temps d'exécution sont deux à trois fois plus rapides selon le cas. Dans l'architecture distribuée, l'exécution des requêtes est pénalisée par le transfert des données via le réseau, mais est améliorée par le calcul parallélisé.

OLAP et interrogation. Dans cette expérimentation, nous considérons les requêtes OLAP correspondant au calcul des cuboïdes OLAP. Ces requêtes sont plus coûteuses que celles utilisées précédemment. Nous évaluons ici des combinaisons de trois dimensions. Le temps de calcul est présenté dans le tableau 2. Les cuboïdes sont nommés avec les noms des attributs de dimensions selon lesquels les données regroupées. Par exemple, *c_city, s_city, p_brand* correspond à un cuboïde qui regroupe les données selon la ville (*c_city*) du client, la ville du fournisseur (*s_city*) et la marque de la pièce (*p_brand*).

Tableau 2. Temps d'exécution pour le calcul de cuboïdes OLAP 3-dimmensionnels

Cuboïde	DFL, 1 machine (s)	RSH, 1 machine (s)	DFL cluster (s)
c_city, s_city, p_brand	7466	9897	6480
c_city, s_city, d_date	3540	6742	2701
c_city, p_brand, d_date	4624	9302	3358
s_city, p_brand, d_date	4133	8509	3301
avg	**4941**	**8612**	**3960**

Nous rapportons uniquement les résultats pour les meilleurs modèles en termes de performances sur PostgreSQL et MongoDB, à savoir les modèles de données RSH et DFL. Ce choix est expliqué par le fait que nous avons observé que les temps d'exécution avec les deux autres modèles (RFL et DSH), sont devenus peu acceptables

Dans les résultats du tableau 2, nous pouvons noter que la situation est inversée face à cet ensemble de requêtes : les requêtes tournent deux plus fois vite dans MongoDB avec le modèle DFL (document plat) dans la configuration avec une seule machine ; les requêtes sont moins rapides sur PostgreSQL, cela est expliqué par le degré de sélectivité de ces requêtes ; le nombre de documents retournés est important dépassant la RAM du système relationnel PostgreSQL. Nous pouvons aussi noter une amélioration du temps d'exécution dans une configuration distribuée (avec un cluster). Avec ce jeu de requêtes, les données intermédiaires gardées au niveau de la RAM sont plus importantes que les données intermédiaires des requêtes Q1, Q2 et Q3. En effet, les données sont réduites en appliquant des filtres (l'équivalent des clauses WHERE dans SQL). Ensuite, les données sont regroupées (selon un nombre de dimensions allant de 0 à 2). Le résultat engendré est un sous-ensemble de données à traiter dans la RAM et quelques documents en sortie. Pour des cuboïdes avec combinaisons de trois dimensions, les données sont moins filtrées, le nombre d'enregistrements en sortie est plus important. Les données ne peuvent être toutes chargées dans la RAM ni dans MongoDB ni dans PostgreSQL. Cependant, MongoDB semble mieux supporter cette charge que PostgreSQL (tableau 3).

Cubes OLAP et interrogation. Afin de bénéficier de meilleures performances en termes de temps d'exécution, on évite d'interroger directement les données brutes, à la place, on calcule souvent les cuboïdes OLAP. Nous évaluons le temps d'exécution pour les trois ensembles de requêtes QS1, QS2 et QS3 ; et pour des raisons de comparaisons, nous considérons quatre systèmes d'entrepôts de données :

– *S0* : données générées dans MongoDB avec le modèle DFL sans cubes OLAP ;

– *S1* : données générées dans MongoDB avec le modèle DFL et cube OLAP ;

– *R0* : données générées dans PostgreSQL avec le modèle RSH sans cubes OLAP ;

Tableau 3. Temps d'exécution avec et sans cuboïdes OLAP, sf = 10

	QS1 (s)	QS2 (s)	QS3 (s)	avg (s)
S0	1255	1232	1335	1274
S1	0.1	0.1	0.2	0.1
R0	88	66	61	4.2
R1	0.1	0.3	0.5	0.3

– *R1* : données générées dans PostgreSQL avec le modèle RSH et cubes OLAP.

Le temps moyen d'exécution est rapporté dans le tableau 4, Les résultats montrent que lorsque nous utilisons des cuboïdes OLAP, l'exécution est nettement plus rapide ; plus de 10 000 fois sur MongoDB et environ 600–800 fois sur PostgreSQL. La

Tableau 4. Temps de calcul et utilisation de mémoire pour les cuboïdes OLAP selon le type et le nombre de dimensions

N° de dimensions, type de cuboïde	OLAP cuboïde exemple	Temps de calcul (s)	Mémoire utilisée (MB)
3D, classic	*c(supplier,date,part,T)*	428	4065
2D, classic	*c(supplier,date,T)*	197	86
1D, classic	*c(date,T)*	1	0,3
Average		207	1198
3D, nested	*c(supplier,date,part, T,[customer,T])*	476	4950
2D, nested	*c(supplier,date,T[part,T])*	227	1897
1D, nested	*c(date,T,[supplier,T])*	10	512
Average		189	2373
3D, detail	*c(supplier,date,part, T, [M])*	521	5917
2D, detail	*c(supplier,date,T, [M])*	251	2600
1D, detail	*c(date,T,[M])*	59	2110
average		273	3407

différence de performances entre PostgreSQL et MongoDB est faible. En effet, les résultats des requêtes sont déjà stockés dans les cuboïdes, il suffit juste de sélectionner celui qui répond à la requête. Dans certains cas, MongoDB est capable de récupérer des données agrégées des cuboïdes OLAP respectives plus rapidement que PostgreSQL, il est doté d'un bon moteur d'agrégation.

Conclusion. Dans cette section, nous avons mené une comparaison entre le modèle orienté documents (MongoDB) et le modèle relationnel (PostgreSQL). Pour chaque modèle, nous avons évalué deux approches d'implantation du schéma conceptuel multidimensionnel en étoile, une approche éclatée où les données sont distribuées sur plusieurs ensembles (plusieurs collections pour le modèle orienté documents et plusieurs tables pour le modèle relationnel) et l'implantation plate où les données sont stockées à plat dans un seul ensemble (une seule collection pour le modèle orienté documents et une seule table pour le modèle relationnel).

Deux jeux de requêtes sont employés : un jeu de requêtes analytiques et un jeu pour la construction du treillis d'agrégats. Comme attendu, l'approche R-OLAP (éclaté relationnel) est moins coûteuse en termes de stockage d'une part, et d'autre part, le temps d'interrogations est aussi meilleur que l'approche éclatée du modèle orienté documents lorsqu'il s'agit de requêtes ne dépassant pas la taille de la RAM. En revanche, la tendance s'inverse avec les requêtes moins sélectives de la construction du treillis d'agrégats, car les données retournées dépassent généralement la taille de la RAM. C'est dans ce cas de figures que le modèle relationnel est mis en difficulté. Le temps est encore meilleur dans une configuration composée du cluster (pour le modèle éclaté orienté documents).

Nous pouvons dire que les deux technologies sont complémentaires et qu'il reste pertinent d'utiliser l'approche R-OLAP pour un volume de données raisonnable. L'implantation orienté documents doit être motivée par un gros volume de données.

5.2. Résultats cubes OLAP étendus

Dans cette partie, nous nous focalisons sur les cuboïdes étendus définis précédemment. Nous étudions leur calcul et leur utilité sur deux ensembles de requêtes différents. Nous avons calculé des cuboïdes imbriqués et détaillées avec différentes combinaisons de dimensions. Nous utilisons le système orienté documents (MongoDB v3.0) et des données brutes générées selon le modèle de données DFL. Au cours des expériences, nous avons dû remédier à la limite de la taille du document (16 Mo) imposée par MongoDB. En effet, les enregistrements de cuboïdes imbriqués ou détaillés dépassaient parfois la taille limite du document. Dans ce cas, nous avons découpé le document en plusieurs parties (chunks).

Utilité des cuboïdes étendus. Pour évaluer l'utilité de cuboïdes imbriqué et détaillé, nous avons conçu deux expériences avec des charges de requêtes qui incluent des requêtes navigationnelles de type *drill-down* et des requêtes nécessitant le niveau de détail le plus fin. Nous considérons 3 types de systèmes d'entrepôt de données :

– *S1* : nous avons des données brutes et des cuboïdes classiques sur toutes les combinaisons de dimensions ;

– *S2* : nous avons des données brutes et des cuboïdes imbriqués sur toutes les combinaisons de dimensions ;

– *S3* : nous avons des données brutes et des cuboïdes détaillés sur toutes les combinaisons de dimensions.

Remarque. Pour tous les systèmes utilisés, nous gardons le résultat de la dernière requête pour des besoins de forage (*drill-down*).

Pour réaliser cette seconde partie d'expérimentations, nous disposons de trois ensembles de requêtes.

Jeu de requêtes 1. On considère un jeu de requêtes composé de requêtes OLAP suivies de requêtes *drill-down*, une séquence typique dans les analyses décisionnelles en ligne (OLAP). Par exemple, nous commençons l'analyse des données par une agrégation sur le pays avant d'affiner les analyses sur des villes spécifiques. Plus précisément, nous considérons un jeu de 40 requêtes, constitué de 8 requêtes OLAP classiques tirées du banc d'essais SSB+ et 32 requêtes de type *drill-down* que nous construisons nous-mêmes :

– $QS_1 = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}, Q_{40}\}$ est le jeu de requêtes utilisé ;

– $QT_1 = \{Q_1, Q_6, Q_{11}, Q_{16}, Q_{19}, \ldots Q_{36}\}$ est l'ensemble des requêtes OLAP classiques.

Chaque requête de l'ensemble QT_1 est suivie de 4 requêtes $\{Q_{i+1}, Q_{i+2}, Q_{i+3}, Q_{i+4}\}$, tirées aléatoirement, avec un niveau de granularité plus élevé, c'est-à-dire avec des *drill-down* (exemple : pays → ville).

Jeu de requêtes 2. Dans le second ensemble, nous constituons un jeu de 40 requêtes, sur le même principe que l'ensemble *QS1,* mais avec des requêtes de détail. Nous modifions l'enchaînement et le nombre de requêtes de type *drill-down* :

– $QS = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}, \ldots Q_{40}\}$ est l'ensemble des requêtes utilisées ;

– $QD = \{Q_5, Q_{10}, Q_{15}, Q_{20}, Q_{25}, \ldots\}$, est l'ensemble des requêtes avec le niveau de granularité le plus fin, niveau détail ;

– *queries* = $\{Q_1, Q_2, Q_3, Q_4, Q_6, Q_7, Q_8, Q_9, Q_{11}, Q_{12} \ldots\}$ est l'ensemble de requêtes OLAP classiques.

Jeu de requêtes 3. Dans la même logique que les deux ensembles précédents, nous constituons un jeu de 40 requêtes de type *drill-down*, 32 requêtés de forage sur le niveau de granularité le plus fin et 8 requêtes de forage restreintes au niveau inférieur ayant permis le calcul de l'agrégat :

– $QI = \{Q_2, Q_3, Q_4, Q_5, Q_7, Q_8, Q_9, Q_{10}, \ldots Q_{40}\}$ est l'ensemble des requêtes avec le niveau de granularité inférieur direct ;

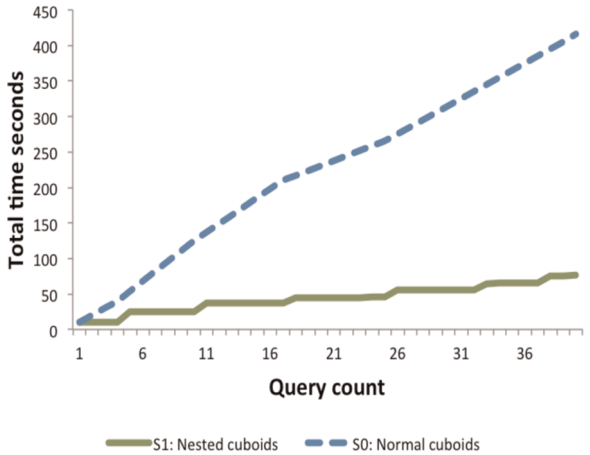

Figure 5. Les cuboïdes classique et imbriqué comparés sur un jeu de requêtes

– QD = {Q_1, Q_6, Q_{11}, Q_{16}, Q_{21}, …}, est l'ensemble des requêtes avec le niveau de granularité le plus fin.

Chaque requête de forage sur le niveau de granularité inférieur direct est suivie de quatre requêtes de détails, sur le niveau le plus fin.

Sur la figure 5, nous comparons le comportement des deux types de cuboïdes classiques et imbriqués en utilisant le premier ensemble de requêtes QS_1. Nous observons que le cuboïde imbriqué offre de meilleures performances par rapport au système classique, notamment lorsqu'il s'agit de requêtes nécessitant un forage vers le bas (*drill-down*). Par exemple, pour un jeu de 25 requêtes (incluant 20 requêtes *drill-down*), le temps d'exécution est d'environ 30 secondes pour le cuboïde imbriqué alors que le cuboïde classique atteint les 140 secondes, soit 4 fois plus. Cette différence est d'autant plus importante quand le nombre de requêtes augmente puisque nous observons un temps d'exécutions 5 fois meilleur pour une charge de 35 requêtes, soit environ 50 secondes pour le cuboïde imbriqué contre environ 360 secondes pour le système classique.

L'avantage du cuboïde imbriqué réside donc dans sa capacité à supporter des requêtes nécessitant un forage vers le bas. L'imbrication des données de l'agrégat diminue le temps de réponse puisqu'il n'est pas nécessaire de solliciter un niveau de granularité inférieur. Autrement dit, le système n'a pas besoin de consulter d'autres documents. Le fonctionnement est plus complexe pour le cuboïde classique qui doit solliciter le cuboïde du niveau le plus fin pour récupérer les données du calcul (jointures) sur un nombre de documents bien plus important. Le cuboïde détaillé, quant à lui, se montre plus adapté face à des requêtes qui nécessitent une navigation vers les données du niveau de granularité le plus fin. Le choix de cuboïde doit être acté pat le type d'analyses à effectuer.

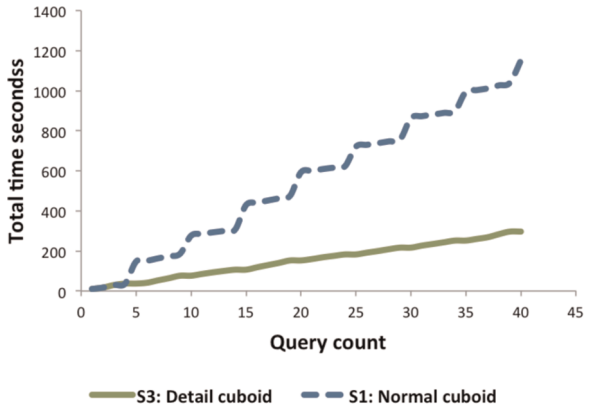

Figure 6. Les cuboïdes classique et détaillé comparés sur un jeu de requêtes

Sur la figure 6, nous évaluons le temps d'exécution pour le cuboïde détaillé et le cuboïde classique face à la seconde charge de requêtes. Dans cette figure, nous observons que le cuboïde détaillé permet d'obtenir de bien meilleurs temps d'exécution en comparaison du système classique. Nous pouvons noter, que le cuboïde classique souffre face à des requêtes de type *drill-down* vers le niveau le plus détaillé.

Sur la figure 7, nous comparons le comportement des deux cuboïdes imbriqué et détaillé. Le cuboïde détaillé offre de meilleurs temps d'exécutions lorsqu'il s'agit de requêtes en détails, nécessitant un forage vers le niveau le plus fin. Le cuboïde imbriqué supporte moins ces requêtes de détails mais reste aussi efficace, face a des requêtes qui nécessitent un forage vers les données du niveau inférieur ayant permis le calcul de l'agrégat, comme le cas pour les requêtes 1, 6, 11, 16 et 21 par exemple.

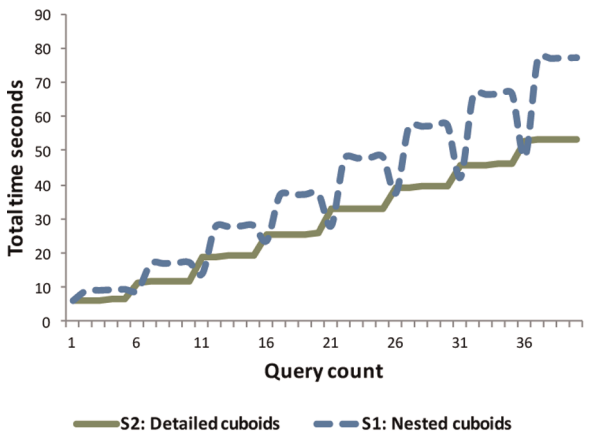

Figure 7. Les cuboïdes imbriqué et détaillé comparés sur un jeu de requêtes

À l'image du cuboïde imbriqué, le cuboïde détaillé supporte efficacement les requêtes avec une vision sur les données détaillées. Stocker les données brutes diminue significativement le temps de réponses contrairement au cuboïde classique mais aussi le cuboïde imbriqué. En revanche, les deux cuboïdes étendus offrent des résultats très proches quand les requêtes concernent des données imbriquées. Le choix de l'un ou de l'autre dépendra des analyses. L'espace de stockage, dont le coût n'est plus un frein depuis les années 2000 (Jacobs, 2009), est aussi un élément qui a son impact sur le choix de la solution et que nous évaluons dans l'expérience suivante.

Dans le tableau 4, nous rapportons les résultats de cette expérimentation. Nous comparons le temps de calcul et l'espace disque nécessaire pour chacun des trois types de cuboïdes étudiés : classique, imbriqué et détaillé. Nous pouvons observer un premier résultat évident : les deux cuboïdes étendus nécessitent plus d'espace disque en comparaison avec le cuboïde classique : ce coût s'explique par l'imbrication des données détaillées ayant permis le calcul de l'agrégat. En effet, celle-ci devient plus importante quand le nombre de dimensions augmente. Par exemple dans la requête ci-dessous, les données agrégées de chaque ville sont imbriquées par le calcul d'agrégat par pays.

6. Conclusion

Dans cet article, nous avons étudié l'instanciation d'entrepôts de données multidimensionnelles avec des systèmes NoSQL orientés documents. Nous avons formalisé et analysé deux modèles logiques. Notre étude met en évidence les faiblesses et les points forts de chacun des modèles ; nous avons étudié les cubes OLAP étendus et comparé les performances avec un SGBDR.

Nous avons montré comment instancier un entrepôt de données multidimensionnelles avec deux approches : modèle dénormalisé (DFL) et modèle normalisée (DSH). Nous avons étudié le chargement de données à différents facteurs d'échelle (10^7 enregistrements de faits – 2.5×10^8 enregistrements de faits). Le modèle en étoile normalisé (DSH) nécessite moins d'espace et les données sont chargées plus rapidement. Cependant, ce dernier modèle souffre lors des jointures de données qui ne sont pas bien prises en charge par les systèmes orientés documents. D'autre part, le modèle de données dénormalisé montre de meilleures performances d'interrogation et les requêtes sont plus simples à écrire.

Le système orientés documents a été comparé avec le SGBDR sur deux modèles de données équivalents. Les résultats montrent que RDBMS est plus rapide sur l'interrogation de données brutes, mais la performance diminue rapidement lorsque les données ne tiennent pas dans la RAM. Le système orienté documents analysé est alors montré plus robuste, c'est-à-dire qu'il ne présente pas de baisse de performance significative avec le passage à l'échelle. De plus, il bénéficie de la distribution des données. Ceci est un avantage évident par rapport aux SGBDR qui souffre de la montée en charge et de la distribution horizontale ; ils ont une taille maximale de base de données inférieure à celle des systèmes NoSQL. Les délais d'interrogation sont réduits

de façon considérable lorsque nous interrogeons directement sur les cuboïdes OLAP. Cela permet une comparaison entre les SGBDR et les systèmes orientés documents quant aux performances des requêtes OLAP.

Dans la deuxième phase d'expérimentations, nous avons comparé le cuboïde classique avec les deux nouveaux cuboïdes étendus. Les cuboïdes imbriqué et détaillé offrent de meilleures performances par rapport au cuboïde classique grâce à la possibilité d'imbriquer les données récupérées directement du résultat de la requête du niveau inférieur direct. Le modèle classique, quant à lui, doit interroger le niveau de détail le plus fin lorsqu'il s'agit d'une requête qui demande des données détaillées.

Ces deux cuboïdes, imbriqué et détaillé, permettent aux décideurs d'avoir une vision plus détaillée des données du calcul sans devoir consulter les niveaux inférieurs. En revanche, en termes de stockage, ces deux nouveaux cuboïdes sont coûteux par rapport au cuboïde classique. Ils nécessitent une mémoire disque plus importante mais dans un contexte facilement extensible, ou le coût du stockage est en chute libre (Jacobs, 2009), cela reste une option intéressante.

De ce constat, il est évident que les deux types cuboïdes n'offrent pas de meilleures performances au niveau de tous les critères mais ils se prêtent à des cas d'utilisations bien particuliers à l'image des nouvelles technologies NoSQL qui ont été développées pour des besoins internes précis. Ils offrent ainsi une vision plus complète sur les agrégats précalculés afin de supporter des analyses OLAP faisant intervenir des requêtes de forages.

Plusieurs perspectives sont envisagées. Dans les systèmes NoSQL, le choix de l'implantation est plutôt déterminé en fonction du traitement à effectuer. Or, comme le traitement peut évoluer, il devient difficile d'avoir une implantation capable de supporter tous les traitements, en l'occurrence, des cuboïdes supportant toutes requêtes. Nous envisageons donc de définir dés règles de conversions qui permettent, en fonction de l'évolution des traitements, de passer d'un cuboïde à l'autre. Nous réfléchissons également à la mise en place d'une plateforme supportant plusieurs cuboïdes mais aussi plusieurs systèmes NoSQL. De cette façon, nous pouvons utiliser le meilleur système NoSQL au traitement envisagé ; mais déterminer le meilleur système nécessite une matrice d'évaluation, une architecture de type *Lambda* pourrait être utilisée à cet effet.

Bibliographie

Abadi D.J., Myers D.S., DeWitt D.J., Madden S.R. (2007). Materialization strategies in a column-oriented DBMS. In *IEEE 23rd ICDE, IEEE*, p. 466-475.

Abdullah A., Zhuge Q. (2015). Research article from relational databases to NoSQL databases: performance evaluation. *Research Journal of Applied Sciences, Engineering and Technology*, vol. 11, n° 4, p. 434-439, ISSN: 2040-7459; e-ISSN: 2040-7467.

Batra S., *et al*. (2016). MONGODB versus SQL: a case study on electricity data. In: *Emerging research in computing, information, communication and applications*. Springer, Singapore. p. 297-308.

Banerjee S., Shaw R., Sarkar A., Debnath N.C. (2015). *Towards logical level design of big data*, 2015 IEEE 13th (INDIN). Cambridge, p. 1665-1671.

Beyer K.S., Chamberlin D.D., Colby L.S., Ozcan F., Pira-hesh H., Xu Y. (2005). Extending XQuery for analytics. In: *24th ACM SIGMOD International conference on management of data (SIGMOD 05)*. Baltimore, Maryland, USA. p. 503-514.

Bosworth A., Gray J., Layman A., Pirahesh H. (2009). Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *ACM*, vol. 52, n° 8, p. 36-44.

Chickerur S., Goudar A., Kinnerkar A. (2015). *Comparison of relational database with document-oriented database (MongoDB) for big data applications*, 2015 8th (ASEA), Jeju. p. 41-47.

Cattell R. (2011). Scalable SQL and NoSQL data stores. SIGMOD record. *ACM*, vol. 39, n° 4, p. 12-27.

Chaudhuri S., Dayal U. (1997). An overview of data warehousing and OLAP technology. SIGMOD record. *ACM*, vol. 26, n° 1, p. 65-74.

Chevalier M., El Malki M., Kopliku A., Teste O., Tournier R. (2015). Implementation of multidimensional databases in column-oriented NoSQL systems. *19th East European conference advances in databases and information systems (ADBIS) 2015*, Poitiers, France, September 8-11, Springer LNCS 9282. p. 79-91.

Chevalier M., El Malki M., Kopliku A., Teste O., Tournier R. (2015). Benchmark for OLAP on NoSQL technologies. *9th IEEE International conference on research challenges in information science (RCIS) 2015*, Athens, Greece, May 13-15. p. 480-485.

Chevalier M., El Malki M., Kopliku A., Teste O., Tournier R. (2015). Not Only SQL Implementation of multidimensional database (regular paper). In: *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2015)*, Valencia, Spain, September 1-4, Springer, p. 379-390.

Colliat G. (1996). OLAP, relational, and multidimensional database systems, SIGMOD Record. *ACM*, vol. 25, n° 3, p. 64-69.

Cuzzocrea A., Bellatreche L., Song I.Y. (2013). Data warehousing and OLAP over big data: current challenges and future research directions. DOLAP*ACM*, p. 67-70.

Dede E., Govindaraju M., Gunter D., Canon R.S., Ramakrishnan L. (2013). Performance evaluation of a mongodb and hadoop platform for scientific data analysis. 4th ACM workshop on scientific cloud computing (Cloud). *ACM*, p. 13-20.

Dehdouh K., Boussaid O., Bentayeb F. (2014). *Columnar NoSQL star schema benchmark. Model and data engineering, LNCS 8748*. Springer, p. 281-288

Dean J., Ghemawat S. (2010). MapReduce: a flexible data processing tool. *Commun. ACM*, vol. 53, n° 1, p. 72-77.

Dey A., Fekete A., Uwe R. (2013). Scalable transactions across heterogeneous NoSQL Key-value data stores. *Proc VLDB Endow*, vol. 6, n° 12, vol. 1434-1439.

D'Orazio L., Bimonte S. (2010). Multidimensional arrays for warehousing data on clouds, *Proceedings of the Third International Conference on Data Management in Grid and Peer-to-peer Systems. Globe'10*, Berlin, Heidelberg, Springer-Verlag, p. 26-37.

Floratou A., Teletia N., Dewitt D., Patel J., Zhang D. (2012). Can the elephants handle the NoSQL onslaught? *Int. Conf. on VLDB, pVLDB*, vol. 5, n° 12, p. 1712-1723. VLDB Endowment.

Fotache, Marin, Cogean, Dragos. (2013). NoSQL and SQL databases for mobile applications. case study: Mongodb *versus* postgresql. *Informatica Economica*, vol. 17, n° 2, p. 41.

Golfarelli M., Maio D., Rizzi S. (1998). The dimensional fact model: a conceptual model for data warehouses. *Int, Journal of Cooperative Information Systems*, vol. 7 n° 2–3, p. 215-247. World Scientific.

Hachicha M., Darmont J. (2013). Problèmes d'additivité dus à la présence de hiérarchies complexes dans les modèles multidimensionnels : définitions, solutions et travaux futurs, *9e journées francophones sur les Entrepôts de données et l'analyse en ligne (EDA 2013)*, Hermann, p. 7-16.

Jiang F.M., Pei J., Fu A.W. (2007). Ix-cubes: iceberg cubes for data warehousing and olap on xml data.*Conf. on CIKM*, ACM, p. 905-908.

Hsu T.-C., Chang D.-M., Lee H.-J. (2014). The study of application and evaluation with NoSQL databases in cloud computing. Trustworthy systems and their applications (TSA). *2014 International conference on IEEE*, p. 57-62.

Jacobs A. (2009). The pathologies of big data. *Communications of the ACM*, vol. 52, n° 8, p. 36-44.

Jensen M.R., Møller T.H., Pedersen T.B. (2001). Specifying OLAP cubes on XML data. *Journal of Intelligent Information Systems*, vol. 17, n° 2–3, p. 255-280.

Kanade A., Gopal A., Kanade S. (2014). A study of normalization and embedding in MongoDB. *IEEE Int. Advance Computing Conf. (IACC), IEEE*, p. 416-421.

Kimball R., Ross M. (2013). *The data warehouse toolkit: the definitive guide to dimensional modeling*, 3rd ed. John Wiley & Sons.

Kit C. (2010). *A study on OLAP analysis in XML databases*, thèse de doctorat, University of Tsukuba Japan.

Lee C.-H., Zheng Y.-L. (2015). SQL-to-NoSQL schema denormalization and migration: a study on content management systems (SMC), *2015 IEEE International conference on IEEE*, p. 2022-2026.

Mahboubi H., Ralaivao J.C., Loudcher S., Boussaid O., Bentayeb F., Darmont J. (2009). X-WACoDa: an XML-based approach for warehousing and analyzing complex data. In: Bellatreche L. ed. *Data warehousing design and advanced engineering applications: methods for complex construction*. Hershey, PA, USA, IGI Publishing, p. 38-54.

Mior M.J. (2014). Automated schema design for NoSQL databases. *Proceedings of the 2014 SIGMOD PhD symposium (SIGMOD'14 PhD Symposium)*. ACM, New York.

Morfonios K., Konakas S., Ioannidis Y., Kotsis N. (2007). ROLAP implementations of the data cube. *ACM Comput. Surv.*, vol. 39, n° 4, article 12.

Nayak A., Poriya A., Poojary D. (2013). *Type of NOSQL databases and its comparison with relational databases (IJAIS)*. Foundation of Computer Science FCS, New York, USA, vol. 5, n° 4.

Niemi T., Niinimaki M., Nummenmaa J., Thanisch P. Constructing an OLAP cube from distributed XML data, *5th ACM international workshop (DOLAP '02)*. ACM, New York, NY, USA, p. 22-27.

ONeil P., ONeil E., Chen X., Revilak S. (2009). *The Star Schema Benchmark and augmented fact table indexing*. Springer, p. 237-252, LNCS 5895.

Park B.K., Han H., Song I.Y. (2005). XML-OLAP:a multidimensional analysis framework for XML warehouses. *7th International Conference on (DaWaK 05)*. Copenhagen, Denmark, Springer, p. 32-42, volume 3589 de LNCS.

Parker Z., Poe S., Vrbsky S.V. (2013). Comparing NoSQL MongoDB to an SQL DB.*Proceedings of the 51st ACM Southeast Conference (ACMSE '13)*. ACM, New York, NY, USA, article 5, 6 p.

Pavlo A., Paulson E., Rasin A., Abadi D.J., DeWitt D.J., Madden S., Stonebraker M. (2009). A comparison of approaches to large-scale data analysis.*Int. Conf. on Management of data (SIGMOD)*. ACM, p. 165-178.

Pedersen T.B., Jensen C.S., Dyreson C.E. (1999). Extending practical pre-aggregation in on-line analytical processing. *VLDB'99*. Morgan Kaufmann, Edinburgh, UK, p. 663-674.

Pedersen T.B. (2003). Achieving adaptivity for OLAP-XML federations.*6th International ACM Workshop on Data Warehousing and OLAP (DOLAP 03)*, New Orleans, USA, ACM, p. 25-32.

Ravat F., Teste O., Zurfluh G. (2002). Langage pour bases multidimensionnelles : OLAP-SQL. *Ingénierie des Systèmes d'Information*, vol. 7, n° 3, p. 11-38.

Ravat F., Teste O., Tournier R., Zurfluh G. (2007). Graphical querying of multidimensional databases. *11th East European Conference Advances in Databases and Information Systems (ADBIS)*, Varna, Bulgaria, September 29-October 3, Springer LNCS 4690, p. 298-313.

Schindler A. (2012). I/O characteristics of NoSQL databases. *Int. Conf. on Very Large Data Bases (VLDB), pVLDB*, vol. 5, n° 12, p. 2020-2021, VLDB Endowment.

Schram A., Anderson K.M. (2012). MySQL to NoSQL: data modeling challenges in supporting scalability. *Proceedings of the 3rd annual conference (SPLASH '12)*, ACM, New York, NY, USA, p. 191-202.

Shute J., Vingralek R., Samwel B., Handy B., Whipkey C., Rollins E., Oancea M., Littlefield K., Menestrina D., Ellner S., Cieslewicz J., Rae I., Stancescu T., Apte H. (2013). F1: a distributed SQL database that scales. *Int. Conf. On VLDB*, vol. 6, n° 11, p. 1068-1079.

Simitsis A., Vassiliadis P., Sellis T. (2005). *Optimizing etl processes in data warehouses. Data Engineering*. ICDE, p. 564-575.

Stonebraker M., Madden S., Abadi D.J., Harizopoulos S., Hachem N., Helland P. (2007). The eof an architectural era: (it's time for a complete rewrite), *33rd Int. conf. (VLDB), ACM*, p. 1150-1160.

Stonebraker M. (2012). New opportunities for new SQL. *Communications of the ACM*, vol. 55, n° 11.

Teste O. (2001). Towards conceptual multidimensional design in decision support systems. 5th East-European conference on advances in databases and information systems (*ADBIS*). *Research Communications*, vol. 1, p. 77-88.

Tahara D., Diamond T., Abadi D.J. (2014). Sinew: a SQL system for multi-structured data, *Int. Conf. on Management of data (SIGMOD)*, ACM, p. 815-826.

Vajk T., Feher P., Fekete K., Charaf H. (2013). Denormalizing data into schema-free databases, *4th Int. Conf. on Cognitive Infocommunications (CogInfoCom), IEEE*, p. 747-752.

Wang H., Li J., He Z., Gao H. (2005). OLAP for X ML data, *1st International Conference (CIT 2005)*, IEEE Computer Society, Shanghai, China, p. 233-237.

Zhao P., Li X., Xin D., Han J. (2011). Graph cube: on warehousing and OLAP multidimensional networks, *International conference on management of data (SIGMOD)*, ACM, p. 853-864.

Zhao H., Ye X. (2014). A practice of TPC-DS multidimensional implementation on NoSQL database systems. Performance characterization and benchmarking, LNCS 8391, Springer, p. 93-108.